[MS-RPRN]:

Print System Remote Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
2/22/2007	0.01		Version 0.01 release
6/1/2007	1.0	Major	Updated and revised the technical content.
7/3/2007	2.0	Major	Editorial changes, plus added a new section for a wsdmon monitor module for the WSD_BACKUP_PORT_DATA packet structure.
7/20/2007	2.0.1	Editorial	Changed language and formatting in the technical content.
8/10/2007	2.0.2	Editorial	Changed language and formatting in the technical content.
9/28/2007	2.1	Minor	Clarified the meaning of the technical content.
10/23/2007	2.2	Minor	Clarified the meaning of the technical content.
11/30/2007	2.2.1	Editorial	Changed language and formatting in the technical content.
1/25/2008	2.3	Minor	Clarified the meaning of the technical content.
3/14/2008	3.0	Major	Updated and revised the technical content.
5/16/2008	4.0	Major	Updated and revised the technical content.
6/20/2008	5.0	Major	Updated and revised the technical content.
7/25/2008	5.1	Minor	Clarified the meaning of the technical content.
8/29/2008	6.0	Major	Updated and revised the technical content.
10/24/2008	7.0	Major	Updated and revised the technical content.
12/5/2008	8.0	Major	Updated and revised the technical content.
1/16/2009	8.1	Minor	Clarified the meaning of the technical content.
2/27/2009	9.0	Major	Updated and revised the technical content.
4/10/2009	10.0	Major	Updated and revised the technical content.
5/22/2009	11.0	Major	Updated and revised the technical content.
7/2/2009	12.0	Major	Updated and revised the technical content.
8/14/2009	12.1	Minor	Clarified the meaning of the technical content.
9/25/2009	12.2	Minor	Clarified the meaning of the technical content.
11/6/2009	12.3	Minor	Clarified the meaning of the technical content.
12/18/2009	13.0	Major	Updated and revised the technical content.
1/29/2010	13.1	Minor	Clarified the meaning of the technical content.
3/12/2010	13.2	Minor	Clarified the meaning of the technical content.
4/23/2010	14.0	Major	Updated and revised the technical content.
6/4/2010	14.1	Minor	Clarified the meaning of the technical content.
7/16/2010	15.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
8/27/2010	15.1	Minor	Clarified the meaning of the technical content.
10/8/2010	16.0	Major	Updated and revised the technical content.
11/19/2010	16.1	Minor	Clarified the meaning of the technical content.
1/7/2011	16.2	Minor	Clarified the meaning of the technical content.
2/11/2011	17.0	Major	Updated and revised the technical content.
3/25/2011	18.0	Major	Updated and revised the technical content.
5/6/2011	18.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	18.1	Minor	Clarified the meaning of the technical content.
9/23/2011	19.0	Major	Updated and revised the technical content.
12/16/2011	20.0	Major	Updated and revised the technical content.
3/30/2012	21.0	Major	Updated and revised the technical content.
7/12/2012	22.0	Major	Updated and revised the technical content.
10/25/2012	23.0	Major	Updated and revised the technical content.
1/31/2013	24.0	Major	Updated and revised the technical content.
8/8/2013	25.0	Major	Updated and revised the technical content.
11/14/2013	26.0	Major	Updated and revised the technical content.
2/13/2014	26.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	26.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	27.0	Major	Significantly changed the technical content.
10/16/2015	27.0	No Change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This is a specification of the Print System Remote Protocol. It is based on the **Remote Procedure Call (RPC)** protocol, as specific in [C706] and [MS-RPCE].

The Print System Remote Protocol supports synchronous printing and spooling operations between a client and server, including **print job** control and **print system** management. An enhanced replacement for this protocol is specified in [MS-PAR].

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. Sections 1.5 and 1.9 are also normative but do not contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are specific to this document:

3D printer: A **device** that constructs a physical, three-dimensional object from a digital model.

access control entry (ACE): An entry in an access control list (ACL) that contains a set of user rights and a **security identifier (SID)** that identifies a principal for whom the rights are allowed, denied, or audited.

access level: The type of access that the client requests for an object, such as read access, write access, or administrative access.

Active Directory: A general-purpose network **directory service**. **Active Directory** also refers to the Windows implementation of a **directory service**. **Active Directory** stores information about a variety of objects in the network. Importantly, user accounts, computer accounts, groups, and all related credential information used by the Windows implementation of Kerberos are stored in **Active Directory**. **Active Directory** is either deployed as Active Directory Domain Services (AD DS) or Active Directory Lightweight Directory Services (AD LDS). [MS-ADTS] describes both forms. For more information, see [MS-AUTHSOD] section 1.1.1.5.2, **Lightweight Directory Access Protocol (LDAP)** versions 2 and 3, Kerberos, and **DNS**.

application server mode: A mode in which **Terminal Services** require a client access license (CAL) to allow remote access to sessions on a terminal server.

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

authentication: The ability of one entity to determine the identity of another entity.

bidirectional: The ability to move, transfer, or transmit in two directions.

big-endian: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

branch office print mode: An operating mode in which a **print client** is able to perform branch office printing. Every shared printer on a **print server** can be configured to operate in branch office print mode.

branch office print remote logging: An operating mode in which a **print client** logs printing-related **Windows Events** on the **print server**. Branch office print remote logging occurs only when the **print client** is in **branch office print mode**.

checksum: A value that is the summation of a byte stream. By comparing the checksums computed from a data item at two different times, one can quickly assess whether the data items are identical.

class printer driver: Any **printer driver** declared by its manufacturer to be one from which a **derived printer driver** can derive. A **class printer driver** cannot itself be a **derived printer driver**. Typically, **class printer drivers** are generic and work with a variety of **devices**, while **derived printer drivers** work with a particular **device** and support features specific to that **device**.

color matching: The conversion of a color, sent from its original color space, to its visually closest color in the destination color space. See also **Image Color Management (ICM)**.

color profile: A file that contains information about how to convert colors in the color space and the color gamut of a specific device into a device-independent color space. A device-specific color profile is called a "device profile". For more information on using color and device profiles, see [MSDN-UDP].

container: An object in the directory that can serve as the parent for other objects. In the absence of schema constraints, all objects would be **containers**. The schema allows only objects of specific classes to be **containers**.

core printer driver: A **printer driver** that other printer drivers depend on. In Windows, this term includes the Unidrv and Pscript printer drivers. For more information, see [MSDN-UNIDRV] and [MSDN-PSCRIPT] respectively.

cyclic redundancy check (CRC): An algorithm used to produce a **checksum** (a small, fixed number of bits) against a block of data, such as a packet of network traffic or a block of a computer file. The CRC is used to detect errors after transmission or storage. A CRC is designed to catch random errors, as opposed to intentional errors. If errors might be introduced by a motivated and intelligent adversary, a cryptographic hash function should be used instead.

data type: A string that specifies the format of data that a printing application sends to a printer in a **print job**. Data types include **enhanced metafile spool format (EMFSPOOL)** and **RAW format**. For rules governing **data type** names, see section 2.2.4.2.

derived printer driver: A **printer driver** declared by its manufacturer to depend on a particular **class printer driver** by sharing modules with the **class printer driver**.

device: Any peripheral or part of a computer system that can send or receive data.

device driver: The software that the system uses to communicate with a device such as a display, printer, mouse, or communications adapter. An abstraction layer that restricts access of applications to various hardware devices on a given computer system. It is often referred to simply as a "driver".

directed discovery: A discovery method used by **WSD** devices. Directed discovery is used to discover devices on a subnet that is not the local subnet.

directory service (DS): A service that stores and organizes information about a computer network's users and network shares, and that allows network administrators to manage users' access to the shares. See also **Active Directory**.

discretionary access control list (DACL): An access control list (ACL) that is controlled by the owner of an object and that specifies the access particular users or groups can have to the object.

distinguished name (DN): A name that uniquely identifies an object by using the **relative distinguished name (RDN)** for the object, and the names of container objects and domains

that contain the object. The distinguished name (DN) identifies the object and its location in a tree.

dithering: A form of digital **halftoning**.

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set must act as a **domain controller (DC)** and host a member list that identifies all members of the domain, as well as optionally hosting the **Active Directory** service. The domain controller provides **authentication** of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

domain controller (DC): The service, running on a server, that implements **Active Directory**, or the server hosting this service. The service hosts the data store for objects and interoperates with other **DCs** to ensure that a local change to an object replicates correctly across all **DCs**. When **Active Directory** is operating as Active Directory Domain Services (AD DS), the **DC** contains full NC replicas of the configuration naming context (config NC), schema naming context (schema NC), and one of the domain NCs in its forest. If the AD DS **DC** is a global catalog server (GC server), it contains partial NC replicas of the remaining domain NCs in its forest. For more information, see [MS-AUTHSOD] section 1.1.1.5.2 and [MS-ADTS]. When **Active Directory** is operating as Active Directory Lightweight Directory Services (AD LDS), several AD LDS **DCs** can run on one server. When **Active Directory** is operating as AD DS, only one AD DS **DC** can run on one server. However, several AD LDS **DCs** can coexist with one AD DS **DC** on one server. The AD LDS **DC** contains full NC replicas of the config NC and the schema NC in its forest. The domain controller is the server side of Authentication Protocol Domain Support [MS-APDS].

domain name: A **domain name** or a NetBIOS name that identifies a **domain**.

Domain Name System (DNS): A hierarchical, distributed database that contains mappings of **domain names** to various types of data, such as IP addresses. DNS enables the location of computers and services by user-friendly names, and it also enables the discovery of other information stored in the database.

driver package: A collection of the files needed to successfully load a driver. This includes the device information (.inf) file, the catalog file, and all of the binaries that are copied by the .inf file. Multiple drivers packaged together for deployment purposes.

driver store: A secure location on the local hard disk where the entire driver package is copied.

endpoint: A network-specific address of a remote procedure call (RPC) server process for remote procedure calls. The actual name and type of the endpoint depends on the **RPC** protocol sequence that is being used. For example, for RPC over TCP (RPC Protocol Sequence ncacn_ip_tcp), an endpoint might be TCP port 1025. For RPC over Server Message Block (RPC Protocol Sequence ncacn_np), an endpoint might be the name of a named pipe. For more information, see [C706].

enhanced metafile format (EMF): A file format that supports the device-independent definitions of images.

enhanced metafile spool format (EMFSPOOL): A format that specifies a structure of **enhanced metafile format (EMF)** records used for defining application and device-independent printer **spool files**.

event channel: A collection of **Windows Events** that is provided by the system. Also referred to as an event log. The name of an event channel is composed of an event provider name combined with a channel type string. Valid channel types are “Admin”, “Analytic”, “Debug”, and “Operational”. For more information, see [MSDN-WINEV].

event ID: An identifier for the data represented by a **Windows Event**. Event IDs are unique with each event provider. For more information, see [MSDN-WINEV].

fax printer: A **print queue** that sends all print jobs to fax recipients as fax documents containing the printed data.

file: An entity of data in the file system that a user can access and manage. A **file** must have a unique name in its directory. It consists of one or more streams of bytes that hold a set of related data, plus a set of attributes (also called properties) that describe the **file** or the data within the **file**. The creation time of a **file** is an example of a file attribute.

file printer: A **print queue** that does not represent a physical device, but instead converts all print jobs to files containing the printed data.

fully qualified domain name (FQDN): An unambiguous **domain name** that gives an absolute location in the **Domain Name System's (DNS)** hierarchy tree, as defined in [RFC1035] section 3.1 and [RFC2181] section 11.

global catalog (GC): A unified partial view of multiple **naming contexts (NCs)** in a distributed partitioned directory. The **Active Directory** directory service **GC** is implemented by GC servers. The definition of **global catalog** is specified in [MS-ADTS] section 3.1.1.1.8.

globally unique identifier (GUID): A term used interchangeably with **universally unique identifier (UUID)** in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also **universally unique identifier (UUID)**.

Graphics Device Interface (GDI): A Windows API, supported on 16-bit and 32-bit versions of the operating system, that supports graphics operations and image manipulation on logical graphics objects.

GUIDString: A **GUID** in the form of an **ASCII** or **Unicode** string, consisting of one group of 8 hexadecimal digits, followed by three groups of 4 hexadecimal digits each, followed by one group of 12 hexadecimal digits. It is the standard representation of a GUID, as described in [RFC4122] section 3. For example, "6B29FC40-CA47-1067-B31D-00DD010662DA". Unlike a curly braced GUID string, a GUIDString is not enclosed in braces.

halftoning: The process of converting grayscale, or continuous-tone graphics or images, to a representation with a discrete number of gray (or tone) levels.

Image Color Management (ICM): Technology that ensures that a color image, graphic, or text object is rendered as closely as possible to its original intent on any device despite differences in imaging technologies and color capabilities between devices.

INF file: A file providing Windows Setup with the information required to set up a device, such as a list of valid logical configurations for the device and the names of driver files associated with the device.

information context: A special-purpose printer object that can only be used to obtain information about fonts that are supported by a printer. For more information, see [MSDN-FONTS].

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

Internet Protocol version 4 (IPv4): An Internet protocol that has 32-bit source and destination addresses. IPv4 is the predecessor of IPv6.

Internet Protocol version 6 (IPv6): A revised version of the Internet Protocol (IP) designed to address growth on the Internet. Improvements include a 128-bit IP address size, expanded routing capabilities, and support for **authentication** and privacy.

language monitor: An executable object that provides a communications path between a **print queue** and a printer's **port monitor**. **Language monitors** add control information to the data stream, such as commands defined by a **Page Description Language (PDL)**. They are optional, and are only associated with a particular type of printer if specified in the printer's **INF file**.

Lightweight Directory Access Protocol (LDAP): The primary access protocol for **Active Directory**. Lightweight Directory Access Protocol (LDAP) is an industry-standard protocol, established by the Internet Engineering Task Force (IETF), which allows users to query and update information in a **directory service (DS)**, as described in [MS-ADTS]. The Lightweight Directory Access Protocol can be either version 2 [RFC1777] or version 3 [RFC3377].

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

LOCALMON: The **port monitor** that manages local serial ("COM") and parallel ("LPT") **ports** on a Windows machine.

LPRMON: The **port monitor module** that allows Windows **print servers** to send **print jobs** to machines that support UNIX **print server** functions.

marshal: To encode one or more data structures into an octet stream using a specific **remote procedure call (RPC)** transfer syntax (for example, marshaling a 32-bit integer).

marshaling: The act of formatting COM parameters for transmission over a **remote procedure call (RPC)**. For more information, see [MS-DCOM].

Microsoft-Windows-PrintService: An event provider for printing services on Windows operating systems.

monitor module: An executable object that provides a communication path between the **print system** and the printers on a server.

multicast discovery: A discovery method used by **WSD** devices. Multicast discovery is used to discover devices on the local subnet.

multisz: A data type that defines an array of null-terminated, 16-bit **Unicode UTF-16LE**-encoded strings, with an additional null after the final string.

naming context (NC): An **NC** is a set of objects organized as a tree. It is referenced by a DSName. The **DN** of the DSName is the distinguishedName attribute of the tree root. The **GUID** of the DSName is the objectGUID attribute of the tree root. The **security identifier (SID)** of the DSName, if present, is the objectSid attribute of the tree root; for Active Directory Domain Services (AD DS), the **SID** is present if and only if the **NC** is a domain naming context (domain NC). **Active Directory** supports organizing several **NCs** into a tree structure.

NetBIOS: A particular network transport that is part of the LAN Manager protocol suite. **NetBIOS** uses a broadcast communication style that was applicable to early segmented local area networks. The LAN Manager protocols were the default in Windows NT operating system environments prior to Windows 2000 operating system. A protocol family including name resolution, datagram, and connection services. For more information, see [RFC1001] and [RFC1002].

Network Data Representation (NDR): A specification that defines a mapping from **Interface Definition Language (IDL)** data types onto octet streams. **NDR** also refers to the runtime

environment that implements the mapping facilities (for example, data provided to **NDR**). For more information, see [MS-RPCE] and [C706] section 14.

n-up printing: The act of arranging multiple logical pages on a physical sheet of paper.

object identifier (OID): In the context of an object server, a 64-bit number that uniquely identifies an object.

Open XML Paper Specification (OpenXPS): The **XML Paper Specification (XPS)** document format based on the European Carton Makers Association (ECMA) standard ECMA-388 [ECMA-388].

opnum: An operation number or numeric identifier that is used to identify a specific **remote procedure call (RPC)** method or a method in an interface. For more information, see [C706] section 12.5.2.12 or [MS-RPCE].

page description language (PDL): The language for describing the layout and contents of a printed page. Common examples are PostScript and **Printer Control Language (PCL)**.

plug-in: An executable module that can be loaded by the **print server** to perform specific functions.

port: A logical name that represents a connection to a **device**. A **port** can represent a network address (for example, a TCP/IP address) or a local connection (for example, a **USB port**).

port monitor: A **plug-in** that communicates with a **device** that is connected to a port. A **port monitor** may interact with the **device** locally, remotely over a network, or through some other communication channel. The data that passes through a **port monitor** is in a form that can be understood by the destination **device**, such as **page description language (PDL)**.

port monitor module: A **monitor module** for a **port monitor**.

PostScript: A **page description language** developed by Adobe Systems that is primarily used for printing documents on laser printers. It is the standard for desktop publishing.

principal: An authenticated entity that initiates a message or channel in a distributed system.

print client: The application or user that is trying to apply an operation on the print system either by printing a job or by managing the data structures or devices maintained by the print system.

print job: The rendered **page description language (PDL)** output data sent to a print device for a particular application or user request.

Print Pipeline: A service in the Windows implementation of the **XPS** printing subsystem that applies a series of **printer driver**-defined filters to the data in an **XPS** printer **spool file**.

print processor: A **plug-in** that runs on the **print server** and processes **print job** data before it is sent to a print **device**.

print provider: A **plug-in** that runs on the **print server** and routes **print system** requests. Print providers are a Windows implementation detail and are not required by this protocol.

print queue: The logical entity to which jobs may be submitted for a particular print device. Associated with a print queue is a print driver, a user's print configuration in the form of a DEVMODE structure, and a system print configuration stored in the system registry.

print server: A machine that hosts the print system and all its different components.

print system: A system component that is responsible for coordinating and controlling the operation of **print queues**, **printer drivers**, and **print jobs**.

print system remote protocol stress analysis: An optional diagnostic procedure that is used to analyze **print server** load, error counts, throughput, and other metrics.

Printer Control Language (PCL): A **page description language (PDL)** developed by Hewlett Packard for its laser and ink-jet printers.

printer driver: The interface component between the operating system and the printer device. It is responsible for processing the application data into a **page description language (PDL)** that can be interpreted by the printer device.

printer driver downgrade: An upgrade operation where an older **printer driver** is installed, replacing a newer **printer driver**.

printer driver isolation: An implementation technology by which a **print server** segregates **printer driver** execution into one or more processes separate from the **print server** to isolate the **print server** and other **printer drivers** from the side effects of faulty drivers.

printer driver manifest: A file that is installed with a **printer driver** and lists attributes of the **printer driver**. The formatting of **printer driver manifests** is specific to the print server implementation.

printer driver upgrade: An upgrade operation where a newer **printer driver** is installed, replacing an older **printer driver**.

printer form: A preprinted blank paper form, or a print job's virtual representation of this form, that enables a printer to position form elements in their physical location on the page.

printer key: A string that uniquely identifies a path under the main **registry** key where printer configuration data is kept. Rules for printer key names are specified in section 2.2.4.7.

printer UI application: An implementation-specific application optionally installed together with a printer driver. A printer UI application provides access to the user to discover available printer features, and monitor and modify printer configuration settings.

RAW format: A **data type** consisting of **PDL** data that can be sent to a **device** without further processing.

registry: A local system-defined database in which applications and system components store and retrieve configuration data. It is a hierarchical data store with lightly typed elements that are logically stored in tree format. Applications use the registry API to retrieve, modify, or delete registry data. The data stored in the registry varies according to the version of Windows.

relative distinguished name (RDN): The name of an object relative to its parent. This is the leftmost attribute-value pair in the **distinguished name (DN)** of an object. For example, in the **DN** "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com", the **RDN** is "cn=Peter Houston". For more information, see [RFC2251].

Remote Administration Protocol (RAP): A synchronous request/response protocol, used prior to the development of the remote procedure call (RPC) protocol, for marshaling and unmarshaling procedure call input and output arguments into messages and for reliably transporting messages to and from clients and servers.

remote procedure call (RPC): A context-dependent term commonly overloaded with three meanings. Note that much of the industry literature concerning RPC technologies uses this term interchangeably for any of the three meanings. Following are the three definitions: (*) The runtime environment providing remote procedure call facilities. The preferred usage for this meaning is "RPC runtime". (*) The pattern of request and response message exchange between two parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (*) A single message from an exchange as defined in the previous definition. The preferred usage for this term is "RPC message". For more information about RPC, see [C706].

RPC context handle: A representation of state maintained between a remote procedure call (RPC) client and server. The state is maintained on the server on behalf of the client. An RPC context handle is created by the server and given to the client. The client passes the RPC context handle back to the server in method calls to assist in identifying the state. For more information, see [C706].

RPC endpoint: A network-specific address of a server process for remote procedure calls (RPCs). The actual name of the RPC endpoint depends on the RPC protocol sequence being used. For example, for the NCACN_IP_TCP RPC protocol sequence an RPC endpoint might be TCP port 1025. For more information, see [C706].

RPC transfer syntax: A method for encoding messages defined in an Interface Definition Language (IDL) file. Remote procedure call (RPC) can support different encoding methods or transfer syntaxes. For more information, see [C706].

RPC transport: The underlying network services used by the remote procedure call (RPC) runtime for communications between network nodes. For more information, see [C706] section 2.

schema: The set of attributes and object classes that govern the creation and update of objects.

SD: See **security descriptor**.

security descriptor: A data structure containing the security information associated with a securable object. A **security descriptor** identifies an object's owner by its **security identifier (SID)**. If access control is configured for the object, its **security descriptor** contains a **discretionary access control list (DACL)** with **SIDs** for the security principals who are allowed or denied access. Applications use this structure to set and query an object's security status. The **security descriptor** is used to guard access to an object as well as to control which type of auditing takes place when the object is accessed. The **security descriptor** format is specified in [MS-DTYP] section 2.4.6; a string representation of **security descriptors**, called SDDL, is specified in [MS-DTYP] section 2.5.1.

security identifier (SID): An identifier for security principals in Windows that is used to identify an account or a group. Conceptually, the **SID** is composed of an account authority portion (typically a **domain**) and a smaller integer representing an identity relative to the account authority, termed the relative identifier (RID). The **SID** format is specified in [MS-DTYP] section 2.4.2; a string representation of **SIDs** is specified in [MS-DTYP] section 2.4.2 and [MS-AZOD] section 1.1.1.2.

security provider: A pluggable security module that is specified by the protocol layer above the **remote procedure call (RPC)** layer, and will cause the **RPC** layer to use this module to secure messages in a communication session with the server. The security provider is sometimes referred to as an **authentication** service. For more information, see [C706] and [MS-RPCE].

server: A computer on which the **remote procedure call (RPC)** server is executing.

Server Message Block (SMB): A protocol that is used to request file and print services from server systems over a network. The SMB protocol extends the CIFS protocol with additional security, file, and disk management support. For more information, see [CIFS] and [MS-SMB].

server restart: Any event that causes the **print server** to stop and start again, including a service or process shutdown and restart, an operating system shutdown and restart, or an unscheduled event, such as a power failure.

service printer: A **print queue** that sends rendered print jobs to a destination external to the print server using an implementation-specific mechanism that is opaque to the print server.

shared printer: A **print queue** that is available to **print clients** as a share.

Simple and Protected GSS-API Negotiation Mechanism (SPNEGO): An **authentication** mechanism that allows Generic Security Services (GSS) peers to determine whether their credentials support a common set of GSS-API security mechanisms, to negotiate different options within a given security mechanism or different options from several security mechanisms, to select a service, and to establish a security context among themselves using that service. **SPNEGO** is specified in [RFC4178].

spool file: A representation of application content data than can be processed by a **printer driver**. Common examples are enhanced metafile format and **XML Paper Specification (XPS)** [MSDN-XMLP]. For more information, see [MSDN-META].

string resource: A string that is stored in a resource file and that can be retrieved with a key. A **string resource** is localizable into multiple languages. It is up to an AsyncUI client implementation to determine which language string to retrieve for a given key.

system access control list (SACL): An access control list (ACL) that controls the generation of audit messages for attempts to access a securable object. The ability to get or set an object's **SACL** is controlled by a privilege typically held only by system administrators.

TCPMON: The **port monitor module** that manages standard **TCP**/IP **ports** on a Windows machine. **TCPMON** supports configuring a **TCP**/IP **port** and obtaining information about the **port** configuration.

terminal services (TS): A service on a server computer that allows delivery of applications, or the desktop itself, to various computing devices. When a user runs an application on a terminal server, the application execution takes place on the server computer and only keyboard, mouse, and display information is transmitted over the network. Each user sees only his or her individual session, which is managed transparently by the server operating system and is independent of any other client session.

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

unicast: A delivery method used by media servers for providing content to connected clients in which each client receives a discrete stream that no other client has access to.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

Unicode string: A **Unicode** 8-bit string is an ordered sequence of 8-bit units, a **Unicode** 16-bit string is an ordered sequence of 16-bit code units, and a **Unicode** 32-bit string is an ordered sequence of 32-bit code units. In some cases, it may be acceptable not to terminate with a terminating null character. Unless otherwise specified, all **Unicode strings** follow the **UTF-16LE** encoding scheme with no Byte Order Mark (BOM).

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

Uniform Resource Locator (URL): A string of characters in a standardized format that identifies a document or resource on the World Wide Web. The format is as specified in [RFC1738].

Universal Naming Convention (UNC): A string format that specifies the location of a resource. For more information, see [MS-DTYP] section 2.2.57.

Universal Plug and Play (UPnP): A set of computer network protocols, published by the UPnP Forum [UPnP], that allow devices to connect seamlessly and that simplify the implementation of networks in home (data sharing, communications, and entertainment) and corporate environments. UPnP achieves this by defining and publishing UPnP device control protocols built upon open, Internet-based communication standards.

universal serial bus (USB): An external bus that supports Plug and Play installation. It allows devices to be connected and disconnected without shutting down or restarting the computer.

universally unique identifier (UUID): A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and **RPC** objects. UUIDs are highly likely to be unique. UUIDs are also known as **globally unique identifiers (GUIDs)** and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.

UNIX: A multiuser, multitasking operating system developed at Bell Laboratories in the 1970s. In this document, the term "**UNIX**" is used to refer to any derivatives of this operating system.

USBMON: The port monitor that manages local USB ports on a Windows machine.

UTF-16LE: The Unicode Transformation Format - 16-bit, Little Endian encoding scheme. It is used to encode **Unicode** characters as a sequence of 16-bit codes, each encoded as two 8-bit bytes with the least-significant byte first.

virtual printer: A **print queue** that does not produce physical printed output, and is not a **fax printer**, **file printer**, or **service printer**.

Web Services for Devices (WSD): A technology and associated API that expands on Microsoft's Web Services Dynamic Discovery Protocol [WS-Discovery] to allow a client to discover and access remote devices and associated services across a network. WSD supports device discovery, description, control, and eventing.

Web Services on Devices (WSD): A function-discovery protocol used to discover and communicate certain data structures in a HomeGroup network environment. Implementation details are specified in [DPWS].

well-known endpoint: A preassigned, network-specific, stable address for a particular client/server instance. For more information, see [C706].

white point: The color value used as the reference to which the user adapts.

Windows Event: A technology and associated API that is typically used for troubleshooting application and driver software on a computer running Windows. A Windows Event contains an identifier and associated data. Events are published by an event provider to an **event channel** for consumption, and the identifiers are unique to the event provider. For more information, see [MSDN-WINEV].

writability: The abstract feature capability representing the ability of a **domain controller (DC)** to accept modifications and issue originating updates, with respect to a given **naming context (NC)** replica.

WSDMON: The **port monitor** that supports printing to network printers that comply with **WSD** technology.

WS-Print: The **schema** for **WSD** printing. For more information, see [MSDN-WSPRINT].

XML Paper Specification (XPS): A Microsoft XML-based document format introduced in Windows Vista operating system. **XML Paper Specification (XPS)** specifies the set of conventions for the use of XML and other widely available technologies to describe the content and appearance of paginated documents. For more information, see [MSFT-XMLPAPER].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, https://www2.opengroup.org/ogsys/catalog/c706

[IEEE1284] Institute of Electrical and Electronics Engineers, "IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers - Description", IEEE Std 1284, 1994, http://standards.ieee.org/reading/ieee/std_public/description/busarch/1284-1994_desc.html

[MS-ADA3] Microsoft Corporation, "Active Directory Schema Attributes N-Z".

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes".

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification".

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol".

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-ERREF] Microsoft Corporation, "Windows Error Codes".

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".

[MS-PAR] Microsoft Corporation, "Print System Asynchronous Remote Protocol".

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions".

[MS-RRP] Microsoft Corporation, "Windows Remote Registry Protocol".

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3".

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol".

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987, http://www.ietf.org/rfc/rfc1001.txt

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and Davin, J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990, http://www.ietf.org/rfc/rfc1157.txt

[RFC1179] McLaughlin III, L., "Line Printer Daemon Protocol", RFC 1179, August 1990, http://www.ietf.org/rfc/rfc1179.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[RFC2254] Howes, T., "The String Representation of LDAP Search Filters", RFC 2254, December 1997, http://www.ietf.org/rfc/rfc2254.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC2781] Hoffman, P., and Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000, http://www.rfc-editor.org/rfc/rfc2781.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, http://www.ietf.org/rfc/rfc3986.txt

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

1.2.2 Informative References

[DEVMODE] Microsoft Corporation, "DEVMODE structure", http://msdn.microsoft.com/en-us/library/dd183565(VS.85).aspx

[ECMA-388] ECMA International, "Open XML Paper Specification", ECMA-388, June 2009, http://www.ecma-international.org/publications/standards/Ecma-388.htm

[IEEE802.3-2008] Institute of Electrical and Electronics Engineers, "Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Description", IEEE Std 802.3, 2008, http://standards.ieee.org/getieee802/802.3.html

[MS-ADLS] Microsoft Corporation, "Active Directory Lightweight Directory Services Schema".

[MS-ADOD] Microsoft Corporation, "Active Directory Protocols Overview".

[MS-AZOD] Microsoft Corporation, "Authorization Protocols Overview".

[MS-EMFSPOOL] Microsoft Corporation, "Enhanced Metafile Spool Format".

[MS-EMF] Microsoft Corporation, "Enhanced Metafile Format".

[MS-PAN] Microsoft Corporation, "Print System Asynchronous Notification Protocol".

[MS-RAP] Microsoft Corporation, "Remote Administration Protocol".

[MSDN-ADOVRVW] Microsoft Corporation, "Active Directory Schema Terminology", http://msdn.microsoft.com/en-us/library/ms675087.aspx

[MSDN-BIDI] Microsoft Corporation, "Bidirectional Communication", http://msdn.microsoft.com/en-us/library/aa907381.aspx

[MSDN-CAB] Microsoft Corporation, "Microsoft Cabinet Format", March 1997, http://msdn.microsoft.com/en-us/library/bb417343.aspx

[MSDN-FIELD] Microsoft Corporation, "Field Attributes", http://msdn.microsoft.com/en-us/library/aa373864(VS.85).aspx

[MSDN-GPDFiles] Microsoft Corporation, "Introduction to GPD Files", http://msdn.microsoft.com/en-us/library/ff551750(VS.85).aspx

[MSDN-MPD] Microsoft Corporation, "Microsoft Print Drivers", http://msdn.microsoft.com/en-us/library/ff556565.aspx

[MSDN-MUI] Microsoft Corporation, "Language Identifier Constants and Strings", https://msdn.microsoft.com/en-us/library/windows/desktop/dd318693(v=vs.85).aspx

[MSDN-SPOOL] Microsoft Corporation, "Print Spooler Components", http://msdn.microsoft.com/en-us/library/ff561109.aspx

[MSDN-UINF] Microsoft Corporation, "Using INF Files", http://msdn.microsoft.com/en-us/library/Aa741213.aspx

[MSDN-XMLP] Microsoft Corporation, "A First Look at APIs For Creating XML Paper Specification Documents", http://msdn.microsoft.com/en-us/magazine/cc163664.aspx

[MSFT-XMLPAPER] Microsoft Corporation, "XML Paper Specification", http://www.microsoft.com/whdc/xps/default.mspx

[PS-PPD4.3] Adobe Systems Incorporated, "PostScript Printer Description File Format Specification", version 4.3, February 1996, http://partners.adobe.com/public/developer/en/ps/5003.PPD_Spec_v4.3.pdf

[RFC819] Su, Z.S. and Postel, J., "The Domain Naming Convention for Internet User Applications", RFC 819, August 1982, http://www.ietf.org/rfc/rfc0819.txt

[USBPRINT] USB Implementers Forum, "Universal Serial Bus Device Class Definition for Printing Devices", version 1.1, January 2000, http://www.usb.org/developers/docs/devclass_docs/usbprint11a021811.pdf

1.3 Overview

The Print System Remote Protocol provides the following functions:

- Management of the print system of a **print server** from a client.
- Communication of print job data from a client to a print server.
- Notifications to the client of changes in the print server's print system.

Server processing instructions are specified by the parameters that are used in the protocol methods. These parameters include:

- **Printer driver** configuration information.
- The **spool file** format for the print data that is sent by the client.
- The **access level** of the connection.
- The target **print queue** name for name-based methods.
- A handle to the target print queue for handle-based methods.

Status information is communicated back to the client in the return codes from calls that are made to the print server.

The following sections give an overview of these functions.

1.3.1 Management of the Print System

A client can use this protocol to perform remote management operations on a print server. With server access credentials, client applications can manipulate the print server state and print server components, such as printer driver configuration and print queue configuration, or add printer drivers and printers; they can monitor the print queue status; and they can perform general print server administration.

These operations are supported in the protocol by a set of container structures that are used by different print system components, specifically: DRIVER_CONTAINER, FORM_CONTAINER, JOB_CONTAINER, PORT_CONTAINER, SECURITY_CONTAINER, and PRINTER_CONTAINER. These print system components are supported as specified in section 2.2.1.

To produce printed output that is the same, regardless of the configuration, the printer driver that is installed on the client computer must be identical to or compatible with the printer driver that is installed on the print server. This protocol provides the methods that the client can use after it connects to a printer on a print server to obtain the information about the printer driver that is associated with the printer. If necessary, the client computer can use this information to download the printer driver from the print server. For more information about printer drivers, see [MSDN-MPD].

The client can also use this protocol to obtain detailed information about the settings of the printer and the printer driver that are installed on the server. The client application can use this information to perform layout and to make device-specific choices about paper formats, resolution, and color handling. After the client connects to a printer, this protocol provides the methods that the client can use to query these settings.

The following figure illustrates this interaction, using the scenario of adding a new printer as an example.

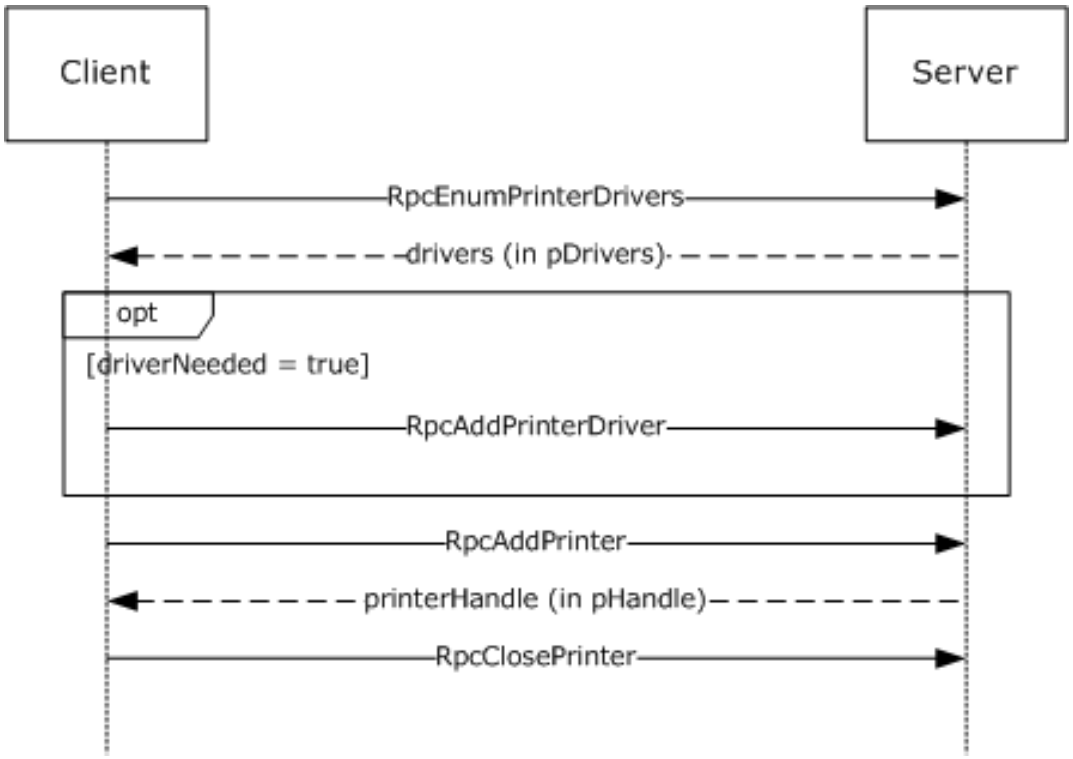


Figure 1: Adding a new printer

1.3.2 Communication of Print Job Data

Communication of print job data enables a client to print to devices that are hosted by print servers.

In one configuration, a client uses a printer driver that is installed on the client computer to convert a graphical representation of application content and layout into **device**-specific **page description language (PDL)** data. It then sends the data, also called **RAW** data, to the print server using methods this protocol provides. The print server can temporarily store the RAW data from the client in a spool file, or it can print it immediately. As the print server sends the data to the target printer, the **print processor** on the print server that is associated with the target printer can post-process the RAW data in an implementation-specific way.

In another configuration, a client sends data to the print server in an intermediate format that contains graphics primitives and layout information in addition to processing instructions for the print server. The print server can temporarily store this intermediate data in a spool file, or it can print it immediately. As the data is sent to the printer, the print processor on the print server that is associated with the printer converts the data from the intermediate spool file to device-specific PDL data, typically by using the printer driver that is installed on the print server.

The following diagram illustrates this interaction.

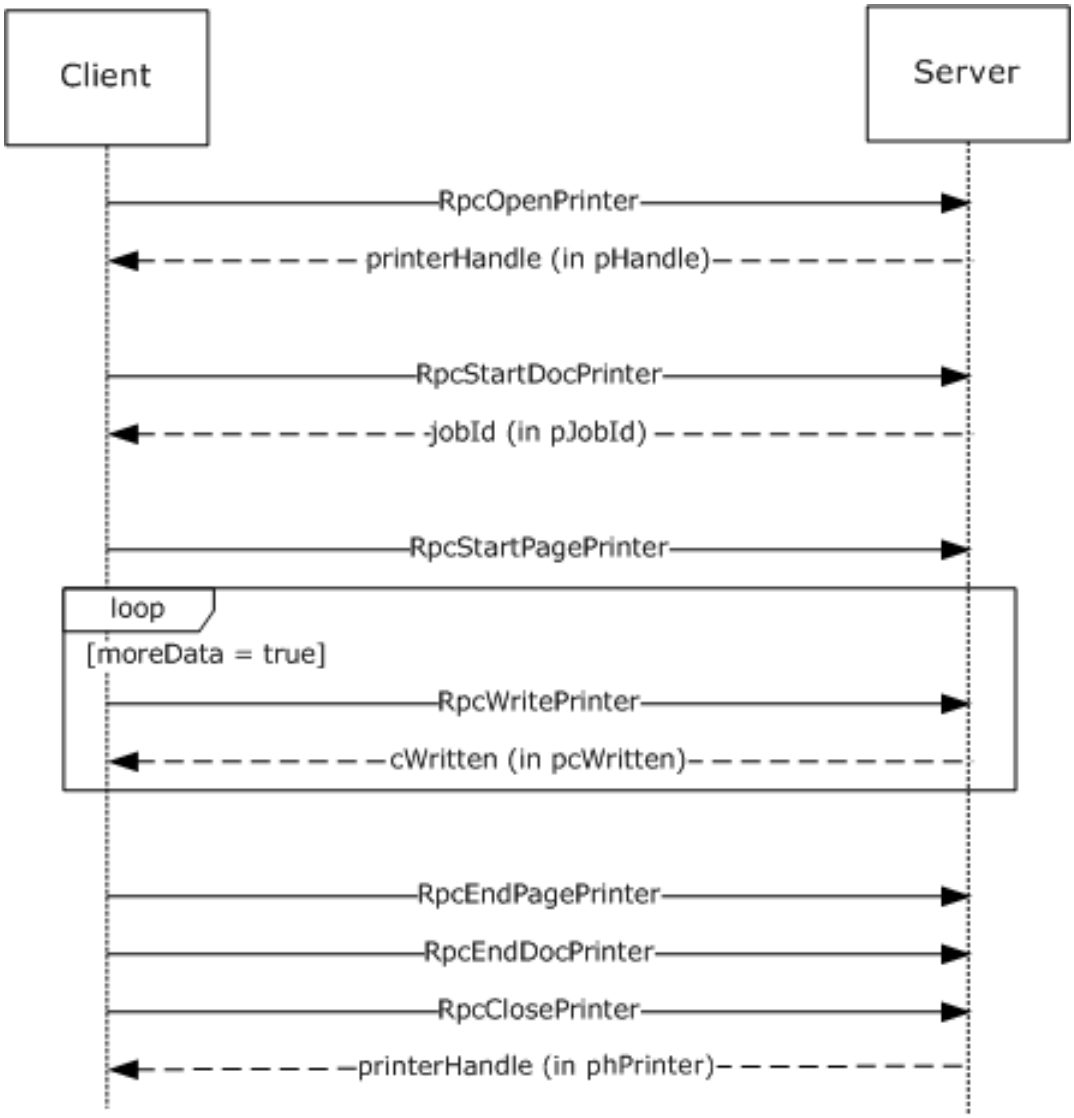


Figure 2: Communication of print job data

1.3.3 Notification of Print System Changes

This protocol also provides the methods that a **print client** can use to register for incremental change notifications. These notifications enable the client application to maintain an accurate local view of the printer and printer driver settings by enabling the client application to synchronize the local view with the actual settings of those components on the print server, without having to repeatedly query the server for its complete configuration information.

For status updates, a print client registers for notifications of state changes when it connects to a print server. The server creates a new remote procedure call (RPC) connection in the reverse direction, back to the client, which is subsequently used to send notifications to the client. When the status of a server resource changes—such as a print queue goes online, goes offline, or enters an error state—the server sends a notification to the registered client.

Notifications include status changes of print server resources; for example, when a print queue goes online, goes offline, or enters an error state.

The following diagram illustrates this interaction. For more information, see section 3.2.4.2.4.

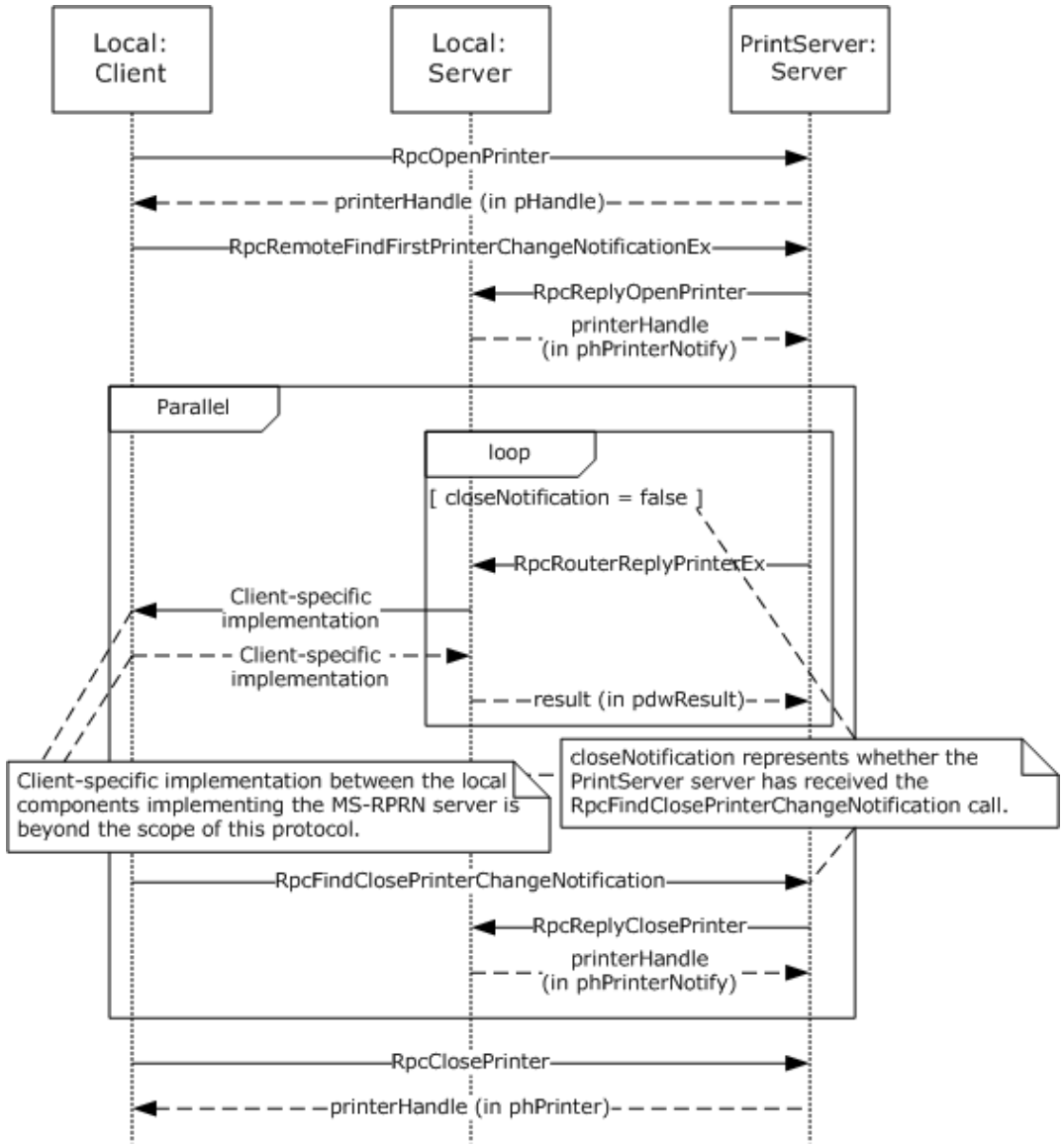


Figure 3: Notification of print system changes

The box labeled 'Local Server' in the above diagram represents an entity running on the client system. This entity is either implemented in the end-user application or in a separate process. Since the entity implements a Print System Remote Protocol **endpoint**, which can only be registered once per system, there can be at most one of these entities running on the client system at any time. If an end-user application directly implements this entity, this end-user application would not be able to run on a system that also runs a print system implementation, and only one such application could.

In addition to composing and returning the notifications, the print server maintains a change identifier that it changes whenever the server-side printing configuration changes; for example, changes to user-configurable settings, print queue items, print job status, or to the printer driver would cause this identifier to change. The print client can query this change identifier by using the RpcGetPrinterData (section 3.1.4.2.7) method that is defined in this protocol and calling it with the *pValueName* parameter pointing to the string "ChangeID".

When a disconnected print client reconnects to the print server, it can query the change identifier again, and if the change identifier is different from the one returned when it queried before it was disconnected, the client should retrieve the complete configuration information and update its view of the server configuration. The client retrieves the complete configuration using the functions for Management of the Print System (section 1.3.1).

1.4 Relationship to Other Protocols

The Print System Remote Protocol is dependent on the RPC protocol specified in [MS-RPCE].

The Print System Remote Protocol does not specify methods for file transfer between client and server; therefore, the **Server Message Block (SMB)** Version 2.0 Protocol, specified in [MS-SMB2], is the preferred protocol for all file transfer operations, including printer driver downloads.

These protocol relationships are shown in the following figure:

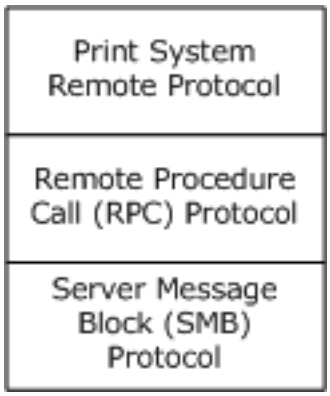


Figure 4: Protocol Relationships

The Print System Remote Protocol is related to other protocols as follows:

- The Print System Asynchronous Notification Protocol, specified in [MS-PAN], is dependent on the Print System Remote Protocol.
- The Print System Asynchronous Remote Protocol, specified in [MS-PAR], uses many data structures and parameter definitions that are also specified in sections 2.2.1 and 2.2.2 of this specification.

  Note: The implementation of Print System Remote is required for all print servers, but a print server can additionally implement Print System Asynchronous Remote.
- Limited enumeration of printer configuration information can be done using the **Remote Administration Protocol (RAP)**, specified in [MS-RAP]; however, the Print System Asynchronous Remote Protocol provides richer semantics than RAP. Because RAP is not supported over SMB Version 2.0, Print System Remote is the preferred access protocol for printer configuration information.
- The print client and print server implementations of this protocol use the Active Directory Technical Specification Protocol specified in [MS-ADTS], to locate domain controllers and use **LDAP**, as specified in [RFC2251], to access data in the **Active Directory**, when available. This protocol does not require availability of Active Directory. For more details, see Directory Service Interaction (section 2.3).<1>

1.5 Prerequisites/Preconditions

This protocol is an RPC interface and therefore has the prerequisites specified in [MS-RPCE] section 1.5 as being common to RPC interfaces.

It is assumed that a client of this protocol has obtained the name of a print server that supports this protocol before it is invoked. There are various ways a client can accomplish that; for information see [MS-ADLS], [MS-ADSC], [MS-RAP], and [MS-SMB2].

1.6 Applicability Statement

The Print System Remote Protocol is applicable only for printing operations between a system functioning as a client and a system functioning as a print server. This protocol scales from home use, in which a single printer is connected to a single computer; to office use, in which print-devices are shared between computers; to enterprise use, in which multiple print servers are employed in a cluster configuration, and the client configuration is managed by a directory access protocol, such as Active Directory as specified in [MS-ADTS].

1.7 Versioning and Capability Negotiation

This specification covers versioning issues in the following areas:

- **Supported Transports**: The Print System Remote Protocol requires use of RPC over named pipes only.
- **Protocol Versions**: The protocol version specified in the **Interface Definition Language (IDL)** file is 1.0.

  Versioning of data structures defined by the protocol is controlled using a Level value in all Containers (section 2.2.1.2); the usage pattern of Level values is defined in section 2.2. Levels can be sequential, and data structures identified by a later version level, if extending an earlier level, are a superset of the data structure identified by the earlier level. The Level value is also a parameter to some RPC methods.
- **Security and Authentication Methods**: Versioning of security is handled by the underlying **RPC transport**; see [MS-RPCE] section 3.3.3.3 for more information.
- **Localization**: This protocol specifies languages and localizable **string resources** for **printer forms**. See LANGID (section 2.2.1.1.3) and RPC_FORM_INFO_2 (section 2.2.1.6.2) for details.
- **Return Values**: The methods that make up this RPC interface require a return value of zero to indicate successful completion and nonzero values to indicate failure, except where specified otherwise. A server-side implementation of this protocol would preferably use any nonzero Win32 error value to signify an error condition, as described in section 1.8. Unless otherwise specified, the client side of the Print System Remote Protocol is not allowed to interpret returned error codes. Unless otherwise specified, the client side of the Print System Remote Protocol is required to simply return error codes to the invoking application without taking any protocol action.
- **Capability Negotiation**: Functional negotiation is supported through the use of Container levels; see section 2.2.1.2. On connection to a server, the client requests a level. If the information level is a level supported by the server, the server is required to process the request. Otherwise, the server has to return an error to the client, and the client would preferably repeat the request with a lower level.

  Furthermore, to avoid unnecessary network calls, the client determines the server's capabilities by comparing the value returned by the server in the **dwBuildNumber** member of OSVERSIONINFO (section 2.2.3.10.1) with well-known version-specific **dwBuildNumber** values.<2>

1.8 Vendor-Extensible Members

The methods defined in the Print System Remote Protocol specify either the DWORD or HRESULT **data type** for return values. DWORD return values are Win32 error codes taken from the Windows error number space specified in [MS-ERREF] section 2.2. Implementers reuse those values with their indicated meanings. Choosing any other value runs the risk of collisions.

HRESULT method return values are used as defined in [MS-ERREF] section 2.1. Vendors can choose their own HRESULT values, but the C bit (0x20000000) is set, indicating that it is a customer code.

Print server implementations MUST generate **GUID**, as defined in [MS-DTYP] sections 2.3.4, 2.3.4.2, and 2.3.4.3, strings for the purpose of identifying specific printers. The 128-bit value encoded by the GUID string SHOULD conform to the specification in [RFC4122] section 4.

1.9 Standards Assignments

The Print System Remote Protocol requires use of the following private assignments:

Parameter	Value	Reference
RPC **UUID**	12345678-1234-ABCD-EF00-0123456789AB	Message Processing Events and Sequencing Rules (section 3.1.4)
RPC **well-known endpoint**	**\pipe\spoolss**	Transport (section 2.1)

2 Messages

2.1 Transport

The Print System Remote Protocol uses RPC over named pipes (see [MS-RPCE] section 2.1.1.2) for RPC sequences. The well-known endpoint **\pipe\spoolss** is used for RPC calls made from the print client to the print server. The client MUST use no **authentication**, and the server MUST accept connections without authentication.

An endpoint with the same name MUST also be used for RPC calls made from the server to send printer change notifications back to the client; those calls are RpcReplyOpenPrinter (section 3.2.4.1.1), RpcRouterReplyPrinter (section 3.2.4.1.2), RpcReplyClosePrinter (section 3.2.4.1.3), and RpcRouterReplyPrinterEx (section 3.2.4.1.4). The client MUST accept connections without authentication from the server for these methods. If the client provides authentication information (as specified in [MS-RPCE] 2.2.1.1.8), the server MAY impersonate the client (as specified in [MS-RPCE] 2.2.1.1.9), while processing a method.<3>

2.2 Common Data Types

The Print System Remote Protocol MUST indicate to the RPC runtime that it is to support both the **Network Data Representation (NDR)** and **NDR64 RPC transfer syntaxes** and provide a negotiation mechanism for determining which transfer syntax will be used, as specified in [MS-RPCE] section 3.

This protocol MUST enable the **ms_union** extension as specified in [MS-RPCE] section 2.2.4.

The Print System Remote Protocol employs a combination of the following data representations:

- IDL data structures used with RPC methods, including structures used as containers for custom-**marshaled**, custom C data, are specified in section 2.2.1.
- Custom C data structures and their wire formats used within custom marshaled data streams are specified in section 2.2.2.

Unless noted otherwise, the following statements apply to this specification:

- All strings defined in this protocol MUST consist of characters encoded in **Unicode UTF-16LE** and MUST be null-terminated. Each UTF-16 codepoint in a string, including terminating null characters, MUST occupy 16 bits as specified in [RFC2781], section 2.1.
- A list of strings is referred to as a **multisz** in this protocol. In a multisz, the characters making up the string N+1 MUST directly follow the terminating null character of string N. The last string in a multisz MUST be terminated by two null characters.
- All parameters or members specifying the number of characters in a string or multisz specify the number of characters including terminating null characters.
- All constraints specifying the maximum number of characters in a string or multisz specify the number of characters including terminating null characters.
- All parameters or members specifying the number of bytes in buffers containing a string or multisz specify the number of bytes including terminating null characters.
- Custom-marshaled data that consists of more than a single byte is specified in **little-endian** byte order.
- The term NULL means a NULL pointer, and zero means the number 0.

- All parameters or members specifying the size of a buffer pointed to by another parameter or member MUST be zero if the pointer parameter or member is NULL.
- The term "empty string" means a string containing only the terminating null character.
- The term "optional pointer" means that providing a pointer value in the parameter or member is optional. If the pointer value is not provided, the value of the parameter or member MUST be NULL.
- This protocol specification uses curly-braced string GUIDs as specified in [MS-DTYP] section 2.3.4.3.

This protocol introduces a variety of data types that bundle information about printers, printer drivers, print jobs, and **ports**. These data types are collectively referred to as INFO data types, and include DRIVER_INFO_1 (section 2.2.1.5.1), PRINTER_INFO_1 (section 2.2.1.10.2), JOB_INFO_1 (section 2.2.1.7.1), and PORT_INFO_1 (section 2.2.1.9.1). As data types were refined in the evolution of this protocol, new INFO data type versions have been introduced to represent extended or different bundles of information. The term "level" is used to differentiate between the different type versions, and the number of the level is reflected in the name of the data type, for example, JOB_INFO_1, JOB_INFO_2, and JOB_INFO_3.

To simplify method parameter lists and to increase robustness of RPC **marshaling**, the protocol introduces CONTAINER data types, which consolidate the input parameters used by RPC methods. Some CONTAINER data types hold a Level value along with a union of pointer values pointing to different INFO data type versions; the specific Level values available for each CONTAINER data type are documented in section 2.2.1.2. For example, a JOB_CONTAINER (section 2.2.1.2.5) contains a level value and a union of pointers to the different JOB_INFO data type versions, which are selected by the Level value. Other CONTAINER data types hold a pointer value that points to a structure, along with a numerical value representing the size of the structure. For example, a DEVMODE_CONTAINER (section 2.2.1.2.1) contains a size value and a pointer to a custom-marshaled structure. Finally, several CONTAINER data types hold a version value, a value representing a set of flags, an array of structures, and a value representing the number of elements in the array. The RPC_BIDI_REQUEST_CONTAINER (section 2.2.1.2.10) is an example of a CONTAINER data type in this category.

Most of the INFO data types have an IDL form and a custom-marshaled form. IDL forms can be used in conjunction with CONTAINER data types, as input parameters to methods that set values, such as RpcSetPrinter (section 3.1.4.2.5), while custom-marshaled forms can be used as output parameters to methods that get values, such as RpcGetPrinter (section 3.1.4.2.6). The layout and order of members of IDL forms are in most cases the same as those of corresponding custom-marshaled forms, with the distinction that IDL forms use the type "[string] wchar_t *" to point to strings, while custom-marshaled forms use an offset relative to the start of the structure.

As an exception to the preceding rule, the layout of IDL-marshaled structures that contain pointers to multisz data **differs** from the layout of custom-marshaled forms, in that IDL-marshaled structures need to define a **length** member for each IDL-marshaled member of type **pointer to** multisz; the names of such IDL-marshaled members start with RPC_.

To increase clarity, an underscore has been prepended to the names of all custom-marshaled structures, for which an IDL-marshaled form exists. For example, _DRIVER_INFO_1 is the name of the custom-marshaled structure that corresponds to DRIVER_INFO_1, the IDL-marshaled form.

When IDL-marshaled structures that contain pointer types to variable-length data without field IDL attributes (see [MSDN-FIELD]), such as [string] or [size_is(...)], are used as input arguments to methods, either directly or in CONTAINER structures, the pointers and variables to which they point cannot be marshaled by RPC, because RPC does not know the length of the data that is pointed to. Examples are the **pSecurityDescriptor** and **pDevMode** members of the PRINTER_INFO_2 structure.

To address this problem, methods that specify such input arguments accept separate CONTAINER structures that pass in custom-marshaled or self-relative forms of the pointers and the variables that

they reference. Examples of such methods are RpcSetPrinter and RpcAddPrinterEx (section 3.1.4.2.15). Individual method sections specify how affected pointer members and CONTAINER structures MUST be treated.

2.2.1 IDL Data Types

In addition to the RPC base types and definitions specified in [C706] and [MS-DTYP], the Print System Remote Protocol defines data types in the following sections:

- Common IDL Data Types (section 2.2.1.1)
- Containers (section 2.2.1.2)
- Members in INFO Structures (section 2.2.1.3)
- DOC_INFO_1 (section 2.2.1.4)
- DRIVER_INFO (section 2.2.1.5)
- FORM_INFO (section 2.2.1.6)
- JOB_INFO (section 2.2.1.7)
- MONITOR_INFO (section 2.2.1.8)
- PORT_INFO (section 2.2.1.9)
- PRINTER_INFO (section 2.2.1.10)
- SPLCLIENT_INFO (section 2.2.1.11)
- Bidirectional Communication Data (section 2.2.1.12)
- Printer Notification Data (section 2.2.1.13)
- Job Named Properties (section 2.2.1.14)
- Branch Office Print Remote Logging Structures (section 2.2.1.15)

2.2.1.1 Common IDL Data Types

2.2.1.1.1 DEVMODE

The DEVMODE structure is a truncated form of the variable-length, custom-marshaled _DEVMODE structure (section 2.2.2.1), which is version-specific and implementation-specific and cannot be expressed using IDL attributes.

All members of this structure are specified in section 2.2.2.1.

```
typedef struct  devicemode {
  wchar t dmDeviceName[32];
  unsigned short dmSpecVersion;
  unsigned short dmDriverVersion;
  unsigned short dmSize;
  unsigned short dmDriverExtra;
  DWORD dmFields;
  short dmOrientation;
  short dmPaperSize;
  short dmPaperLength;
  short dmPaperWidth;
  short dmScale;
```

```
  short dmCopies;
  short dmDefaultSource;
  short dmPrintQuality;
  short dmColor;
  short dmDuplex;
  short dmYResolution;
  short dmTTOption;
  short dmCollate;
  wchar_t dmFormName[32];
  unsigned short reserved0;
  DWORD reserved1;
  DWORD reserved2;
  DWORD reserved3;
  DWORD dmNup;
  DWORD reserved4;
  DWORD dmICMMethod;
  DWORD dmICMIntent;
  DWORD dmMediaType;
  DWORD dmDitherType;
  DWORD reserved5;
  DWORD reserved6;
  DWORD reserved7;
  DWORD reserved8;
} DEVMODE;
```

2.2.1.1.2 GDI_HANDLE

The GDI_HANDLE serves as an **RPC context handle** for methods that specify a printer **information context** handle parameter. RPC context handles are specified in [C706] sections 2 and 6.

This type is declared as follows:

```
typedef [context_handle] void* GDI_HANDLE;
```

The GDI_HANDLE context handle is returned by RpcCreatePrinterIC.

2.2.1.1.3 LANGID

The LANGID data type identifies the human language used for the user interface for printing. Details are specified in [MS-LCID].

This type is declared as follows:

```
typedef unsigned short LANGID;
```

2.2.1.1.4 PRINTER_HANDLE

The PRINTER_HANDLE serves as an RPC context handle for methods that specify a printer object handle parameter. RPC context handles are specified in [C706] sections 2 and 6.

This type is declared as follows:

```
typedef [context_handle] void* PRINTER_HANDLE;
```

The PRINTER_HANDLE context handle is returned by RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, and RpcOpenPrinterEx.

2.2.1.1.5 RECTL

The RECTL structure defines a rectangle on a form, with two (x,y) coordinates in 1/1000 millimeter units.

```
typedef struct {
  long left;
  long top;
  long right;
  long bottom;
} RECTL;
```

left: The value of this member MUST specify the x-coordinate of the upper-left corner of the rectangle relative to the left edge of the form. This value MUST be an integer greater than or equal to 0 and it MUST be smaller than or equal to the 'right'.

top: The value of this member MUST specify the y-coordinate of the upper-left corner of the rectangle relative to the top edge of the form. This value MUST be an integer greater than or equal to 0 and it MUST be smaller than or equal to the 'bottom'.

right: The value of this member MUST specify the x-coordinate of the lower-right corner of the rectangle relative to the left edge of the form. This value MUST be greater than or equal to 'left'.

bottom: The value of this member MUST specify the y-coordinate of the lower-right corner of the rectangle relative to the top edge of the form. This value MUST be greater than or equal to 'top'.

2.2.1.1.6 SIZE

The SIZE structure defines the area of a form, with a width and height in thousandth-of-a-millimeter units.

```
typedef struct {
  long cx;
  long cy;
} SIZE;
```

cx: The value of this member MUST specify the width, and it MUST be an integer greater than or equal to 0.

cy: The value of this member MUST specify the height, and it MUST be an integer greater than or equal to 0.

2.2.1.1.7 STRING_HANDLE

The STRING_HANDLE serves as an RPC binding handle for methods that do not specify a PRINTER_HANDLE parameter. RPC binding handles are specified in [C706].

This type is declared as follows:

```
typedef [handle] wchar_t* STRING_HANDLE;
```

To build the binding handle for those methods, RPC requires an RPC protocol sequence, a network address, and an endpoint. Both the RPC protocol sequence and the endpoint are bound to the RPC interface; they MUST be named pipes and **\pipe\spoolss**, respectively. The network address MUST be defined by the printer or print server name. The printer name can be in the form **\\server\printer** (for rules governing printer names, see section 2.2.4.14), and the server MUST be used as the network address.

2.2.1.2 Containers

2.2.1.2.1 DEVMODE_CONTAINER

The DEVMODE_CONTAINER structure specifies a _DEVMODE structure (section 2.2.2.1), which contains data for the initialization of a print device by a printer driver.

```
typedef struct _DEVMODE_CONTAINER {
  DWORD cbBuf;
  [size_is(cbBuf), unique] BYTE* pDevMode;
} DEVMODE_CONTAINER;
```

cbBuf: The size, in bytes, of the buffer pointed to by the **pDevMode** member.

pDevMode: An optional pointer to a variable-length, custom-marshaled _DEVMODE structure. The NULL value MUST be used to indicate that the default initialization data for the printer driver SHOULD be used.

2.2.1.2.2 DOC_INFO_CONTAINER

The DOC_INFO_CONTAINER structure provides information about the document to be printed, using the DOC_INFO_1 structure.

```
typedef struct _DOC_INFO_CONTAINER {
  DWORD Level;
  [switch_is(Level)] union {
    [case(1)]
      DOC_INFO_1* pDocInfo1;
  } DocInfo;
} DOC_INFO_CONTAINER;
```

Level: This member specifies the information level of the **DocInfo** member data. The value of this member MUST be set to 0x00000001.

DocInfo: This member MUST define document properties, using an information structure that MUST correspond to the value of the **Level** member.

pDocInfo1: This member MUST be a non-NULL pointer to a DOC_INFO_1 structure that describes the document that will be printed. Details are specified in section 2.2.1.4.

2.2.1.2.3 DRIVER_CONTAINER

The DRIVER_CONTAINER structure provides information about printer drivers by using **DRIVER_INFO** structures (section 2.2.1.5). The **DriverInfo** member specifies the structure that defines the properties of a printer driver.

```
typedef struct _DRIVER_CONTAINER {
  DWORD Level;
  [switch is(Level)] union {
    [case(1)]
      DRIVER_INFO_1* pNotUsed;
    [case(2)]
      DRIVER INFO 2* Level2;
    [case(3)]
      RPC DRIVER INFO 3* Level3;
    [case(4)]
      RPC_DRIVER_INFO_4* Level4;
    [case(6)]
      RPC_DRIVER_INFO_6* Level6;
```

```
    [case(8)]
      RPC_DRIVER_INFO_8* Level8;
  } DriverInfo;
} DRIVER_CONTAINER;
```

Level: Specifies the information level of the **DriverInfo** data. The value of this member MUST be in the range 0x00000002 to 0x00000004 inclusive, 0x00000006, or 0x00000008.

DriverInfo: Defines printer driver properties by using an information structure that corresponds to the value of the **Level** member.

pNotUsed: A pointer to a structure that is specified only as a placeholder in the IDL and MUST be ignored.

Level2: If the **Level** member is 0x00000002, this member is a pointer to a DRIVER_INFO_2 structure that provides printer driver information. For details, see section 2.2.1.5.2.

Level3: If the **Level** member is 0x00000003, this member is a pointer to an RPC_DRIVER_INFO_3 structure that provides printer driver information. For details, see section 2.2.1.5.3.

Level4: If the **Level** member is 0x00000004, this member is a pointer to an RPC_DRIVER_INFO_4 structure that provides printer driver information. For details, see section 2.2.1.5.4.

Level6: If the **Level** member is 0x00000006, this member is a pointer to an RPC_DRIVER_INFO_6 structure that provides printer driver information. For details, see section 2.2.1.5.5.

Level8: If the **Level** member is 0x00000008, this member is a pointer to an RPC_DRIVER_INFO_8 structure that provides printer driver information. For details, see section 2.2.1.5.6.

2.2.1.2.4 FORM_CONTAINER

The FORM_CONTAINER structure provides information about printer forms, using FORM_INFO structures. The **FormInfo** member specifies the structure that defines the printer form properties.

```
typedef struct _FORM_CONTAINER {
  DWORD Level;
  [switch_is(Level)] union {
    [case(1)]
      FORM_INFO_1* pFormInfo1;
    [case(2)]
      RPC_FORM_INFO_2* pFormInfo2;
  } FormInfo;
} FORM_CONTAINER;
```

Level: This member MUST specify the information level of the **FormInfo** data. The value of this member MUST be 0x00000001 or 0x00000002.

FormInfo: This member MUST define printer form properties, using an information structure that MUST correspond to the value of the **Level** member.

pFormInfo1: If the **Level** member is 0x00000001, this member MUST be a non-NULL pointer to a FORM_INFO_1 structure, which provides information about a printer form. For details, see section 2.2.1.6.1.

pFormInfo2: If the **Level** member is 0x00000002, this member MUST be a non-NULL pointer to a RPC_FORM_INFO_2 structure, which provides information about a printer form. For details, see section 2.2.1.6.2.

2.2.1.2.5 JOB_CONTAINER

The JOB_CONTAINER structure provides information about print jobs, using JOB_INFO structures. The **JobInfo** member specifies the structure that defines the print job properties.

```
typedef struct _JOB_CONTAINER {
  DWORD Level;
  [switch is(Level)] union {
    [case(1)]
      JOB_INFO_1* Level1;
    [case(2)]
      JOB_INFO_2* Level2;
    [case(3)]
      JOB_INFO_3* Level3;
    [case(4)]
      JOB_INFO_4* Level4;
  } JobInfo;
} JOB_CONTAINER;
```

Level: Specifies the information level of the **JobInfo** data. The value of this member MUST be in the range 0x00000001 to 0x00000004 inclusive.

JobInfo: Defines print job properties, using an information structure that corresponds to the value of the **Level** member.

Level1: If the **Level** member is 0x00000001, this member is a pointer to a JOB_INFO_1 structure that provides print job information. For details, see section 2.2.1.7.1.

Level2: If the **Level** member is 0x00000002, this member is a pointer to a JOB_INFO_2 structure that provides print job information. For details, see section 2.2.1.7.2.

Level3: If the **Level** member is 0x00000003, this member is a pointer to a JOB_INFO_3 structure that provides print job information. For details, see section 2.2.1.7.3.

Level4: If the **Level** member is 0x00000004, this member is a pointer to a JOB_INFO_4 structure that provides print job information. For details, see section 2.2.1.7.4.

2.2.1.2.6 MONITOR_CONTAINER

The MONITOR_CONTAINER structure provides information about **port monitors**, using MONITOR_INFO structures. The **MonitorInfo** member specifies the structure that defines the port monitor properties.

```
typedef struct  MONITOR_CONTAINER {
  DWORD Level;
  [switch is(Level)] union {
    [case(1)]
      MONITOR_INFO_1* pMonitorInfo1;
    [case(2)]
      MONITOR_INFO_2* pMonitorInfo2;
  } MonitorInfo;
} MONITOR_CONTAINER;
```

Level: Specifies the information level of the **MonitorInfo** data. The value of this member MUST be 0x00000001 or 0x00000002.

MonitorInfo: Defines port monitor properties, using an information structure that corresponds to the value of the **Level** member.

pMonitorInfo1: If the **Level** member is 0x00000001, this member is a pointer to a MONITOR_INFO_1 structure that provides information about a port monitor. For details, see section 2.2.1.8.1.

pMonitorInfo2: If the **Level** member is 0x00000002, this member is a pointer to a MONITOR_INFO_2 structure that provides information about a port monitor. For details, see section 2.2.1.8.2.

2.2.1.2.7 PORT_CONTAINER

The PORT_CONTAINER structure provides information about printer ports, using PORT_INFO structures.<4> The **PortInfo** member specifies the structure that defines the port properties.

```
typedef struct _PORT_CONTAINER {
  DWORD Level;
  [switch_is(0x00FFFFFF & Level)]
    union {
    [case(1)]
      PORT_INFO_1* pPortInfo1;
    [case(2)]
      PORT_INFO_2* pPortInfo2;
    [case(3)]
      PORT_INFO_3* pPortInfo3;
    [case(0x00FFFFFF)]
      PORT_INFO_FF* pPortInfoFF;
  } PortInfo;
} PORT_CONTAINER;
```

Level: Specifies the information level of the **PortInfo** data. The value of this member MUST be in the range 0x00000001 to 0x00000003 inclusive, or 0xFFFFFFFF.

PortInfo: Defines port properties, using an information structure that corresponds to the value of the **Level** member.

Note: Despite the bitwise AND of **Level** with 0x00FFFFFF, no values for **Level** are valid besides those specified.

pPortInfo1: If the **Level** member is 0x00000001, this member is a pointer to a PORT_INFO_1 structure that provides information about the printer port. For details, see section 2.2.1.9.1.

pPortInfo2: If the **Level** member is 0x00000002, this member is a pointer to a PORT_INFO_2 structure that provides information about the printer port. For details, see section 2.2.1.9.2.

pPortInfo3: If the **Level** member is 0x00000003, this member is a pointer to a PORT_INFO_3 structure that provides information about the printer port. For details, see section 2.2.1.9.3.

pPortInfoFF: If the **Level** member is 0xFFFFFFFF, this member is a pointer to a PORT_INFO_FF structure that provides information about the printer port. For details, see section 2.2.1.9.4.

2.2.1.2.8 PORT_VAR_CONTAINER

The PORT_VAR_CONTAINER structure provides information for supported printer port monitors.<5>

```
typedef struct  PORT_VAR_CONTAINER {
  DWORD cbMonitorData;
  [size_is(cbMonitorData), unique, disable_consistency_check]
    BYTE* pMonitorData;
} PORT_VAR_CONTAINER;
```

cbMonitorData: The size, in bytes, of the buffer that is pointed to by the **pMonitorData** member.

pMonitorData: An optional pointer to a block of data that is passed to the port monitor.

2.2.1.2.9 PRINTER_CONTAINER

The PRINTER_CONTAINER structure provides information about printer properties and state information, using PRINTER_INFO structures. The **PrinterInfo** member specifies the structure that defines the printer properties.

```
typedef struct _PRINTER_CONTAINER {
  DWORD Level;
  [switch_is(Level)] union {
    [case(0)]
      PRINTER_INFO_STRESS* pPrinterInfoStress;
    [case(1)]
      PRINTER_INFO_1* pPrinterInfo1;
    [case(2)]
      PRINTER_INFO_2* pPrinterInfo2;
    [case(3)]
      PRINTER_INFO_3* pPrinterInfo3;
    [case(4)]
      PRINTER_INFO_4* pPrinterInfo4;
    [case(5)]
      PRINTER_INFO_5* pPrinterInfo5;
    [case(6)]
      PRINTER_INFO_6* pPrinterInfo6;
    [case(7)]
      PRINTER_INFO_7* pPrinterInfo7;
    [case(8)]
      PRINTER_INFO_8* pPrinterInfo8;
    [case(9)]
      PRINTER_INFO_9* pPrinterInfo9;
  } PrinterInfo;
} PRINTER_CONTAINER;
```

Level: Specifies the information level of the **PrinterInfo** data. The value of this member MUST be in the range 0x00000000 to 0x00000009 inclusive.

PrinterInfo: Provides printer information using a container structure that corresponds to the value specified by the **Level** member.

pPrinterInfoStress: If the **Level** member is 0x00000000, this member is a pointer to a PRINTER_INFO_STRESS structure (section 2.2.1.10.1), which provides diagnostic printer information.

pPrinterInfo1: If the **Level** member is 0x00000001, this member is a pointer to a PRINTER_INFO_1 (section 2.2.1.10.2) structure, which provides printer information.

pPrinterInfo2: If the **Level** member is 0x00000002, this member is a pointer to a PRINTER_INFO_2 (section 2.2.1.10.3) structure, which provides detailed printer information.

pPrinterInfo3: If the **Level** member is 0x00000003, this member is a pointer to a PRINTER_INFO_3 (section 2.2.1.10.4) structure, which provides printer security information.

pPrinterInfo4: If the **Level** member is 0x00000004, this member is a pointer to a PRINTER_INFO_4 (section 2.2.1.10.5) structure, which provides a subset of the printer information.

pPrinterInfo5: If the **Level** member is 0x00000005, this member is a pointer to a PRINTER_INFO_5 (section 2.2.1.10.6) structure, which provides information about the printer attributes.

pPrinterInfo6: If the **Level** member is 0x00000006, this member is a pointer to a PRINTER_INFO_6 (section 2.2.1.10.7) structure, which provides information about the status of the printer.

pPrinterInfo7: If the **Level** member is 0x00000007, this member is a pointer to a PRINTER_INFO_7 (section 2.2.1.10.8) structure, which provides **directory service (DS)** information.

pPrinterInfo8: If the **Level** member is 0x00000008, this member is a pointer to a PRINTER_INFO_8 (section 2.2.1.10.9) structure, which provides information about the global printer driver settings for a printer.

pPrinterInfo9: If the **Level** member is 0x00000009, this member is a pointer to a PRINTER_INFO_9 (section 2.2.1.10.10) structure. The PRINTER_INFO_9 structure is not used remotely, but it is included in this structure to yield a compatible IDL file. The print server MUST respond with ERROR_NOT_SUPPORTED if it receives a PRINTER_CONTAINER structure with a **Level** value equal to 0x00000009.

2.2.1.2.10 RPC_BIDI_REQUEST_CONTAINER

The RPC_BIDI_REQUEST_CONTAINER structure is a container for a list of **bidirectional** requests.<6>

```
typedef struct _RPC_BIDI_REQUEST_CONTAINER {
  DWORD Version;
  DWORD Flags;
  DWORD Count;
  [size_is(Count), unique] RPC_BIDI_REQUEST_DATA aData[];
} RPC_BIDI_REQUEST_CONTAINER;
```

Version: The version of the bidirectional API schema. The value of this member MUST be 0x00000001.

Flags: A value that MUST be set to zero when sent and MUST be ignored on receipt.

Count: The number of bidirectional requests in the **aData** member.

aData: An array of RPC_BIDI_REQUEST_DATA structures. Each structure in this member contains a single bidirectional request. For details, see section 2.2.1.12.1.

2.2.1.2.11 RPC_BIDI_RESPONSE_CONTAINER

The RPC_BIDI_RESPONSE_CONTAINER structure is a container for a list of bidirectional responses.<7>

```
typedef struct _RPC_BIDI_RESPONSE_CONTAINER {
  DWORD Version;
  DWORD Flags;
  DWORD Count;
  [size_is(Count), unique] RPC_BIDI_RESPONSE_DATA aData[];
} RPC_BIDI_RESPONSE_CONTAINER;
```

Version: This member MUST contain the value that specifies the version of the bidirectional API **schema**. The value of this member MUST be 0x00000001.

Flags: This member is a set of flags that are reserved for system use. The value of this member MUST be set to zero when sent and MUST be ignored on receipt.

Count: This member MUST specify the number of bidirectional responses in the **aData** member.

aData: This member is an array of RPC_BIDI_RESPONSE_DATA structures. Each structure in this member MUST contain a single bidirectional response. For more information, see section 2.2.1.12.2.

2.2.1.2.12 RPC_BINARY_CONTAINER

The RPC_BINARY_CONTAINER structure is a container for binary printer data and is used in the RPC_BIDI_DATA (section 2.2.1.12.3) structure.<8>

```
typedef struct  RPC_BINARY_CONTAINER {
  DWORD cbBuf;
  [size_is(cbBuf), unique] BYTE* pszString;
} RPC_BINARY_CONTAINER;
```

cbBuf: This member MUST specify the size, in bytes, of the buffer that is pointed to by the **pszString** member.

pszString: This member MUST be a non-NULL pointer to an array of bytes that contain binary printer data.

2.2.1.2.13 SECURITY_CONTAINER

The SECURITY_CONTAINER structure specifies a SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6), which contains security information.

```
typedef struct SECURITY_CONTAINER {
  DWORD cbBuf;
  [size_is(cbBuf), unique] BYTE* pSecurity;
} SECURITY_CONTAINER;
```

cbBuf: The size, in bytes, of the buffer that is pointed to by the **pSecurity** member.

pSecurity: An optional pointer to a self-relative SECURITY_DESCRIPTOR structure.

2.2.1.2.14 SPLCLIENT_CONTAINER

The SPLCLIENT_CONTAINER structure contains an information structure that provides data about the connecting client.<9>

```
typedef struct _SPLCLIENT_CONTAINER {
  DWORD Level;
  [switch_is(Level)] union {
    [case(1)]
      SPLCLIENT_INFO_1* pClientInfo1;
    [case(2)]
      SPLCLIENT_INFO_2* pNotUsed1;
    [case(3)]
      SPLCLIENT_INFO_3* pNotUsed2;
  } ClientInfo;
} SPLCLIENT_CONTAINER;
```

Level: The information level that is used by the **ClientInfo** member to determine the information structure. The value MUST be 0x00000001.

ClientInfo: Client information in a structure that corresponds to the information level specified by the **Level** member.

pClientInfo1: A pointer to an SPLCLIENT_INFO_1 (section 2.2.1.11.1) information structure.

pNotUsed1: A pointer to a structure that is specified only as a placeholder in the IDL and MUST be ignored.

pNotUsed2: A pointer to a structure that is specified only as a placeholder in the IDL and MUST be ignored.

2.2.1.2.15 STRING_CONTAINER

The STRING_CONTAINER structure contains a string.<10>

```
typedef struct _STRING_CONTAINER {
  DWORD cbBuf;
  [size_is(cbBuf/2), unique] WCHAR* pszString;
} STRING_CONTAINER;
```

cbBuf: This member MUST specify the size, in bytes, of the buffer that is pointed to by the **pszString** member. The value of this number MUST be an even number.

pszString: This member MUST be a non-NULL pointer to a string. The string that is referenced by this member MUST NOT be empty.

2.2.1.2.16 SYSTEMTIME_CONTAINER

The SYSTEMTIME_CONTAINER structure is a container for a SYSTEMTIME structure that specifies a date and time using individual members for the month, day, year, weekday, hour, minute, second, and millisecond.<11>

```
typedef struct _SYSTEMTIME_CONTAINER {
  DWORD cbBuf;
  SYSTEMTIME* pSystemTime;
} SYSTEMTIME_CONTAINER;
```

cbBuf: This member MUST specify the size, in bytes, of the buffer that is pointed to by the **pSystemTime** member.

pSystemTime: This member MUST be a non-NULL pointer to a SYSTEMTIME structure.

2.2.1.2.17 RPC_BranchOfficeJobDataContainer

The RPC_BranchOfficeJobDataContainer structure is a container for an array of RPC_BranchOfficeJobData (section 2.2.1.15.2) structures.<12>

```
typedef struct {
  DWORD cJobDataEntries;
  [size is(cJobDataEntries), unique]
    RPC BranchOfficeJobData JobData[];
} RPC_BranchOfficeJobDataContainer;
```

cJobDataEntries: The number of RPC_BranchOfficeJobData structures in the **JobData** member.

JobData: An array of RPC_BranchOfficeJobData structures. Each structure in the array contains a single **Branch Office Print Remote Log Entry** (section 3.1.1).

2.2.1.3 Members in INFO Structures

This section specifies members that are commonly used in a consistent fashion in IDL-marshaled and custom-marshaled INFO structures. The individual INFO sections provide definitions only for the following:

- Members that are not defined in this section.
- Members that are not defined in corresponding INFO subsections within this section.
- Members whose definitions in their corresponding INFO structures differ from their definitions in this section and subsections.

The type of each member is specified in its corresponding INFO structure section.

pPrinterName: This member MUST be a non-NULL pointer to a string that MUST specify the name of a printer. For rules governing printer names, see section 2.2.4.14.

pServerName: This member MUST be a non-NULL pointer to a string that MUST specify the name of the server that hosts the printer. For rules governing server names, see section 2.2.4.16.

Reserved: This member is reserved for future use. The value of this member SHOULD be set to zero when sent and MUST be ignored on receipt.

dwReserved2: This member is reserved for future use. The value of this member SHOULD be set to zero when sent and MUST be ignored on receipt.

dwReserved3: This member is reserved for future use. The value of this member SHOULD be set to zero when sent and MUST be ignored on receipt.

2.2.1.3.1 DRIVER_INFO and RPC_DRIVER_INFO Members

This section describes members commonly used in DRIVER_INFO (section 2.2.1.5) and RPC_DRIVER_INFO (section 2.2.1.3.1) structures.

pName: A pointer to a string that specifies the name of the printer driver; for example, "QMS 810". For rules governing printer driver names, see section 2.2.4.3.

cVersion: An implementation-specific value that identifies the driver version and the operating system version for which the printer driver was written. The driver version contained by each printer driver object in the "List of Printer Drivers" described in section 3.1.1.<13>

pEnvironment: A pointer to a string that specifies the environment that the printer driver supports. For rules governing environment names, see section 2.2.4.4.

pDriverPath: A pointer to a string that specifies a file name or full path and file name for the file that contains the printer driver. For further information on driver files, see [MSDN-MPD]. For rules governing path names, see section 2.2.4.9.

pDataFile: A pointer to a string that specifies a file name or a full path and file name for the file that contains printer driver data. For further information on driver files, see [MSDN-MPD]. For rules governing path names, see section 2.2.4.9.

pConfigFile: A pointer to a string that specifies a file name or a full path and file name for the printer driver configuration module. For further information on driver files, see [MSDN-MPD]. For rules governing path names, see section 2.2.4.9.

pHelpFile: An optional pointer to a string that specifies a file name or a full path and file name for the printer driver help file. For further information on driver files, see [MSDN-MPD]. For rules governing path names, see section 2.2.4.9.

pMonitorName: An optional pointer to a string that specifies a **language monitor**. For rules governing monitor names, see section 2.2.4.8.<14>

pDefaultDataType: An optional pointer to a string that specifies the default data type of print jobs created with this driver (for example, **enhanced metafile spool format (EMFSPOOL)** or RAW Format). For rules governing data type names, see section 2.2.4.2.

cchDependentFiles: The number of characters in the multisz pointed to by **pDependentFiles**.

pDependentFiles: An optional pointer to a multisz that MUST specify the names of the files that the printer driver is dependent on. If specified, this list MUST include at least one file name and SHOULD be ordered as follows:

- The file name of the **printer driver manifest**, if present.
- If the printer driver is a **derived printer driver**, the names of all the files the derived printer driver depends on. If the printer driver is not a derived printer driver, all of the other files the printer driver depends on.
- If the printer driver is a derived printer driver, the file name of the printer driver manifest of the corresponding **class printer driver**.
- If the printer driver is a derived printer driver, the names of all of the files the corresponding class printer driver depends on.<15>

cchPreviousNames: The value of this member MUST be the number of characters in the multisz pointed to by **pszzPreviousNames**.

pszzPreviousNames: An optional pointer to a multisz that MUST specify any previous printer drivers that are compatible with this driver.

dwlDriverVersion: The printer driver version number. The format of this number is specified by each printer driver manufacturer. A print client MAY use this value to determine whether a printer driver on the print server matches the version available on the client.<16>

ftDriverDate: The value of this member MUST be the manufacturer build date of the printer driver. The FILETIME format is specified in [MS-DTYP] section 2.3.3.

pMfgName: An optional pointer to a string that specifies the manufacturer's name.

pOEMUrl: An optional pointer to a string that specifies the URL for the manufacturer of the printer driver.

pHardwareID: An optional pointer to a string that specifies the hardware identifier for the printer driver.

pProvider: An optional pointer to a string that specifies the publisher of the printer driver.

2.2.1.3.2 FORM_INFO and RPC_FORM_INFO Members

This section describes the members that are commonly used in FORM_INFO and RPC_FORM_INFO structures.

Flags: This member MUST specify the form property. The value of this member MUST be a value from the following table.

Name/Value	Meaning
FORM_USER 0x00000000	If the value of this member is **FORM_USER**, the form has been defined by the user. Forms that have this flag set are defined in the registry.
FORM_BUILTIN 0x00000001	If the value of this member is **FORM_BUILTIN**, the form is part of the spooler. Form definitions that have this flag set do not appear in the registry.
FORM_PRINTER 0x00000002	If the value of this member is **FORM_PRINTER**, the form is associated with a particular printer and its definition appears in the registry.

pName: This member MUST be a non-NULL pointer to a string that MUST specify the form name. For rules governing form names, see section 2.2.4.5.

Size: The value of this member MUST specify the form's width and height in thousandths of millimeters using a SIZE structure.

ImageableArea: This member MUST specify the part of the form that the printer can print on as a rectangle in thousandths of millimeters using a RECTL structure.

2.2.1.3.3 JOB_INFO Members

This section describes members commonly used in JOB_INFO structures.

pMachineName: This member is a pointer to a string that specifies the name of a server that hosts a printer. For rules governing server names, see section 2.2.4.16.

pUserName: This member is an optional pointer to a string that specifies the name of a user that owns a print job. For rules governing user names, see section 2.2.4.17.

pNotifyName: This member is an optional pointer to a string that specifies the name of a user to be notified when a job is complete or when an error occurs while printing a job. For rules governing user names, see section 2.2.4.17.

pDocument: This member is an optional pointer to a string that specifies the name of a print job.

pDatatype: This member is a pointer to a string that specifies the type of data that a printing application sends to a printer in a print job. The identified data type MUST be supported by the print processor that is associated with the printer that is processing the job. For rules governing data type names, see section 2.2.4.2.

pPrintProcessor: This member is a pointer to a string that specifies the name of a print processor that is used to print a job. For rules governing print processor names, see section 2.2.4.11.

pParameters: This member is an optional pointer to a string that specifies default print processor parameters.

pDriverName: This member is an optional pointer to a string that specifies the name of a printer driver to process a print job. For rules governing printer driver names, see section 2.2.4.3.

pDevMode: This member is an optional pointer to a truncated DEVMODE structure (section 2.2.1.1.1), and MUST be ignored on receipt. Actual **DEVMODE** data is passed to a method via a custom-marshaled _DEVMODE structure (section 2.2.2.1) in a DEVMODE_CONTAINER (section 2.2.1.2.1).

pSecurityDescriptor: This member is an optional pointer to a SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6), and MUST be ignored on receipt. Actual **SECURITY_DESCRIPTOR** data is passed to a method via a self-relative SECURITY_DESCRIPTOR structure in a SECURITY_CONTAINER (section 2.2.1.2.13).

JobId: This member contains an identifier for a print job.

pStatus: This member is an optional pointer to a string that describes job status. The text is implementation-specific and can be displayed to the user, but it MUST NOT have any other functional effect. An example of job status is "Cannot print - Black ink must be replaced."

Status: This member specifies job status. The value of this member is the result of a bitwise OR of zero or more of the job status values defined in section 2.2.3.12.

Client applications can display the job status to a user. It is an implementation-specific string and SHOULD support all job status descriptions specified in section 2.2.3.12 for all corresponding status bits. If **pStatus** is not NULL, the string that is pointed to by **pStatus** SHOULD be displayed instead.

Priority: This member specifies information about job priority. The value of this member MUST be a decimal number from 0 through 99, inclusive.

Position: This member specifies a job's position in a queue, where one represents the next job that will be printed.

TotalPages: This member specifies the number of pages a document contains. It can be zero.

PagesPrinted: This member specifies the number of pages that have been printed. It can be zero.

Submitted: This member is a SYSTEMTIME structure ([MS-DTYP] section 2.3.13) that specifies when a document was spooled.

StartTime: This member specifies the earliest time that a printer can print a job. The time is expressed as the number of minutes after 12:00 AM GMT within a 24-hour boundary.

UntilTime: This member specifies the latest time that the printer can print a job. The time is expressed as the number of minutes after 12:00 AM GMT within a 24-hour boundary.

Size: This member specifies the size of a job, in bytes.

Time: This member specifies the number of milliseconds that have elapsed since printing began.

2.2.1.3.4 MONITOR_INFO Members

This section describes the members that are commonly used in MONITOR_INFO structures.

pName: This member MUST be a non-NULL pointer to a string that MUST specify the name of the port monitor. For rules governing port monitor names, see section 2.2.4.8.

2.2.1.3.5 PORT_INFO Members

This section describes members commonly used in PORT_INFO structures.

pPortName: A pointer to a string that specifies a supported printer port. For rules governing port names, see section 2.2.4.10.

2.2.1.3.6 PRINTER_INFO Members

This section describes members commonly used in PRINTER_INFO structures.

pDescription: This member is an optional pointer to a string that specifies a description of the printer.<17>

pComment: This member is an optional pointer to a string that MUST specify additional information about the printer.<18>

Status: This member specifies the printer status. It is the result of a bitwise OR of zero or more printer status values (section 2.2.3.12).

Attributes: This member specifies printer attributes. It is the result of a bitwise OR of zero or more printer attribute values (section 2.2.3.12).

pDevMode: This member is an optional pointer to a truncated DEVMODE structure (section 2.2.1.1.1), and MUST be ignored on receipt. Actual **DEVMODE** data is passed to a method via a custom-marshaled _DEVMODE structure (section 2.2.2.1) in a DEVMODE_CONTAINER (section 2.2.1.2.1).

pSecurityDescriptor: This member is an optional pointer to a SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6), and MUST be ignored on receipt. Actual **SECURITY_DESCRIPTOR** data is

passed to a method via a self-relative SECURITY_DESCRIPTOR structure in a SECURITY_CONTAINER (section 2.2.1.2.13).

pPortName: This member is a pointer to a string that specifies the port(s) used to transmit data to a printer. For rules governing port names, see section 2.2.4.10.

2.2.1.3.7 SPLCLIENT_INFO Members

This section describes members commonly used in SPLCLIENT_INFO structures.

pMachineName: This member is a pointer to a string that provides the client computer name. Client computer names are governed by the same rules as server names (section 2.2.4.16).

pUserName: This member is a pointer to a string that provides a user name.

dwBuildNum: The value of this member specifies the build number of the client operating system.

dwMajorVersion: The value of this member is the implementation-specific major version number of the client operating system.<19>

dwMinorVersion: The value of this member is the implementation-specific minor version number of the client operating system.<20>

wProcessorArchitecture: The value of this member is the implementation-specific identifier for the client system's processor architecture.<21> The value of this member SHOULD be ignored on receipt.

2.2.1.4 DOC_INFO_1

The DOC_INFO_1 structure describes a document that will be printed.

```
typedef struct _DOC_INFO_1 {
  [string] wchar_t* pDocName;
  [string] wchar t* pOutputFile;
  [string] wchar t* pDatatype;
} DOC_INFO_1;
```

pDocName: An optional pointer to a string that provides the name of the document. If this member is NULL, the print server SHOULD use an implementation-specific default job name.<22>

pOutputFile: An optional pointer to a string that specifies the name of an output file. For rules governing path names, see section 2.2.4.9.

pDatatype: An optional pointer to a string that identifies the type of data used to record the document. For rules governing data type names, see section 2.2.4.2.

2.2.1.5 DRIVER_INFO

2.2.1.5.1 DRIVER_INFO_1

The DRIVER_INFO_1 structure provides information about a printer driver.

```
typedef struct _DRIVER_INFO_1 {
  [string] wchar_t* pName;
} DRIVER_INFO_1;
```

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.5.2 DRIVER_INFO_2

The DRIVER_INFO_2 structure provides information about a printer driver.

```
typedef struct _DRIVER_INFO_2 {
  DWORD cVersion;
  [string] wchar_t* pName;
  [string] wchar_t* pEnvironment;
  [string] wchar_t* pDriverPath;
  [string] wchar_t* pDataFile;
  [string] wchar t* pConfigFile;
} DRIVER_INFO_2;
```

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.5.3 RPC_DRIVER_INFO_3

The RPC_DRIVER_INFO_3 structure provides information about a printer driver.<23>

```
typedef struct  RPC DRIVER INFO 3 {
  DWORD cVersion;
  [string] wchar_t* pName;
  [string] wchar_t* pEnvironment;
  [string] wchar_t* pDriverPath;
  [string] wchar_t* pDataFile;
  [string] wchar t* pConfigFile;
  [string] wchar_t* pHelpFile;
  [string] wchar_t* pMonitorName;
  [string] wchar_t* pDefaultDataType;
  DWORD cchDependentFiles;
  [size_is(cchDependentFiles), unique]
    wchar t* pDependentFiles;
} RPC_DRIVER_INFO_3;
```

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.5.4 RPC_DRIVER_INFO_4

The RPC_DRIVER_INFO_4 structure provides information about a printer driver.<24>

```
typedef struct _RPC_DRIVER_INFO_4 {
  DWORD cVersion;
  [string] wchar_t* pName;
  [string] wchar_t* pEnvironment;
  [string] wchar_t* pDriverPath;
  [string] wchar t* pDataFile;
  [string] wchar t* pConfigFile;
  [string] wchar_t* pHelpFile;
  [string] wchar_t* pMonitorName;
  [string] wchar_t* pDefaultDataType;
  DWORD cchDependentFiles;
  [size_is(cchDependentFiles), unique]
    wchar t* pDependentFiles;
  DWORD cchPreviousNames;
  [size_is(cchPreviousNames), unique]
    wchar_t* pszzPreviousNames;
} RPC_DRIVER_INFO_4;
```

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.5.5 RPC_DRIVER_INFO_6

The RPC_DRIVER_INFO_6 structure provides extended printer driver information.<25>

```
typedef struct _RPC_DRIVER_INFO_6 {
  DWORD cVersion;
  [string] wchar t* pName;
  [string] wchar_t* pEnvironment;
  [string] wchar_t* pDriverPath;
  [string] wchar_t* pDataFile;
  [string] wchar t* pConfigFile;
  [string] wchar_t* pHelpFile;
  [string] wchar t* pMonitorName;
  [string] wchar_t* pDefaultDataType;
  DWORD cchDependentFiles;
  [size_is(cchDependentFiles), unique]
    wchar_t* pDependentFiles;
  DWORD cchPreviousNames;
  [size_is(cchPreviousNames), unique]
    wchar_t* pszzPreviousNames;
  FILETIME ftDriverDate;
  DWORDLONG dwlDriverVersion;
  [string] wchar_t* pMfgName;
  [string] wchar t* pOEMUrl;
  [string] wchar_t* pHardwareID;
  [string] wchar_t* pProvider;
} RPC_DRIVER_INFO_6;
```

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.5.6 RPC_DRIVER_INFO_8

The RPC_DRIVER_INFO_8 structure specifies extended printer driver information.<26>

```
typedef struct _RPC_DRIVER_INFO_8 {
  DWORD cVersion;
  [string] wchar_t* pName;
  [string] wchar t* pEnvironment;
  [string] wchar_t* pDriverPath;
  [string] wchar_t* pDataFile;
  [string] wchar_t* pConfigFile;
  [string] wchar_t* pHelpFile;
  [string] wchar_t* pMonitorName;
  [string] wchar_t* pDefaultDataType;
  DWORD cchDependentFiles;
  [size_is(cchDependentFiles), unique]
    wchar_t* pDependentFiles;
  DWORD cchPreviousNames;
  [size is(cchPreviousNames), unique]
    wchar_t* pszzPreviousNames;
  FILETIME ftDriverDate;
  DWORDLONG dwlDriverVersion;
  [string] wchar_t* pMfgName;
  [string] wchar_t* pOEMUrl;
  [string] wchar t* pHardwareID;
  [string] wchar_t* pProvider;
  [string] wchar t* pPrintProcessor;
  [string] wchar_t* pVendorSetup;
  DWORD cchColorProfiles;
  [size_is(cchColorProfiles), unique]
    wchar t* pszzColorProfiles;
  [string] wchar_t* pInfPath;
  DWORD dwPrinterDriverAttributes;
  DWORD cchCoreDependencies;
  [size_is(cchCoreDependencies), unique]
```

```
    wchar_t* pszzCoreDriverDependencies;
  FILETIME ftMinInboxDriverVerDate;
  DWORDLONG dwlMinInboxDriverVerVersion;
} RPC_DRIVER_INFO_8;
```

pPrintProcessor: A pointer to a string that specifies the print processor for this printer. For rules governing print processor names, see section 2.2.4.11.

pVendorSetup: An optional pointer to a string that specifies the name of the vendor setup file used for hardware vendor-provided custom setup.

cchColorProfiles: The number of characters in the multisz structure pointed to by the **pszzColorProfiles** member.

pszzColorProfiles: An optional pointer to a multisz structure that contains the names of all **color profile** files for this driver.

pInfPath: An optional pointer to a string that specifies the path to the installation configuration file in the **driver store** that identifies the printer driver for installation.<27>

When used as an input parameter in a call to RpcAddPrinterDriverEx (section 3.1.4.4.8), this pointer MUST be NULL. When used as output in the custom-marshaled form of this structure (_DRIVER_INFO_8 section 2.2.2.4.8), the server SHOULD set this value for package-aware drivers.

dwPrinterDriverAttributes: A bit field that specifies attributes of the printer driver.

When used as an input parameter in a call to RpcAddPrinterDriverEx, this value MUST be zero. When used as output in the custom-marshaled form of this structure (_DRIVER_INFO_8), this bit field is defined as one or more of the bit flags described in the following table.

Value	Meaning
PRINTER_DRIVER_PACKAGE_AWARE 0x00000001	The printer driver is part of a driver package.<28>
PRINTER_DRIVER_XPS 0x00000002	The printer driver supports the Microsoft **XML Paper Specification (XPS)** format described in [MSFT-XMLPAPER].<29><30>
PRINTER_DRIVER_SANDBOX_ENABLED 0x00000004	The printer driver is compatible with **printer driver isolation**.<31><32>
PRINTER_DRIVER_CLASS 0x00000008	The printer driver is a class printer driver.<33><34>
PRINTER_DRIVER_DERIVED 0x00000010	The printer driver is a derived printer driver.<35><36>
PRINTER_DRIVER_NOT_SHAREABLE 0x00000020	Printers using the printer driver cannot be shared.<37><38>
PRINTER_DRIVER_CATEGORY_FAX 0x00000040	The printer driver is intended for use with **fax printers**.<39><40>
PRINTER_DRIVER_CATEGORY_FILE 00x00000080	The printer driver is intended for use with **file printers**.<41><42>
PRINTER_DRIVER_CATEGORY_VIRTUAL	The printer driver is intended for use with **virtual**

Value	Meaning
0x00000100	**printers**.<43><44>
PRINTER_DRIVER_CATEGORY_SERVICE 0x00000200	The printer driver is intended for use with **service printers**.<45><46>
PRINTER_DRIVER_SOFT_RESET_REQUIRED 0x00000400	Printers using this printer driver SHOULD use an implementation-specific mechanism to reset the printer when a print job is canceled.<47><48>
PRINTER_DRIVER_CATEGORY_3D 0x00000200	The printer driver is intended for use with **3D printers**.<49><50>

cchCoreDependencies: The number of characters in the multisz structure pointed to by the **pszzCoreDriverDependencies** member.

pszzCoreDriverDependencies: An optional pointer to a multisz structure that contains the names of the core dependencies as specified by the installation configuration file. These names specify the core sections of the installation configuration file that are required by the printer driver.<51>

ftMinInboxDriverVerDate: The minimum date version that is required in order for any **core printer driver** to be used as a dependency as listed in the multisz structure pointed to by the **pszzCoreDriverDependencies** member. The value of this member MUST be specified in the same format as the **ftDriverDate** member.<52>

dwlMinInboxDriverVerVersion: The minimum file version that is required for any core printer driver to be used as a dependency as listed in the multisz structure pointed to by the **pszzCoreDriverDependencies** member. The value of this member MUST be specified in the same format as the **dwlDriverVersion** member.<53>

All members not defined in this section are specified in sections 2.2.1.3.1 and 2.2.1.3.

2.2.1.6 FORM_INFO

2.2.1.6.1 FORM_INFO_1

The FORM_INFO_1 structure provides information about a printer form.

```
typedef struct _FORM_INFO_1 {
  DWORD Flags;
  [string] wchar t* pName;
  SIZE Size;
  RECTL ImageableArea;
} FORM_INFO_1;
```

All members not defined in this section are specified in sections 2.2.1.3.2 and 2.2.1.3.

2.2.1.6.2 RPC_FORM_INFO_2

The RPC_FORM_INFO_2 structure provides information about a printer form that includes its origin, dimensions, the dimensions of its printable area, and its display name.<54>

```
typedef struct _RPC_FORM_INFO_2 {
  DWORD Flags;
  [string, unique] const wchar_t* pName;
  SIZE Size;
  RECTL ImageableArea;
```

```
  [string, unique] const char* pKeyword;
  DWORD StringType;
  [string, unique] const wchar_t* pMuiDll;
  DWORD dwResourceId;
  [string, unique] const wchar_t* pDisplayName;
  LANGID wLangID;
} RPC_FORM_INFO_2;
```

pKeyword: This member MUST be set to NULL by the client if the value of the **Flags** member is set to **FORM_BUILTIN**, or this member MUST be a non-NULL pointer to a string that MUST specify a unique, localization-independent identifier for this form.<55>

StringType: The value of this member MUST specify how a form's display name is passed. The value of this member MUST be a value from the following table.

Name/Value	Meaning
STRING_NONE 0x00000001	Use the default display name, a string that is pointed to by the **pName** member. No localized display name exists.
STRING_MUIDLL 0x00000002	Load the form name from the library of string resources that is identified by the **pMuiDll** member. The **dwResourceId** member specifies the ID of the form name string in that library.
STRING_LANGPAIR 0x00000004	Use the form name, a string that is pointed to by the **pDisplayName** member, and the language that is identified by the **wLangID** member.

pMuiDll: This member MUST be a NULL pointer and MUST be ignored on receipt if **StringType** is not equal to **STRING_MUIDLL**, or it MUST be a non-NULL pointer to a string that contains the name of a library of string resources. String resources MAY<56> be localized into multiple languages.

dwResourceId: The value of this member SHOULD be set to zero when sent and ignored on receipt if the value of the **StringType** member is not equal to **STRING_MUIDLL**; otherwise, the value of this member MUST specify the string resource ID of the form name in the library that is identified by the string that is pointed to by the **pMuiDll** member.

pDisplayName: This member MUST be a NULL pointer and ignored on receipt if **StringType** is not equal to **STRING_LANGPAIR**; otherwise, this member MUST be a non-NULL pointer to a string that MUST specify the form name.

wLangID: The value of this member SHOULD be set to zero when sent and ignored on receipt if **StringType** is not equal to **STRING_LANGPAIR**; otherwise, the value of this member MUST be the Language Identifier of the **pDisplayName** member as specified in [MS-LCID].

All members not defined in this section are specified in sections 2.2.1.3.2 and 2.2.1.3.

2.2.1.7 JOB_INFO

2.2.1.7.1 JOB_INFO_1

The JOB_INFO_1 structure provides information about a print job.

```
typedef struct _JOB_INFO_1 {
  DWORD JobId;
  [string] wchar t* pPrinterName;
  [string] wchar_t* pMachineName;
  [string] wchar t* pUserName;
  [string] wchar_t* pDocument;
```

```
  [string] wchar_t* pDatatype;
  [string] wchar_t* pStatus;
  DWORD Status;
  DWORD Priority;
  DWORD Position;
  DWORD TotalPages;
  DWORD PagesPrinted;
  SYSTEMTIME Submitted;
} JOB_INFO_1;
```

All members not defined in this section are specified in sections 2.2.1.3.3 and 2.2.1.3.

2.2.1.7.2 JOB_INFO_2

The JOB_INFO_2 structure provides information about a print job.

```
typedef struct _JOB_INFO_2 {
  DWORD JobId;
  [string] wchar t* pPrinterName;
  [string] wchar_t* pMachineName;
  [string] wchar t* pUserName;
  [string] wchar_t* pDocument;
  [string] wchar_t* pNotifyName;
  [string] wchar_t* pDatatype;
  [string] wchar t* pPrintProcessor;
  [string] wchar_t* pParameters;
  [string] wchar t* pDriverName;
  ULONG_PTR pDevMode;
  [string] wchar_t* pStatus;
  ULONG_PTR pSecurityDescriptor;
  DWORD Status;
  DWORD Priority;
  DWORD Position;
  DWORD StartTime;
  DWORD UntilTime;
  DWORD TotalPages;
  DWORD Size;
  SYSTEMTIME Submitted;
  DWORD Time;
  DWORD PagesPrinted;
} JOB_INFO_2;
```

All members not defined in this section are specified in sections 2.2.1.3.3 and 2.2.1.3.

2.2.1.7.3 JOB_INFO_3

The JOB_INFO_3 structure provides information about a print job.<57>

```
typedef struct _JOB_INFO_3 {
  DWORD JobId;
  DWORD NextJobId;
  DWORD Reserved;
} JOB_INFO_3;
```

NextJobId: An identifier that specifies the print job in the queue following the job identified by the **JobId** member. A value of zero indicates that there are no jobs following the job identified by the **JobId** member.

When used as input to RpcSetJob (section 3.1.4.3.1) to alter the order of print jobs and link them together, **JobId** and **NextJobId** MUST be nonzero and SHOULD be obtained through RpcEnumJobs (section 3.1.4.3.3) or RpcGetJob (section 3.1.4.3.2).

All members not defined in this section are specified in sections 2.2.1.3.3 and 2.2.1.3.

2.2.1.7.4 JOB_INFO_4

The JOB_INFO_4 structure provides information about a print job.<58>

```
typedef struct  JOB INFO 4 {
  DWORD JobId;
  [string] wchar t* pPrinterName;
  [string] wchar_t* pMachineName;
  [string] wchar_t* pUserName;
  [string] wchar t* pDocument;
  [string] wchar_t* pNotifyName;
  [string] wchar t* pDatatype;
  [string] wchar_t* pPrintProcessor;
  [string] wchar_t* pParameters;
  [string] wchar_t* pDriverName;
  ULONG PTR pDevMode;
  [string] wchar t* pStatus;
  ULONG PTR pSecurityDescriptor;
  DWORD Status;
  DWORD Priority;
  DWORD Position;
  DWORD StartTime;
  DWORD UntilTime;
  DWORD TotalPages;
  DWORD Size;
  SYSTEMTIME Submitted;
  DWORD Time;
  DWORD PagesPrinted;
  long SizeHigh;
} JOB_INFO_4;
```

SizeHigh: This member specifies the high-order 32 bits of a 64-bit unsigned integer that specifies the size of the job, in bytes.

All members not defined in this section are specified in sections 2.2.1.3.3 and 2.2.1.3.

2.2.1.8 MONITOR_INFO

2.2.1.8.1 MONITOR_INFO_1

The MONITOR_INFO_1 structure provides information about a monitor.

```
typedef struct _MONITOR_INFO_1 {
  [string] wchar_t* pName;
} MONITOR_INFO_1;
```

All members not defined in this section are specified in sections 2.2.1.3.4 and 2.2.1.3.

2.2.1.8.2 MONITOR_INFO_2

The MONITOR_INFO_2 structure provides information about a monitor.

```
typedef struct _MONITOR_INFO_2 {
  [string] wchar t* pName;
  [string] wchar_t* pEnvironment;
  [string] wchar_t* pDLLName;
} MONITOR_INFO_2;
```

pEnvironment: This member MUST be a non-NULL pointer to a string that MUST specify the environment that the monitor supports. The environment specified MUST match the print server's operating system. For rules governing environment names and Windows behaviors, see section 2.2.4.4.

pDLLName: This member MUST be a non-NULL pointer to a string that MUST specify the name of the port monitor executable object.

All members not defined in this section are specified in sections 2.2.1.3.4 and 2.2.1.3.

2.2.1.9 PORT_INFO

2.2.1.9.1 PORT_INFO_1

The PORT_INFO_1 structure provides information about a port.

```
typedef struct  PORT_INFO_1 {
  [string] wchar_t* pPortName;
} PORT_INFO_1;
```

All members not defined in this section are specified in sections 2.2.1.3.5 and 2.2.1.3.

2.2.1.9.2 PORT_INFO_2

The PORT_INFO_2 structure provides information about a port.

```
typedef struct _PORT_INFO_2 {
  [string] wchar_t* pPortName;
  [string] wchar_t* pMonitorName;
  [string] wchar_t* pDescription;
  DWORD fPortType;
  DWORD Reserved;
} PORT_INFO_2;
```

pMonitorName: A pointer to a string that specifies an installed port monitor. For rules governing port monitor names, see section 2.2.4.8.

pDescription: An optional pointer to a string that specifies additional implementation-specific information about the printer port.<59>

fPortType: A bit field that specifies attributes of the printer port. These flags can be combined to specify multiple attributes.

Value	Meaning
PORT_TYPE_WRITE 0x00000001	The port can be written to.
PORT_TYPE_READ 0x00000002	The port can be read from.
PORT_TYPE_REDIRECTED 0x00000004	The port is a **Terminal Services** redirected port.
PORT_TYPE_NET_ATTACHED 0x00000008	The port is a network TCP/IP port.

All members not defined in this section are specified in sections 2.2.1.3.5 and 2.2.1.3.

2.2.1.9.3 PORT_INFO_3

The PORT_INFO_3 structure provides information about a port.<60>

```
typedef struct  PORT INFO 3 {
  DWORD dwStatus;
  [string] wchar t* pszStatus;
  DWORD dwSeverity;
} PORT_INFO_3;
```

dwStatus: The new port status. This value MUST be one of the following.

Name/value	Meaning
PORT_STATUS_CLEAR 0x00000000	Clears the printer port status.
PORT_STATUS_OFFLINE 0x00000001	The port's printer is offline.
PORT_STATUS_PAPER_JAM 0x00000002	The port's printer has a paper jam.
PORT_STATUS_PAPER_OUT 0x00000003	The port's printer is out of paper.
PORT_STATUS_OUTPUT_BIN_FULL 0x00000004	The port's printer's output bin is full.
PORT_STATUS_PAPER_PROBLEM 0x00000005	The port's printer has a paper problem.
PORT_STATUS_NO_TONER 0x00000006	The port's printer is out of toner.
PORT_STATUS_DOOR_OPEN 0x00000007	The door of the port's printer is open.
PORT_STATUS_USER_INTERVENTION 0x00000008	The port's printer requires user intervention.
PORT_STATUS_OUT_OF_MEMORY 0x00000009	The port's printer is out of memory.
PORT_STATUS_TONER_LOW 0x0000000A	The port's printer is low on toner.
PORT_STATUS_WARMING_UP 0x0000000B	The port's printer is warming up.
PORT_STATUS_POWER_SAVE 0x0000000C	The port's printer is in a power-conservation mode.

pszStatus: An optional pointer to a string that specifies a status description.

dwSeverity: The severity of the port status value. This value MUST be one of the following.

Name/value	Meaning
PORT_STATUS_TYPE_ERROR 0x00000001	The port status value indicates an error.
PORT_STATUS_TYPE_WARNING 0x00000002	The port status value is a warning.
PORT_STATUS_TYPE_INFO 0x00000003	The port status value is informational.

All members not defined in this section are specified in sections 2.2.1.3.5 and 2.2.1.3.

2.2.1.9.4 PORT_INFO_FF

The PORT_INFO_FF is used to communicate port information to a local port monitor.<61>

```
typedef struct  PORT_INFO_FF {
  [string] wchar_t* pPortName;
  DWORD cbMonitorData;
  BYTE* pMonitorData;
} PORT_INFO_FF;
```

cbMonitorData: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

pMonitorData: A pointer that SHOULD be set to NULL when sent and MUST be ignored on receipt.

All members not defined in this section are specified in sections 2.2.1.3.5 and 2.2.1.3.

2.2.1.10 PRINTER_INFO

2.2.1.10.1 PRINTER_INFO_STRESS

The PRINTER_INFO_STRESS structure provides diagnostic printer information used for **print system remote protocol stress analysis**.<62>

```
typedef struct _PRINTER_INFO_STRESS {
  [string] wchar_t* pPrinterName;
  [string] wchar_t* pServerName;
  DWORD cJobs;
  DWORD cTotalJobs;
  DWORD cTotalBytes;
  SYSTEMTIME stUpTime;
  DWORD MaxcRef;
  DWORD cTotalPagesPrinted;
  DWORD dwGetVersion;
  DWORD fFreeBuild;
  DWORD cSpooling;
  DWORD cMaxSpooling;
  DWORD cRef;
  DWORD cErrorOutOfPaper;
  DWORD cErrorNotReady;
  DWORD cJobError;
  DWORD dwNumberOfProcessors;
  DWORD dwProcessorType;
  DWORD dwHighPartTotalBytes;
  DWORD cChangeID;
  DWORD dwLastError;
  DWORD Status;
  DWORD cEnumerateNetworkPrinters;
  DWORD cAddNetPrinters;
```

```
  unsigned short wProcessorArchitecture;
  unsigned short wProcessorLevel;
  DWORD cRefIC;
  DWORD dwReserved2;
  DWORD dwReserved3;
} PRINTER_INFO_STRESS;
```

cJobs: The number of jobs that are currently in the print queue.

cTotalJobs: The total number of jobs that have been spooled since the print server was started.

cTotalBytes: The low-order 32 bits of an unsigned 64-bit value that specifies the total number of bytes that have been printed since system startup. The high-order 32 bits are specified by the **dwHighPartTotalBytes** member.

stUpTime: The time the printer data structure was created, in SYSTEMTIME format.

MaxcRef: The historic maximum value of the **cRef** member.

cTotalPagesPrinted: The total number of pages printed.

dwGetVersion: An implementation-specific value that specifies the version of the operating system.<63>

fFreeBuild: An implementation-specific value that MUST be ignored on receipt.<64>

cSpooling: The number of actively spooling jobs.

cMaxSpooling: The historic maximum number of actively spooling jobs.

cRef: The reference count for opened printer objects.

cErrorOutOfPaper: The total number of out-of-paper errors.

cErrorNotReady: The total number of not-ready errors.

cJobError: The total number of job errors.

dwNumberOfProcessors: The number of processors in the computer on which the print server is running.

dwProcessorType: An implementation-specific value that identifies the type of processor in the computer.<65>

dwHighPartTotalBytes: The high-order 32 bits of an unsigned 64-bit value that specifies the total number of bytes that have been printed since system startup. The low-order 32 bits are specified by the **cTotalBytes** member.

cChangeID: A unique number that identifies the last change.

dwLastError: An implementation-specific error code for the last error that occurred with this printer.<66>

Status: The current printer status (section 2.2.3.12).

cEnumerateNetworkPrinters: The number of times the network printers in the "List of Known Printers" have been requested.

cAddNetPrinters: The number of network printers added, per server.

wProcessorArchitecture: An implementation-specific value that identifies the system's processor architecture. This value SHOULD be ignored on receipt.<67>

wProcessorLevel: An implementation-specific value that identifies the system's architecture-dependent processor level. This value SHOULD be ignored on receipt.<68>

cRefIC: The number of open information context handles.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.2 PRINTER_INFO_1

The PRINTER_INFO_1 structure provides information about a printer.

```
typedef struct _PRINTER_INFO_1 {
  DWORD Flags;
  [string] wchar t* pDescription;
  [string] wchar_t* pName;
  [string] wchar_t* pComment;
} PRINTER_INFO_1;
```

Flags: The value of this member MUST be the result of a bitwise OR of zero or more of the Printer Enumeration Flags (section 2.2.3.7).

If the PRINTER_INFO_1 structure is used in a PRINTER_CONTAINER (section 2.2.1.2.9) as input to RpcAddPrinter (section 3.1.4.2.3) or RpcAddPrinterEx (section 3.1.4.2.15), **Flags** MUST be a bitwise OR of zero or more of the PRINTER_ATTRIBUTE values defined in Status and Attribute Values (section 2.2.3.12).

pName: This member is synonymous with **pPrinterName**, as specified in section 3.1.4.1.5.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.3 PRINTER_INFO_2

The PRINTER_INFO_2 structure provides information about a printer.

```
typedef struct _PRINTER_INFO_2 {
  [string] wchar t* pServerName;
  [string] wchar_t* pPrinterName;
  [string] wchar_t* pShareName;
  [string] wchar_t* pPortName;
  [string] wchar_t* pDriverName;
  [string] wchar t* pComment;
  [string] wchar_t* pLocation;
  ULONG PTR pDevMode;
  [string] wchar_t* pSepFile;
  [string] wchar_t* pPrintProcessor;
  [string] wchar_t* pDatatype;
  [string] wchar t* pParameters;
  ULONG PTR pSecurityDescriptor;
  DWORD Attributes;
  DWORD Priority;
  DWORD DefaultPriority;
  DWORD StartTime;
  DWORD UntilTime;
  DWORD Status;
  DWORD cJobs;
  DWORD AveragePPM;
} PRINTER_INFO_2;
```

pShareName: This member is an optional pointer to a string that specifies the share name for the printer. This string MUST be ignored unless the **Attributes** member contains the **PRINTER_ATTRIBUTED_SHARED** flag. For rules governing path names, see section 2.2.4.9.

pDriverName: This member is a pointer to a string that specifies the name of the printer driver. For rules governing printer driver names, see section 2.2.4.3.

pLocation: This member is an optional pointer to a string that specifies the location of the printer.

pSepFile: This member is an optional pointer to a string that specifies the name of a file whose contents are used to create a separator page. This page is used to separate print jobs sent to the printer. For rules governing path names, see section 2.2.4.9.

pPrintProcessor: This member is an optional pointer to a string that specifies the name of the print processor used by the printer. For rules governing print processor names, see section 2.2.4.11.

If this member is NULL on input, the server SHOULD use the print processor that is associated with the printer driver identified by the string pointed to by the **pDriverName** member.

pDatatype: This member is an optional pointer to a string that specifies the default data format used to record print jobs on the printer. For rules governing data type names, see section 2.2.4.2.

If this member is NULL on input, the server MUST choose a default data type from one of the data types supported by the print processor associated with the printer.<69>

pParameters: This member is an optional pointer to a string that specifies the default print processor parameters.

Priority: The value of this member specifies a priority value that the spooler uses to route each print job. The value of this member MUST be from 0 through 99, inclusive.

DefaultPriority: The value of this member specifies the default priority value assigned to each print job. The value of this member MUST be from 0 through 99, inclusive.

StartTime: The value of this member specifies the earliest time that a job can be printed. The time is expressed as the number of minutes after 12:00 AM GMT within a 24-hour boundary.

UntilTime: The value of this member specifies the latest time that a job can be printed. The time is expressed as the number of minutes after 12:00 AM GMT within a 24-hour boundary.

cJobs: The value of this member specifies the number of print jobs that have been queued for the printer.

AveragePPM: The value of this member specifies the average pages per minute that have been printed on the printer.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.4 PRINTER_INFO_3

The PRINTER_INFO_3 structure provides information about a printer.

```
typedef struct _PRINTER_INFO_3 {
  ULONG_PTR pSecurityDescriptor;
} PRINTER_INFO_3;
```

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.5 PRINTER_INFO_4

The PRINTER_INFO_4 structure provides information about a printer.<70>

```
typedef struct _PRINTER_INFO_4 {
  [string] wchar_t* pPrinterName;
  [string] wchar t* pServerName;
  DWORD Attributes;
} PRINTER_INFO_4;
```

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.6 PRINTER_INFO_5

The PRINTER_INFO_5 structure provides information about a printer.<71>

```
typedef struct _PRINTER_INFO_5 {
  [string] wchar_t* pPrinterName;
  [string] wchar_t* pPortName;
  DWORD Attributes;
  DWORD DeviceNotSelectedTimeout;
  DWORD TransmissionRetryTimeout;
} PRINTER_INFO_5;
```

DeviceNotSelectedTimeout: The value of this member MUST specify the maximum number of milliseconds between select attempts. The **DeviceNotSelectedTimeout** value controls communication between the print server and a print device. It does not have any effect on communication between the print client and the print server.

TransmissionRetryTimeout: The value of this member MUST specify the maximum number of milliseconds between retransmission attempts. The **TransmissionRetryTimeout** value controls communication between the print server and a print device. It does not have any effect on communication between the print client and the print server.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.7 PRINTER_INFO_6

The PRINTER_INFO_6 structure provides information about a printer.<72>

```
typedef struct _PRINTER_INFO_6 {
  DWORD dwStatus;
} PRINTER_INFO_6;
```

dwStatus: The value of this member MUST specify the printer status. It MUST be the result of a bitwise OR of zero or more of the printer status values defined in section 2.2.3.12.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.8 PRINTER_INFO_7

The PRINTER_INFO_7 structure provides directory service (DS) information about a printer.<73>

```
typedef struct  PRINTER INFO 7 {
  [string] wchar_t* pszObjectGUID;
  DWORD dwAction;
} PRINTER_INFO_7;
```

pszObjectGUID: A pointer to a string that represents the GUID that is used by the DS to identify this printer, if it is used in a response to RpcGetPrinter (section 3.1.4.2.6).

The GUID string MUST conform to the UUID grammar, as specified in [RFC4122] section 3. The string representation of a 128-bit GUID is a **GUIDString**.

This member SHOULD be set to NULL when sent and MUST be ignored on receipt by the server, if it is used by the client in a call to RpcSetPrinter (section 3.1.4.2.5).

dwAction: An action for the printer to perform, if it used by the client in a call to RpcSetPrinter.

The value of this member MUST represent a DS-specific publishing state by the server if it is used in a response to RpcGetPrinter.

The value of this member MUST be a constant from the following table:

Name	Meaning
DSPRINT_PUBLISH 0x00000001	RpcSetPrinter: The server MUST publish the printer's data in the DS as described in section 2.3.3.1. RpcGetPrinter: The server MUST set this value to indicate the printer is published in the DS.
DSPRINT_UPDATE 0x00000002	RpcSetPrinter: The server MUST update the printer's published data in the DS as described in section 2.3.3.2. RpcGetPrinter: This value MUST NOT be returned by the server.
DSPRINT_UNPUBLISH 0x00000004	RpcSetPrinter: The server MUST remove the printer's published data from the DS as described in section 2.3.3.2. RpcGetPrinter: The server MUST set this value to indicate the printer is not published.
DSPRINT_REPUBLISH 0x00000008	RpcSetPrinter: The server MUST unpublish (as described in section 2.3.3.2) and publish again (as described in section 2.3.3.1) the DS data for the printer. Republishing also MUST change the GUID of the published printer. RpcGetPrinter: The server MUST NOT set this value.
DSPRINT_PENDING 0x80000000	RpcSetPrinter: This value MUST NOT be used by the client. RpcGetPrinter: The server MUST return this value, if a previous publish or unpublish action initiated by RpcSetPrinter is still in progress.

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.9 PRINTER_INFO_8

The PRINTER_INFO_8 structure provides information about a printer.<74>

This structure is used for the global default settings of a printer.

```
typedef struct _PRINTER_INFO_8 {
  ULONG_PTR pDevMode;
} PRINTER_INFO_8;
```

All members not defined in this section are specified in sections 2.2.1.3.6 and 2.2.1.3.

2.2.1.10.10 PRINTER_INFO_9

The PRINTER_INFO_9 structure is not used remotely.<75>

```
typedef struct _PRINTER_INFO_9 {
  ULONG_PTR pDevMode;
} PRINTER_INFO_9;
```

2.2.1.11 SPLCLIENT_INFO

2.2.1.11.1 SPLCLIENT_INFO_1

The SPLCLIENT_INFO_1 structure provides information about the calling client of the print server.<76>

```
typedef struct _SPLCLIENT_INFO_1 {
  DWORD dwSize;
  [string] wchar_t* pMachineName;
  [string] wchar_t* pUserName;
  DWORD dwBuildNum;
  DWORD dwMajorVersion;
  DWORD dwMinorVersion;
  unsigned short wProcessorArchitecture;
} SPLCLIENT_INFO_1;
```

dwSize: The value of this member MUST specify the size, in bytes, of the structure.

All members not defined in this section are specified in sections 2.2.1.3.7 and 2.2.1.3.

2.2.1.11.2 SPLCLIENT_INFO_2

The SPLCLIENT_INFO_2 structure is specified only as a placeholder in the IDL. It is not sent over the wire.

```
typedef struct  SPLCLIENT_INFO_2 {
  LONG_PTR notUsed;
} SPLCLIENT_INFO_2;
```

notUsed: A value that MUST be ignored.

2.2.1.11.3 SPLCLIENT_INFO_3

The SPLCLIENT_INFO_3 structure provides information about the calling client of the print server.<77>

```
typedef struct _SPLCLIENT_INFO_3 {
  unsigned int cbSize;
  DWORD dwFlags;
  DWORD dwSize;
  [string] wchar_t* pMachineName;
  [string] wchar_t* pUserName;
  DWORD dwBuildNum;
  DWORD dwMajorVersion;
  DWORD dwMinorVersion;
  unsigned short wProcessorArchitecture;
  unsigned __int64 hSplPrinter;
} SPLCLIENT_INFO_3;
```

cbSize: The value of this member MUST specify the size, in bytes, of the structure.

dwFlags: This member is reserved for future use. The value of this member SHOULD be set to zero when sent and MUST be ignored on receipt.

dwSize: This member is reserved for future use. The value of this member SHOULD be set to zero when sent and MUST be ignored on receipt.

hSplPrinter: This member MUST NOT be used remotely and the value of this member SHOULD be set to zero for calls that are made remotely.

All members not defined in this section are specified in sections 2.2.1.3.7 and 2.2.1.3.

2.2.1.12 Bidirectional Communication Data

2.2.1.12.1 RPC_BIDI_REQUEST_DATA

The RPC_BIDI_REQUEST_DATA structure holds a single bidirectional request.<78> The request is part of a bidirectional communication request using the RpcSendRecvBidiData (section 3.1.4.2.27) method. One or more RPC_BIDI_REQUEST_DATA structures MUST be contained in a RPC_BIDI_REQUEST_CONTAINER (section 2.2.1.2.10).

```
typedef struct _RPC_BIDI_REQUEST_DATA {
  DWORD dwReqNumber;
  [string, unique] wchar_t* pSchema;
  RPC_BIDI_DATA data;
} RPC_BIDI_REQUEST_DATA;
```

dwReqNumber: The index of the request, which is used to match a response to a request in a multi-request operation.

pSchema: A pointer to the schema string that identifies the requested information. <79>

data: The data that is associated with the schema.

2.2.1.12.2 RPC_BIDI_RESPONSE_DATA

The RPC_BIDI_RESPONSE_DATA structure holds a single bidirectional response.<80>

```
typedef struct _RPC_BIDI_RESPONSE_DATA {
  DWORD dwResult;
  DWORD dwReqNumber;
  [string, unique] wchar_t* pSchema;
  RPC_BIDI_DATA data;
} RPC_BIDI_RESPONSE_DATA;
```

dwResult: The result of the operation that used this structure. If the operation was successful, the value of this member MUST be set to zero; otherwise, the value of this member MUST be set to a nonzero value.<81>

dwReqNumber: The index of the response, which is used to match the response to the request in a multi-request operation.

pSchema: A pointer to the schema string that identifies the requested information.<82>

data: The data that is associated with the schema. This can be a single piece of data or a homogeneous data list. The data MUST be composed of a name, a type, and a value; for example, "\Printer.Stapler:CurrentValue". It is referenced by its name under **Properties**.

2.2.1.12.3 RPC_BIDI_DATA

The RPC_BIDI_DATA structure is used to store the values of a bidirectional schema.<83>

```
typedef struct _RPC_BIDI_DATA {
  DWORD dwBidiType;
  [switch is(dwBidiType)] union {
    [case(BIDI_NULL,BIDI_BOOL)]
      int bData;
    [case(BIDI_INT)]
      long iData;
    [case(BIDI_STRING,BIDI_TEXT,BIDI_ENUM)]
      [unique, string] wchar t* sData;
    [case(BIDI_FLOAT)]
      float fData;
    [case(BIDI_BLOB)]
      RPC_BINARY_CONTAINER biData;
  } u;
} RPC_BIDI_DATA;
```

dwBidiType: The type of data in a bidirectional request. The value of this member specifies a valid structure for the **u** union. The value of this member MUST be one of the **BIDI_TYPE** enumeration values specified in section 2.2.3.13.

u: The bidirectional data in the format specified by the value of the **dwBidiType** member.

bData: This case indicates that either there is no bidirectional data, or the bidirectional data is a Boolean value.

iData: This case indicates that the bidirectional data is an integer.

sData: This case indicates that the bidirectional data is either a string, text data, or an enumeration.

fData: The bidirectional data is a floating-point number.

biData: This case indicates that the bidirectional data is an RPC_BINARY_CONTAINER structure.

2.2.1.13 Printer Notification Data

2.2.1.13.1 RPC_V2_NOTIFY_OPTIONS

The RPC_V2_NOTIFY_OPTIONS structure specifies options for a change notification object that monitors a printer or print server for any changes in state.<84>

```
typedef struct _RPC_V2_NOTIFY_OPTIONS {
  DWORD Version;
  DWORD Reserved;
  DWORD Count;
  [size_is(Count), unique] RPC_V2_NOTIFY_OPTIONS_TYPE* pTypes;
} RPC_V2_NOTIFY_OPTIONS;
```

Version: The version of the structure. The value of this member MUST be 0x00000002.

Reserved: A bit field that specifies attributes of the change notification. The name "Reserved" is intentional.

Value	Meaning
PRINTER_NOTIFY_OPTIONS_REFRESH 0x00000001	Refreshed data is requested from the server for all monitored members.

Count: The number of RPC_V2_NOTIFY_OPTIONS_TYPE structures (section 2.2.1.13.2) in the array pointed to by the **pTypes** member.

pTypes: A pointer to an array of RPC_V2_NOTIFY_OPTIONS_TYPE structures, each of which identifies a set of print job or printer information members to be monitored by a printer change notification object.

2.2.1.13.2 RPC_V2_NOTIFY_OPTIONS_TYPE

The RPC_V2_NOTIFY_OPTIONS_TYPE structure MUST specify the set of printer or job information members to be monitored by a printer change notification object.<85>

```
typedef struct _RPC_V2_NOTIFY_OPTIONS_TYPE {
  unsigned short Type;
  unsigned short Reserved0;
  DWORD Reserved1;
  DWORD Reserved2;
  DWORD Count;
  [size_is(Count), unique] unsigned short* pFields;
} RPC_V2_NOTIFY_OPTIONS_TYPE;
```

Type: The value of this member specifies the type of notification to watch for. The value of this member MUST be one of the constant values from the following table.

Name/Value	Meaning
PRINTER_NOTIFY_TYPE 0x0000	Indicates that the members specified in the array that is pointed to by the **pFields** member are printer notification constants.
JOB_NOTIFY_TYPE 0x0001	Indicates that the members specified in the array that is pointed to by the **pFields** member are job notification constants.
SERVER_NOTIFY_TYPE 0x0002	Indicates that the members specified in the array that is pointed to by the **pFields** member are server notification constants.<86>

Reserved0: The value of this member MUST be set to zero when sent and MUST be ignored on receipt.

Reserved1: The value of this member MUST be set to zero when sent and MUST be ignored on receipt.

Reserved2: The value of this member MUST be set to zero when sent and MUST be ignored on receipt.

Count: The value of this member MUST specify the number of elements in the **pFields** array.

pFields: This member MUST be a non-NULL pointer to an array that MUST identify the job or printer information members to be monitored. The array MUST consist entirely of elements that are either job notification values (as specified in section 2.2.3.3) or printer notification values (as specified in section 2.2.3.8), depending on the value of the **Type** member. The two types of notification values MUST NOT be mixed within a given instance of the array.

2.2.1.13.3 RPC_V2_NOTIFY_INFO

The RPC_V2_NOTIFY_INFO structure specifies printer or print job notification information.<87>

```
typedef struct _RPC_V2_NOTIFY_INFO {
  DWORD Version;
  DWORD Flags;
```

```
  DWORD Count;
  [size_is(Count), unique] RPC_V2_NOTIFY_INFO_DATA aData[];
} RPC_V2_NOTIFY_INFO;
```

Version: The version of the structure. The value of this member MUST be 0x00000002.

Flags: A bit field that specifies the state of the notification structure.

Value	Meaning
PRINTER_NOTIFY_INFO_DISCARDED 0x00000001	An overflow or error has occurred, and notifications have been lost. The print server MUST NOT send additional notifications until the client has made a call to RpcRouterRefreshPrinterChangeNotification (section 3.1.4.10.5).

Count: The number of RPC_V2_NOTIFY_INFO_DATA structures (section 2.2.1.13.4) in the **aData** array.

aData: An array of RPC_V2_NOTIFY_INFO_DATA structures, each of which identifies a single print job or printer information member and specifies the current data for that member.

2.2.1.13.4 RPC_V2_NOTIFY_INFO_DATA

The RPC_V2_NOTIFY_INFO_DATA structure specifies printer or print job notification information data.<88>

```
typedef struct _RPC_V2_NOTIFY_INFO_DATA {
  unsigned short Type;
  unsigned short Field;
  DWORD Reserved;
  DWORD Id;
  [switch_is(Reserved & 0x0000FFFF)]
    RPC_V2_NOTIFY_INFO_DATA_DATA Data;
} RPC_V2_NOTIFY_INFO_DATA;
```

Type: The type of notification information that is contained in this structure. This MUST be one of the following values:

Name/value	Meaning
PRINTER_NOTIFY_TYPE 0x0000	Printer-related notifications
JOB_NOTIFY_TYPE 0x0001	Job-related notifications

Field: The member that changed using the printer notification values and job notification values in sections 2.2.3.8 and 2.2.3.3.

Reserved: The member of the RPC_V2_NOTIFY_INFO_DATA_DATA union (section 2.2.1.13.5) that is used to specify the data type of the **Data** member. Only the 16 least-significant bits of this member are used. The value of these bits MUST be one of the Notification Data Type values (section 2.2.3.5).

Id: The job identifier if the **Type** member specifies **JOB_NOTIFY_TYPE**; otherwise, this value MUST be ignored.

Data: The data determined by the values of the **Type** and **Reserved** members of this structure. The data is in an **RPC_V2_NOTIFY_INFO_DATA_DATA** structure using the data type described by the value of the **Reserved** member.

2.2.1.13.5 RPC_V2_NOTIFY_INFO_DATA_DATA

The RPC_V2_NOTIFY_INFO_DATA_DATA union specifies the data information container for the current notification.<89> The case attributes of this union are specified by Notification Data Type values (section 2.2.3.5).

```
typedef
[switch_type(DWORD)]
union  RPC_V2_NOTIFY_INFO_DATA_DATA {
  [case(TABLE_STRING)]
    STRING_CONTAINER String;
  [case(TABLE_DWORD)]
    DWORD dwData[2];
  [case(TABLE_TIME)]
    SYSTEMTIME_CONTAINER SystemTime;
  [case(TABLE_DEVMODE)]
    DEVMODE_CONTAINER DevMode;
  [case(TABLE_SECURITYDESCRIPTOR)]
    SECURITY_CONTAINER SecurityDescriptor;
} RPC_V2_NOTIFY_INFO_DATA_DATA;
```

String: Case **TABLE_STRING**: This member specifies a STRING_CONTAINER structure (section 2.2.1.2.15).

dwData: Case **TABLE_DWORD**: This member specifies an array of two **DWORD** values that contain the member's current data.

SystemTime: Case **TABLE_TIME** This member specifies a SYSTEMTIME_CONTAINER structure (section 2.2.1.2.16).

DevMode: Case **TABLE_DEVMODE**: This member specifies a DEVMODE_CONTAINER structure (section 2.2.1.2.1) that defines default printer attributes such as the paper orientation and printing resolution.

SecurityDescriptor: Case **TABLE_SECURITYDESCRIPTOR**: This member specifies a SECURITY_CONTAINER structure (section 2.2.1.2.13), in which the **pSecurity** member is a pointer to a SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6) in self-relative form.

2.2.1.13.6 RPC_V2_UREPLY_PRINTER

The RPC_V2_UREPLY_PRINTER union defines printer notification responses.<90>

```
typedef
[switch type(DWORD)]
union _RPC_V2_UREPLY_PRINTER {
  [case(0x00000000)]
    RPC_V2_NOTIFY_INFO* pInfo;
} RPC_V2_UREPLY_PRINTER;
```

pInfo: This member MUST be a non-NULL pointer to an RPC_V2_NOTIFY_INFO structure, which MUST contain notification information.

2.2.1.14 Job Named Properties

2.2.1.14.1 RPC_PrintPropertyValue

The **RPC_PrintPropertyValue** structure specifies the value of a Job Named Property (section 3.1.1).<91>

```
typedef struct RPC_EPrintPropertyType {
   ePropertyType;
  [switch_is(ePropertyType)] union {
    [case(kRpcPropertyTypeString), string]
      wchar_t* propertyString;
    [case(kRpcPropertyTypeInt32)] LONG propertyInt32;
    [case(kRpcPropertyTypeInt64)] LONGLONG propertyInt64;
    [case(kRpcPropertyTypeByte)] BYTE propertyByte;
    [case(kRpcPropertyTypeBuffer] struct {
      DWORD cbBuf;
      [size_is(cbBuf)] BYTE* pBuf;
    } propertyBlob;
  } value;
} RPC_PrintPropertyValue;
```

ePropertyType: The type of the value. All enumeration values described in section 2.2.1.14.3 are valid.

propertyString: A pointer to a string containing the property value. Valid only if **ePropertyType** is set to **kRpcPropertyTypeString**.

propertyInt32: The property value as a signed 32-bit integer. Valid only if **ePropertyType** is set to **kRpcPropertyTypeInt32**.

propertyInt64: The property value as a signed 64-bit integer. Valid only if **ePropertyType** is set to **kRpcPropertyTypeInt64**.

propertyByte: The property value as a byte. Valid only if **ePropertyType** is set to kRpcPropertyTypeByte.

propertyBlob: An embedded structure that describes the buffer containing the property value as an array of bytes. Valid only if **ePropertyType** is set to **kRpcPropertyTypeBuffer**.

cbBuf: Member of the **propertyBlob** structure that specifies the length, in bytes, of the property value contained in the **pBuf** buffer when **ePropertyType** is set to **kRpcPropertyTypeBuffer**.

pBuf: Member of the **propertyBlob** structure that contains a pointer to the buffer containing the property value when **ePropertyType** is set to **kRpcPropertyTypeBuffer**.

2.2.1.14.2 RPC_PrintNamedProperty

The **RPC_PrintNamedProperty** structure specifies a Job Named Property (section 3.1.1).<92>

```
typedef struct {
  [string] wchar_t* propertyName;
  RPC_PrintPropertyValue propertyValue;
} RPC_PrintNamedProperty;
```

propertyName: A pointer to a string containing the name of the property.

propertyValue: An RPC_PrintPropertyValue structure (section 2.2.1.14.1) containing the value of the property.

2.2.1.14.3 RPC_EPrintPropertyType

The **RPC_EPrintPropertyType** enumeration specifies the type of the value contained by a Job Named Property (section 3.1.1).<93>

```
typedef  enum
{
  kRpcPropertyTypeString = 1,
  kRpcPropertyTypeInt32,
  kRpcPropertyTypeInt64,
  kRpcPropertyTypeByte,
  kRpcPropertyTypeBuffer
} RPC_EPrintPropertyType;
```

kRpcPropertyTypeString: The property value is a string.

kRpcPropertyTypeInt32: The property value is a signed 32-bit integer.

kRpcPropertyTypeInt64: The property value is a signed 64-bit integer.

kRpcPropertyTypeByte: The property value is a byte.

kRpcPropertyTypeBuffer: The property value consists of an array of bytes contained in a buffer.

2.2.1.14.4 SPLFILE_CONTENT_TYPE_PROP_NAME

The SPL_CONTENT_TYPE_PROP_NAME constant defines the name of a standard **Job Named Property** (section 3.1.1) that specifies the spool file format for the print data that is sent by the client.<94>

```
#define SPLFILE CONTENT TYPE PROP NAME L"Spool File Contents"
```

The following are the valid string values for this property.

Property Value	Meaning
L"TYPE_XPS_MS"	Microsoft XML Paper Specification (XPS) format. For more information, see [MSFT-XMLPAPER].
L"TYPE_XPS_OPEN"	**OpenXPS** format. For more information, see [ECMA-388].
L"TYPE_PDL_POSTSCRIPT"	Page description language (PDL) **PostScript** format.
L"SPLFILE_CONTENT_TYPE_PDL_UNKNOWN"	Unknown PDL format.

Print servers are not required to support this property. Print clients are not required to support or set this property. If a print client does not set this property on a print job that the client submits to a print server, but the print server supports this property, the print server SHOULD by default select the value L"TYPE_XPS_MS".

2.2.1.15 Branch Office Print Remote Logging Structures

2.2.1.15.1 EBranchOfficeJobEventType

The EBranchOfficeJobEventType enumeration specifies the type of **Windows Event** contained by a **Branch Office Print Remote Log Entry** (section 3.1.1).<95>

```
typedef  enum
{
  kInvalidJobState = 0,
  kLogJobPrinted,
  kLogJobRendered,
  kLogJobError,
  kLogJobPipelineError,
  kLogOfflineFileFull
} EBranchOfficeJobEventType;
```

kInvalidJobState: The Windows Event is an unknown type.

kLogJobPrinted: The **Branch Office Print Remote Log Entry** contains a Windows Event that corresponds to **event ID** 307.

kLogJobRendered: The Branch Office Print Remote Log Entry contains a Windows Event that corresponds to event ID 805.

kLogJobError: The Branch Office Print Remote Log Entry contains a Windows Event that corresponds to event ID 372.

kLogJobPipelineError: The Branch Office Print Remote Log Entry contains a Windows Event that corresponds to event ID 824.

kLogOfflineFileFull: The Branch Office Print Remote Log Entry contains a Windows Event that corresponds to event ID 868.

2.2.1.15.2 RPC_BranchOfficeJobData

The RPC_BranchOfficeJobData structure holds a **branch office print remote logging** structure that contains the data required to log a single **Branch Office Print Remote Log Entry** (section 3.1.1) corresponding to a specific type of Windows Event.<96>

```
typedef struct {
  EBranchOfficeJobEventType eEventType;
  DWORD JobId;
  [switch type(EBranchOfficeJobEventType), switch is(eEventType)]
    union {
    [case(kLogJobPrinted)]
      RPC_BranchOfficeJobDataPrinted LogJobPrinted;
    [case(kLogJobRendered)]
      RPC_BranchOfficeJobDataRendered LogJobRendered;
    [case(kLogJobError)]
      RPC_BranchOfficeJobDataError LogJobError;
    [case(kLogJobPipelineError)]
      RPC_BranchOfficeJobDataPipelineFailed LogJobPipelineFailed;
    [case(kLogOfflineFileFull)]
      RPC_BranchOfficeLogOfflineFileFull LogOfflineFileFull;
  } JobInfo;
} RPC_BranchOfficeJobData;
```

eEventType: The type of Windows Event to which the **Branch Office Print Remote Log Entry** corresponds, which MUST be an EBranchOfficeJobEventType (section 2.2.1.15.1) value.

JobId: The identifier of a print job.

JobInfo: The branch office print remote logging structure that contains the data required to log a **Branch Office Print Remote Log Entry** corresponding to the **eEventType** member value.

LogJobPrinted: An RPC_BranchOfficeJobDataPrinted (section 2.2.1.15.5) structure for a **Branch Office Print Remote Log Entry** corresponding to event ID 307. This structure is present only if **eEventType** is set to **kLogJobPrinted**.

LogJobRendered: An RPC_BranchOfficeJobDataRendered (section 2.2.1.15.6) structure for a **Branch Office Print Remote Log Entry** corresponding to event ID 805. This structure is present only if **eEventType** is set to **kLogJobRendered**.

LogJobError: An RPC_BranchOfficeJobDataError (section 2.2.1.15.3) structure for a **Branch Office Print Remote Log Entry** corresponding to event ID 372. This structure is present only if **eEventType** is set to **kLogJobError**.

LogJobPipelineFailed: An RPC_BranchOfficeJobDataPipelineFailed (section 2.2.1.15.4) structure for a **Branch Office Print Remote Log Entry** corresponding to event ID 824. This structure is present only if **eEventType** is set to **kLogJobPipelineError**.

LogOfflineFileFull: An RPC_BranchOfficeLogOfflineFileFull (section 2.2.1.15.7) structure for a **Branch Office Print Remote Log Entry** corresponding to event ID 868. This structure is present only if **eEventType** is set to **kLogOfflineFileFull**.

2.2.1.15.3 RPC_BranchOfficeJobDataError

The RPC_BranchOfficeJobDataError structure holds a single **Branch Office Remote Logging Entry** (section 3.1.1).<97> This entry contains the information needed to create event ID 372 in the **Microsoft-Windows-PrintService**/Admin **event channel**.

```
typedef struct {
  DWORD LastError;
  [string] wchar_t* pDocumentName;
  [string] wchar_t* pUserName;
  [string] wchar_t* pPrinterName;
  [string] wchar_t* pDataType;
  LONGLONG TotalSize;
  LONGLONG PrintedSize;
  DWORD TotalPages;
  DWORD PrintedPages;
  [string] wchar_t* pMachineName;
  [string] wchar_t* pJobError;
  [string] wchar_t* pErrorDescription;
} RPC_BranchOfficeJobDataError;
```

LastError: A 32-bit unsigned integer that specifies an implementation-specific error code for the last error that occurred during processing of this print job.

pDocumentName: A pointer to a string that specifies the name of the print document for this print job.

pUserName: A pointer to a string that specifies the name of the user that owns the print job. For rules governing user names, see section 2.2.4.17.

pPrinterName: A pointer to a string that specifies the name of the printer used for the print job. For rules governing printer names, see section 2.2.4.14.

pDataType: A pointer to a string that specifies the type of data that the printing application sent to the printer in the print job. The identified data type MUST be supported by the print processor that is associated with the printer that is processing the job. For rules governing data type names, see section 2.2.4.2.

TotalSize: A 64-bit signed integer that specifies the size of the print job, in bytes. This value MUST be greater than zero.

PrintedSize: A 64-bit signed integer that specifies the amount of data for the print job that actually got processed and sent to the printer, in bytes. This value MUST be zero or greater.

TotalPages: A 32-bit unsigned integer that specifies the number of pages the document contains.

PrintedPages: A 32-bit unsigned integer that specifies the number of pages of the document that actually got processed and sent to the printer.

pMachineName: A pointer to a string that specifies the name of the client computer that owns the print job. For rules governing computer names, see section 2.2.4.16.

pJobError: A pointer to a string that specifies the text representation of the value of the **LastError** error code.

pErrorDescription: A pointer to an optional string that specifies message text for a system-defined error corresponding to the value of the **LastError** error code.

2.2.1.15.4 RPC_BranchOfficeJobDataPipelineFailed

The RPC_BranchOfficeJobDataPipelineFailed structure holds a single **Branch Office Print Remote Log Entry** (section 3.1.1).<98> This entry contains the information needed to create event ID 824 in the Microsoft-Windows-PrintService/Operational event channel.

```
typedef struct {
  [string] wchar_t* pDocumentName;
  [string] wchar_t* pPrinterName;
  [string] wchar_t* pExtraErrorInfo;
} RPC_BranchOfficeJobDataPipelineFailed;
```

pDocumentName: A pointer to a string that specifies the name of the print document for this print job.

pPrinterName: A pointer to a string that specifies the name of the printer used for the print job. For rules governing printer names, see section 2.2.4.14.

pExtraErrorInfo: A pointer to an optional string that specifies additional text associated with the failure in the **Print Pipeline**.

2.2.1.15.5 RPC_BranchOfficeJobDataPrinted

The RPC_BranchOfficeJobDataPrinted structure holds a single **Branch Office Print Remote Log Entry** (section 3.1.1).<99> This entry contains the information needed to create event ID 307 in the Microsoft-Windows-PrintService/Operational event channel.

```
typedef struct {
  DWORD Status;
  [string] wchar_t* pDocumentName;
  [string] wchar_t* pUserName;
  [string] wchar_t* pMachineName;
  [string] wchar t* pPrinterName;
  [string] wchar_t* pPortName;
  LONGLONG Size;
  DWORD TotalPages;
} RPC_BranchOfficeJobDataPrinted;
```

Status: A 32-bit unsigned integer that specifies an implementation-specific error code for the last error that occurred during the processing of this print job.

pDocumentName: A string that specifies the name of the print document for this print job.

pUserName: A pointer to a string that specifies the name of the user that owns the print job. For rules governing user names, see section 2.2.4.17.

pMachineName: A pointer to a string that specifies the name of the client computer that owns the print job. For rules governing computer names, see section 2.2.4.16.

pPrinterName: A pointer to a string that specifies the name of the printer used for the print job. For rules governing printer names, see section 2.2.4.14.

pPortName: A pointer to a string that specifies a supported printer port. For rules governing port names, see section 2.2.4.10.

Size: A 64-bit signed integer that specifies the size of the print job, in bytes.

TotalPages: A 32-bit unsigned integer that specifies the number of pages the document contains.

2.2.1.15.6 RPC_BranchOfficeJobDataRendered

The RPC_BranchOfficeJobDataRendered structure holds a single **Branch Office Print Remote Log Entry** (section 3.1.1).<100> This entry contains the information needed to create event ID 805 in the Microsoft-Windows-PrintService/Operational event channel.

```
typedef struct {
  LONGLONG Size;
  DWORD ICMMethod;
  short Color;
  short PrintQuality;
  short YResolution;
  short Copies;
  short TTOption;
} RPC_BranchOfficeJobDataRendered;
```

Size: A 64-bit signed integer that specifies the size of the print job, in bytes.

ICMMethod: A 32-bit unsigned integer that specifies how **Image Color Management (ICM)** is handled for the print job. See the definition of the **dmICMMethod** field in section 2.2.2.1.

Color: A 16-bit signed integer that specifies the color mode to use for the print job. See the definition of the **dmColor** field in section 2.2.2.1.

PrintQuality: A 16-bit signed integer that specifies the printer resolution for the print job. See the definition of the **dmPrintQuality** field in section 2.2.2.1.

YResolution: A 16-bit signed integer that specifies the vertical-resolution in dots per inch for the print job. See the definition of the **dmYResolution** field in section 2.2.2.1.

Copies: A 16-bit signed integer that specifies the number of copies to be printed for the print job. See the definition of the **dmCopies** field in section 2.2.2.1.

TTOption: A 16-bit signed integer that specifies how TrueType fonts are printed for the print job. See the definition of the **dmTTOption** field in section 2.2.2.1.

2.2.1.15.7 RPC_BranchOfficeLogOfflineFileFull

The RPC_BranchOfficeLogOfflineFileFull structure holds a single **Branch Office Print Remote Log Entry** (section 3.1.1)<101> that indicates the **Branch Office Print Remote Log Offline Archive** (section 3.2.1) has exceeded the maximum allowed size.<102> This entry contains the information needed to create event ID 868 in the Microsoft-Windows-PrintService/Admin event channel.

```
typedef struct {
```

```
    [string] wchar_t* pMachineName;
} RPC_BranchOfficeLogOfflineFileFull;
```

pMachineName: A pointer to a string that specifies the name of the client computer that encountered the **Branch Office Print Remote Log Offline Archive** full condition. For rules governing computer names, see section 2.2.4.16.

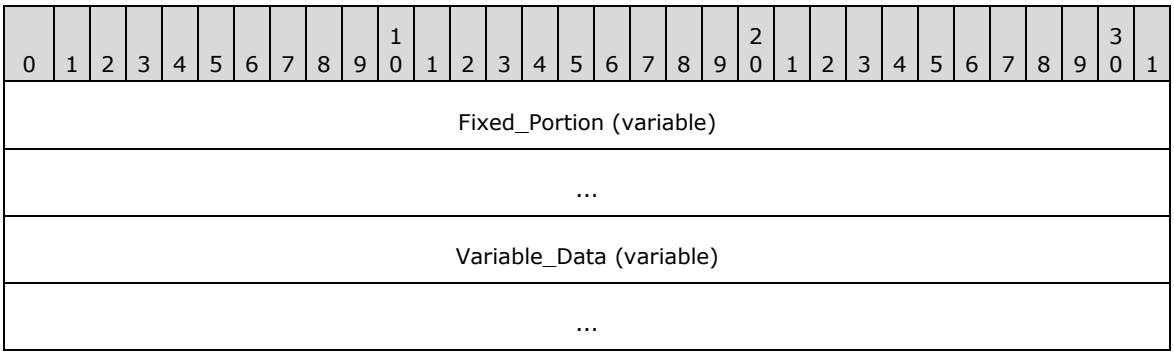
2.2.2 Custom-Marshaled Data Types

This section specifies data structures that are custom-marshaled, including those that contain "_INFO" in their names. All custom-marshaled **INFO** data structures MUST be completely ignored on input, and validation of their contents MUST NOT take place.

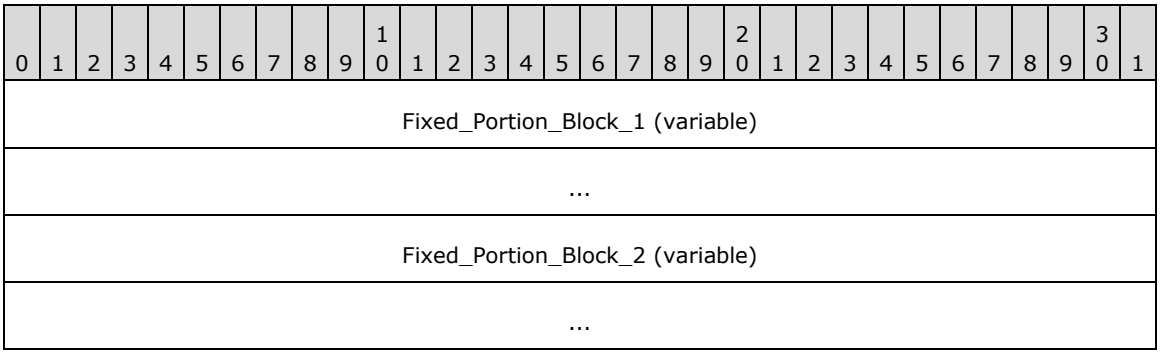
Custom-marshaled **INFO** data structures consist of single **Fixed_Portion** blocks for methods accepting or returning a single structure, and arrays of one or more **Fixed_Portion** blocks for methods accepting or returning an array of structures. The size of the **Fixed_Portion** data is the size of a single **Fixed_Portion** block multiplied by the number of **Fixed_Portion** blocks returned. The **Fixed_Portion** data is followed by a single **Variable_Data** block, which contains variable-length fields. The size of the **Variable_Data** block is the size specified by the caller in the *cbBuf* parameter of the call minus the size of the **Fixed_Portion** data.

For each field in a **Variable_Data** block, a corresponding offset value is specified in a field of a **Fixed_Portion** block. A **Variable_Data** field is located by adding that offset value to the address of the start of the **Fixed_Portion** block in which that offset is defined.

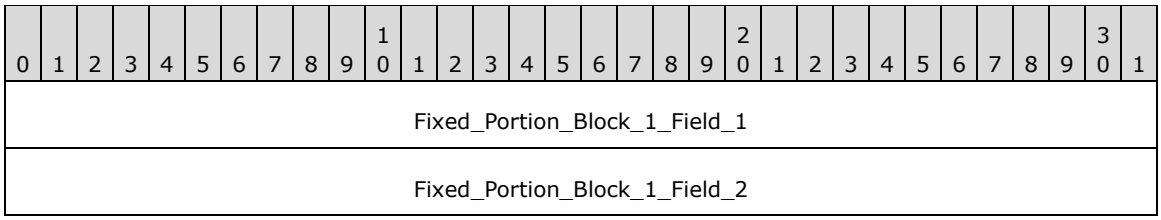
This generic structure of custom-marshaled **INFO** data structures is represented by the following diagram.



Fixed_Portion (variable): An array of one or more **Fixed_Portion** blocks, each consisting of one or more fixed-length fields. The specific structure of the **Fixed_Portion** block is defined for each **INFO** structure.



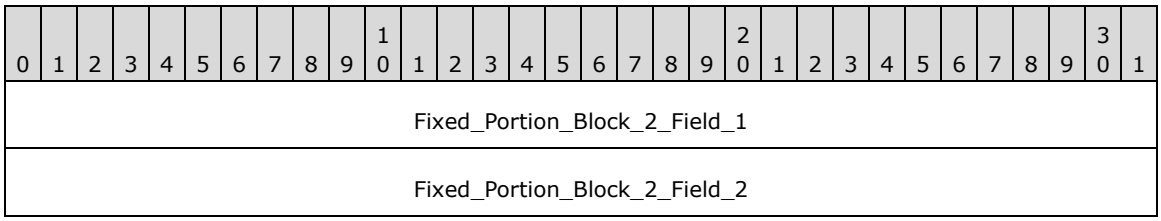
Fixed_Portion_Block_1 (variable): Fixed_Portion block 1.



Fixed_Portion_Block_1_Field_1 (4 bytes): Fixed-length field 1 of **Fixed_Portion** block 1. Although the length of this field is shown as 4 bytes, its actual length is indeterminate in this generic structure.

Fixed_Portion_Block_1_Field_2 (4 bytes): Fixed-length field 2 of **Fixed_Portion** block 1. This field contains an offset to **Variable_Data_Field_1**, which is relative to the start of **Fixed_Portion** block 1.

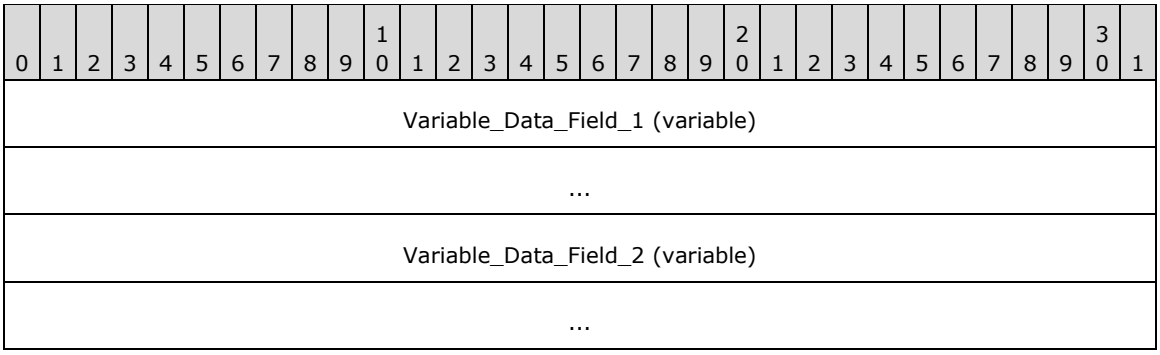
Fixed_Portion_Block_2 (variable): Optional **Fixed_Portion** block 2.



Fixed_Portion_Block_2_Field_1 (4 bytes): Fixed-length field 1 of **Fixed_Portion** block 2. Although the length of this field is shown as 4 bytes, its actual length is indeterminate in this generic structure.

Fixed_Portion_Block_2_Field_2 (4 bytes): Fixed-length field 2 of **Fixed_Portion** block 2. This field contains an offset to **Variable_Data_Field_2**, which is relative to the start of **Fixed_Portion** block 2.

Variable_Data (variable): A data block of variable length. Because the data is not necessarily aligned on 16-bit boundaries, it is specified as an array of bytes of arbitrary length; however, data fields in the **Variable_Data** block MUST be aligned on natural boundaries matching their data type. That is, WCHAR fields MUST be aligned on 2-byte boundaries, DWORD fields MUST be aligned on 4-byte boundaries, and so on.



Variable_Data_Field_1 (variable): Variable-length field 1 of the **Variable_Data** block.

Variable_Data_Field_2 (variable): Variable-length field 2 of the **Variable_Data** block.

The following characteristics apply to the fields in custom-marshaled **INFO** data structures:

- The start of the **Fixed_Portion** block MUST be 32-bit aligned.
- The order of fields in the **Fixed_Portion** block is defined by the specific **INFO** structure layout.
- Data fields in the **Variable_Data** block can appear in arbitrary order.
- One or more offsets in **Fixed_Portion** blocks can locate the same field in the **Variable_Data** block; or there can be a one-to-one correspondence between offsets and **Variable_Data** fields.
- The **Variable_Data** fields SHOULD be packed tightly in the **Variable_Data** block, filling the **Variable_Data** block from the end toward the beginning, such that, if the *cbBuf* parameter specified by the caller is larger than the sum of all **Fixed_Portion** blocks and all **Variable_Data** fields, the unused space in the **[out]** buffer receiving the custom-marshaled **INFO** structure will form a gap between the end of the last **Fixed_Portion** block and the beginning of the first **Variable_Data** field; however, client-side unmarshaling code that processes a custom-marshaled **INFO** structure SHOULD be prepared to correctly handle data that does not fill the **Variable_Data** block from the end toward the beginning, or is not tightly packed and includes unused space in arbitrary positions of the **Variable_Data** block.<103>

2.2.2.1 _DEVMODE

The _DEVMODE structure defines initialization data for a printer. Although the _DEVMODE structure does not contain any pointers, it is still custom-marshaled, because the size of the structure is version-specific and implementation-specific, and cannot be expressed using IDL attributes. It has the following message format.

The print server MUST accept _DEVMODE structures with truncated public information. A truncated _DEVMODE structure contains a subset of fields, from **dmDeviceName** up to and including **dmFields**, plus at least those fields that are initialized as specified by **dmFields**. The truncated _DEVMODE structure is followed by private, printer driver-specific data, the size of which is specified by the value of the **dmDriverExtra** field.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
dmDeviceName (64 bytes)																															
...																															
...																															
dmSpecVersion																dmDriverVersion															
dmSize																dmDriverExtra															
dmFields																															
dmOrientation																dmPaperSize															
dmPaperLength																dmPaperWidth															
dmScale																dmCopies															

dmDefaultSource	dmPrintQuality
dmColor	dmDuplex
dmYResolution	dmTTOption
dmCollate	dmFormName (64 bytes)
...	
...	
...	reserved0
reserved1	
reserved2	
reserved3	
dmNup	
reserved4	
dmICMMethod	
dmICMIntent	
dmMediaType	
dmDitherType	
reserved5	
reserved6	
reserved7	
reserved8	
dmDriverExtraData (variable)	
...	

dmDeviceName (64 bytes): A 32-element array of 16-bit Unicode characters that form a null-terminated string that specifies the name of the printer. Printer name strings that are longer than 32 characters MUST be truncated to fit the array. For more rules governing printer names, see section 2.2.4.14.

dmSpecVersion (2 bytes): The version of initialization data specification on which the _DEVMODE structure is based. This value SHOULD be 0x0401.<104>

dmDriverVersion (2 bytes): For printers, an optional, implementation-defined version of the printer driver.<105>

dmSize (2 bytes): The size, in bytes, of the _DEVMODE structure, which MUST be a multiple of 4 bytes. This value MUST NOT include the length of any private, printer driver–specific data that might follow the _DEVMODE structure's public fields. The size, in bytes, of such private data is specified by the **dmDriverExtra** value.

If the _DEVMODE structure contains truncated public information, the value of **dmSize** is at least the size, in bytes, of a subset of fields, from **dmDeviceName** up to and including **dmFields**, plus fields that are initialized as specified by **dmFields**.

dmDriverExtra (2 bytes): The size, in bytes, of the private printer drivers data that follows this structure.

dmFields (4 bytes): A bitfield that specifies the fields of the _DEVMODE structure that have been initialized. If a bit is set, the corresponding field MUST be initialized and MUST be processed on receipt. If a bit is not set, the value of the corresponding field SHOULD be set to zero when sent and MUST be ignored on receipt.

The value of this field is the result of a bitwise OR of the following bits.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	UP	0	SC	PW	PL	PS	OR	CL	TT	Y	DX	CR	PQ	DS	CP	CM	0	0	0	0	0	0	FM	0	0	0	0	0	DT	MT	CI

Value	Description
CI DM_ICMINTENT	If this bit is set, the **dmICMIntent** field MUST be initialized.
MT DM_MEDIATYPE	If this bit is set, the **dmMediaType** field MUST be initialized.
DT DM_DITHERTYPE	If this bit is set, the **dmDitherType** field MUST be initialized.
FM DM_FORMNAME	If this bit is set, the **dmFormName** field MUST be initialized.
CM DM_ICMMETHOD	If this bit is set, the **dmICMMethod** field MUST be initialized.
CP DM_COPIES	If this bit is set, the **dmCopies** field MUST be initialized.
DS DM_DEFAULTSOURCE	If this bit is set, the **dmDefaultSource** field MUST be initialized.
PQ DM_PRINTQUALITY	If this bit is set, the **dmPrintQuality** field MUST be initialized.

Value	Description
CR DM_COLOR	If this bit is set, the **dmColor** field MUST be initialized.
DX DM_DUPLEX	If this bit is set, the **dmDuplex** field MUST be initialized.
Y DM_YRESOLUTION	If this bit is set, the **dmYResolution** field MUST be initialized.
TT DM_TTOPTION	If this bit is set, the **dmTTOption** field MUST be initialized.
CL DM_COLLATE	If this bit is set, the **dmCollate** field MUST be initialized.
OR DM_ORIENTATION	If this bit is set, the **dmOrientation** field MUST be initialized.
PS DM_PAPERSIZE	If this bit is set, the **dmPaperSize** field MUST be initialized. This bit MUST NOT be set if either DM_PAPERLENGTH or DM_PAPERWIDTH are set.
PL DM_PAPERLENGTH	If this bit is set, the **dmPaperLength** field MUST be initialized. This bit MUST NOT be set if DM_PAPERSIZE is set.
PW DM_PAPERWIDTH	If this bit is set, the **dmPaperWidth** field MUST be initialized. This bit MUST NOT be set if DM_PAPERSIZE is set.
SC DM_SCALE	If this bit is set, the **dmScale** field MUST be initialized.
UP DM_NUP	If this bit is set, the **dmNup** field MUST be initialized.

dmOrientation (2 bytes): For printers, the orientation for output. If the DM_ORIENTATION bit is set in **dmFields**, this value MUST be specified. The value of this field SHOULD be one of the following.

Name/value	Meaning
DMORIENT_POTRAIT 0x0001	"Portrait" orientation.
DMORIENT_LANDSCAPE 0x0002	"Landscape" orientation.

dmPaperSize (2 bytes): For printers, the size of the output media. If the DM_PAPERSIZE bit is set in **dmFields**, this value MUST be specified. The value of this field SHOULD be one of the following, or it MAY be a device-specific value that is greater than or equal to 0x0100.

Name/value	Meaning
DMPAPER_LETTER 0x0001	Letter, 8 1/2 x 11 inches
DMPAPER_LEGAL 0x0005	Legal, 8 1/2 x 14 inches

Name/value	Meaning
DMPAPER_10X14 0x0010	10 x 14-inch sheet
DMPAPER_11X17 0x0011	11 x 17-inch sheet
DMPAPER_12X11 0x005A	12 x 11-inch sheet<106>
DMPAPER_A3 0x0008	A3 sheet, 297 x 420 millimeters
DMPAPER_A3_ROTATED 0x004C	A3 rotated sheet, 420 x 297 millimeters<107>
DMPAPER_A4 0x0009	A4 sheet, 210 x 297 millimeters
DMPAPER_A4_ROTATED 0x004D	A4 rotated sheet, 297 x 210 millimeters<108>
DMPAPER_A4SMALL 0x000A	A4 small sheet, 210 x 297 millimeters
DMPAPER_A5 0x000B	A5 sheet, 148 x 210 millimeters
DMPAPER_A5_ROTATED 0x004E	A5 rotated sheet, 210 x 148 millimeters<109>
DMPAPER_A6 0x0046	A6 sheet, 105 x 148 millimeters<110>
DMPAPER_A6_ROTATED 0x0053	A6 rotated sheet, 148 x 105 millimeters<111>
DMPAPER_B4 0x000C	B4 sheet, 250 x 354 millimeters
DMPAPER_B4_JIS_ROTATED 0x004F	B4 (JIS) rotated sheet, 364 x 257 millimeters<112>
DMPAPER_B5 0x000D	B5 sheet, 182 x 257-millimeter paper
DMPAPER_B5_JIS_ROTATED 0x0050	B5 (JIS) rotated sheet, 257 x 182 millimeters<113>
DMPAPER_B6_JIS 0x0058	B6 (JIS) sheet, 128 x 182 millimeters<114>
DMPAPER_B6_JIS_ROTATED 0x0059	B6 (JIS) rotated sheet, 182 x 128 millimeters<115>
DMPAPER_CSHEET 0x0018	C Sheet, 17 x 22 inches

Name/value	Meaning
DMPAPER_DBL_JAPANESE_POSTCARD 0x0045	Double Japanese Postcard, 200 x 148 millimeters<116>
DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED 0x0052	Double Japanese Postcard Rotated, 148 x 200 millimeters<117>
DMPAPER_DSHEET 0x0019	D Sheet, 22 x 34 inches
DMPAPER_ENV_9 0x0013	#9 Envelope, 3 7/8 x 8 7/8 inches
DMPAPER_ENV_10 0x0014	#10 Envelope, 4 1/8 x 9 1/2 inches
DMPAPER_ENV_11 0x0015	#11 Envelope, 4 1/2 x 10 3/8 inches
DMPAPER_ENV_12 0x0016	#12 Envelope, 4 3/4 x 11 inches
DMPAPER_ENV_14 0x0017	#14 Envelope, 5 x 11 1/2 inches
DMPAPER_ENV_C5 0x001C	C5 Envelope, 162 x 229 millimeters
DMPAPER_ENV_C3 0x001D	C3 Envelope, 324 x 458 millimeters
DMPAPER_ENV_C4 0x001E	C4 Envelope, 229 x 324 millimeters
DMPAPER_ENV_C6 0x001F	C6 Envelope, 114 x 162 millimeters
DMPAPER_ENV_C65 0x0020	C65 Envelope, 114 x 229 millimeters
DMPAPER_ENV_B4 0x0021	B4 Envelope, 250 x 353 millimeters
DMPAPER_ENV_B5 0x0022	B5 Envelope, 176 x 250 millimeters
DMPAPER_ENV_B6 0x0023	B6 Envelope, 176 x 125 millimeters
DMPAPER_ENV_DL 0x001B	DL Envelope, 110 x 220 millimeters
DMPAPER_ENV_ITALY 0x0024	Italy Envelope, 110 x 230 millimeters
DMPAPER_ENV_MONARCH 0x0025	Monarch Envelope, 3 7/8 x 7 1/2 inches

Name/value	Meaning
DMPAPER_ENV_PERSONAL 0x0026	6 3/4 Envelope, 3 5/8 x 6 1/2 inches
DMPAPER_ESHEET 0x001A	E Sheet, 34 x 44 inches
DMPAPER_EXECUTIVE 0x0007	Executive, 7 1/4 x 10 1/2 inches
DMPAPER_FANFOLD_US 0x0027	US Std Fanfold, 14 7/8 x 11 inches
DMPAPER_FANFOLD_STD_GERMAN 0x0028	German Std Fanfold, 8 1/2 x 12 inches
DMPAPER_FANFOLD_LGL_GERMAN 0x0029	German Legal Fanfold, 8 x 13 inches
DMPAPER_FOLIO 0x000E	Folio, 8 1/2 x 13-inch paper
DMPAPER_JAPANESE_POSTCARD_ROTATED 0x0051	Japanese Postcard Rotated, 148 x 100 millimeters<118>
DMPAPER_JENV_CHOU3 0x0049	Japanese Envelope Chou #3<119>
DMPAPER_JENV_CHOU3_ROTATED 0x0056	Japanese Envelope Chou #3 Rotated<120>
DMPAPER_JENV_CHOU4 0x004A	Japanese Envelope Chou #4<121>
DMPAPER_JENV_CHOU4_ROTATED 0x0057	Japanese Envelope Chou #4 Rotated<122>
DMPAPER_JENV_KAKU2 0x0047	Japanese Envelope Kaku #2<123>
DMPAPER_JENV_KAKU2_ROTATED 0x0054	Japanese Envelope Kaku #2 Rotated<124>
DMPAPER_JENV_KAKU3 0x0048	Japanese Envelope Kaku #3<125>
DMPAPER_JENV_KAKU3_ROTATED 0x0055	Japanese Envelope Kaku #3 Rotated<126>
DMPAPER_JENV_YOU4 0x005B	Japanese Envelope You #4<127>
DMPAPER_JENV_YOU4_ROTATED 0x005C	Japanese Envelope You #4<128>
DMPAPER_LEDGER 0x0004	Ledger, 17 x 11 inches

Name/value	Meaning
DMPAPER_LETTER_ROTATED 0x004B	Letter Rotated, 11 by 8 1/2 inches
DMPAPER_LETTERSMALL 0x0002	Letter Small, 8 1/2 x 11 inches
DMPAPER_NOTE 0x0012	Note, 8 1/2 x 11-inches
DMPAPER_P16K 0x005D	PRC 16K, 146 x 215 millimeters<129>
DMPAPER_P16K_ROTATED 0x006A	PRC 16K Rotated, 215 x 146 millimeters<130>
DMPAPER_P32K 0x005E	PRC 32K, 97 x 151 millimeters<131>
DMPAPER_P32K_ROTATED 0x006B	PRC 32K Rotated, 151 x 97 millimeters<132>
DMPAPER_P32KBIG 0x005F	PRC 32K(Big) 97 x 151 millimeters<133>
DMPAPER_P32KBIG_ROTATED 0x006C	PRC 32K(Big) Rotated, 151 x 97 millimeters<134>
DMPAPER_PENV_1 0x0060	PRC Envelope #1, 102 by 165 millimeters<135>
DMPAPER_PENV_1_ROTATED 0x006D	PRC Envelope #1 Rotated, 165 x 102 millimeters<136>
DMPAPER_PENV_2 0x0061	PRC Envelope #2, 102 x 176 millimeters<137>
DMPAPER_PENV_2_ROTATED 0x006E	PRC Envelope #2 Rotated, 176 x 102 millimeters<138>
DMPAPER_PENV_3 0x0062	PRC Envelope #3, 125 x 176 millimeters<139>
DMPAPER_PENV_3_ROTATED 0x006F	PRC Envelope #3 Rotated, 176 x 125 millimeters<140>
DMPAPER_PENV_4 0x0063	PRC Envelope #4, 110 x 208 millimeters<141>
DMPAPER_PENV_4_ROTATED 0x0070	PRC Envelope #4 Rotated, 208 x 110 millimeters<142>
DMPAPER_PENV_5 0x0064	PRC Envelope #5, 110 x 220 millimeters<143>
DMPAPER_PENV_5_ROTATED 0x0071	PRC Envelope #5 Rotated, 220 x 110 millimeters<144>

Name/value	Meaning
DMPAPER_PENV_6 0x0065	PRC Envelope #6, 120 x 230 millimeters<145>
DMPAPER_PENV_6_ROTATED 0x0072	PRC Envelope #6 Rotated, 230 x 120 millimeters<146>
DMPAPER_PENV_7 0x0066	PRC Envelope #7, 160 x 230 millimeters<147>
DMPAPER_PENV_7_ROTATED 0x0073	PRC Envelope #7 Rotated, 230 x 160 millimeters<148>
DMPAPER_PENV_8 0x0067	PRC Envelope #8, 120 x 309 millimeters<149>
DMPAPER_PENV_8_ROTATED 0x0074	PRC Envelope #8 Rotated, 309 x 120 millimeters<150>
DMPAPER_PENV_9 0x0068	PRC Envelope #9, 229 x 324 millimeters<151>
DMPAPER_PENV_9_ROTATED 0x0075	PRC Envelope #9 Rotated, 324 x 229 millimeters<152>
DMPAPER_PENV_10 0x0069	PRC Envelope #10, 324 x 458 millimeters<153>
DMPAPER_PENV_10_ROTATED 0x0076	PRC Envelope #10 Rotated, 458 x 324 millimeters<154>
DMPAPER_QUARTO 0x000F	Quarto, 215 x 275 millimeter paper
DMPAPER_STATEMENT 0x0006	Statement, 5 1/2 x 8 1/2 inches
DMPAPER_TABLOID 0x0003	Tabloid, 11 x 17 inches
0x0100 ≤ *value*	The value is device-specific.

dmPaperLength (2 bytes): If the DM_PAPERLENGTH bit is set in the **dmFields** field, the value of this field specifies the length of the paper, in tenths of a millimeter, to use in the printer for which the job is destined.

dmPaperWidth (2 bytes): If the DM_PAPERWIDTH bit is set in the **dmFields** field, the value of this field specifies the width of the paper, in tenths of a millimeter, to use in the printer for which the job is destined.

dmScale (2 bytes): If the DM_SCALE bit is set in the **dmFields** field, the value of this field specifies the percentage factor by which the printed output is to be scaled.

dmCopies (2 bytes): If the DM_COPIES bit is set in the **dmFields** field, the value of this field specifies the number of copies to be printed, if the device supports multiple-page copies.

dmDefaultSource (2 bytes): If the DM_DEFAULTSOURCE bit is set in the **dmFields** field, the value of this field specifies the paper source.

The value of this field SHOULD be one of the following, or it MAY be a device-specific value that is greater than or equal to 0x0100.

Name/value	Meaning
DMBIN_UPPER 0x0001	Select the upper paper bin. This value is also used for the paper source for printers that only have one paper bin.
DMBIN_LOWER 0x0002	Select the lower bin.
DMBIN_MIDDLE 0x0003	Select the middle paper bin.
DMBIN_MANUAL 0x0004	Manually select the paper bin.
DMBIN_ENVELOPE 0x0005	Select the auto envelope bin.
DMBIN_ENVMANUAL 0x0006	Select the manual envelope bin.
DMBIN_AUTO 0x0007	Auto-select the bin.
DMBIN_TRACTOR 0x0008	Select the bin with the tractor paper.
DMBIN_SMALLFMT 0x0009	Select the bin with the smaller paper format.
DMBIN_LARGEFMT 0x000A	Select the bin with the larger paper format.
DMBIN_LARGECAPACITY 0x000B	Select the bin with large capacity.
DMBIN_CASSETTE 0x000E	Select the cassette bin.
DMBIN_FORMSOURCE 0x000F	Select the bin with the required form.

dmPrintQuality (2 bytes): If the DM_PRINTQUALITY bit is set in the **dmFields** field, the value of this field specifies the printer resolution. The value of this field MUST be either a positive value that specifies a device-dependent resolution in dots per inch (DPI) or one of the following four predefined device-independent values that are mapped to a device-specific resolution in an implementation-specific manner.

Name/value	Meaning
DMRES_HIGH 0xFFFC	High-resolution printouts
DMRES_MEDIUM 0xFFFD	Medium-resolution printouts

Name/value	Meaning
DMRES_LOW 0xFFFE	Low-resolution printouts
DMRES_DRAFT 0xFFFF	Draft-resolution printouts

dmColor (2 bytes): If the DM_COLOR bit is set in the **dmFields** field, the value of this field specifies the color mode to use with color printers. The value of this field MUST be one of the following.

Name/value	Meaning
DMRES_MONOCHROME 0x0001	Use monochrome printing mode.
DMRES_COLOR 0x0002	Use color printing mode.

dmDuplex (2 bytes): If the DM_DUPLEX bit is set in the **dmFields** field, the value of this field specifies duplex or double-sided printing for printers that are capable of duplex printing. The value of this field MUST be one of the following.

Name/value	Meaning
DMDUP_SIMPLEX 0x0001	Normal (non-duplex) printing.
DMDUP_VERTICAL 0x0002	Long-edge binding; that is, the long edge of the page is vertical.
DMDUP_HORIZONTAL 0x0003	Short-edge binding; that is, the long edge of the page is horizontal.

dmYResolution (2 bytes): If the DM_YRESOLUTION bit is set in the **dmFields**, the value of this field specifies the y-resolution, in dots per inch, of the printer.

dmTTOption (2 bytes): If the DM_TTOPTION bit is set in the **dmFields** field, the value of this field specifies how TrueType fonts MUST be printed. The value of this field MUST be one of the following.

Name/value	Meaning
DMTT_BITMAP 0x0001	Prints TrueType fonts as graphics. This is the default action for dot-matrix printers.
DMTT_DOWNLOAD 0x0002	Downloads TrueType fonts as soft fonts. This is the default action for Hewlett-Packard printers that use **printer control language (PCL)**.
DMTT_SUBDEV 0x0003	Substitutes device fonts for TrueType fonts. This is the default action for PostScript printers.
DMTT_DOWNLOAD_OUTLINE 0x0004	Downloads TrueType fonts as outline soft fonts.<155>

dmCollate (2 bytes): If the DM_COLLATE bit is set in the **dmFields** field, the value of this field specifies whether collation MUST be used when printing multiple copies. The value of this field MUST be one of the following:

Value	Meaning
DMCOLLATE_FALSE 0x0000	Do not collate when printing multiple copies.
DMCOLLATE_TRUE 0x0001	Collate when printing multiple copies.

dmFormName (64 bytes): This field is a 32-element array of 16-bit Unicode characters. If the DM_FORMNAME bit is set in the **dmFields** field, the value of this field specifies the name of the form to use, for example, "Letter" or "Legal". The value of this field is restricted to 32 characters, including the trailing null. Form names that are longer than 32 characters, including the trailing null, MUST be truncated to fit the array.

reserved0 (2 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved1 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved2 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved3 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

dmNup (4 bytes): If the DM_NUP bit is set in the **dmFields**, the value of this field specifies the responsibility for performing page layout for **N-Up Printing**. It MUST be one of the following values:

Name/Value	Meaning
DMNUP_SYSTEM 0x00000001	The print server does the page layout.
DMNUP_ONEUP 0x00000002	The application does the page layout.

reserved4 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

dmICMMethod (4 bytes): If the DM_ICMMETHOD bit is set in the **dmFields** field, the value of this field specifies how Image Color Management (ICM) is handled. For a non-ICM application, this field determines if ICM is enabled or disabled. For ICM applications, the system examines this field to determine how to handle ICM support. The value of this field MUST be one of the following predefined values or a printer driver-defined value greater than or equal to the value of 0x00000100.

Name/Value	Meaning
DMICMMETHOD_NONE 0x00000001	Specifies that ICM is disabled.
DMICMMETHOD_SYSTEM	Specifies that ICM is handled by the system on which the page description

Name/Value	Meaning
0x00000002	language (PDL) data is generated.
DMICMMETHOD_DRIVER 0x00000003	Specifies that ICM is handled by the printer driver.
DMICMMETHOD_DEVICE 0x00000004	Specifies that ICM is handled by the destination device.

dmICMIntent (4 bytes): If the DM_ICMINTENT bit is set in the **dmFields** field, the value of this field specifies which **color matching** method, or intent, MUST be used by default. This field is primarily for non-ICM applications. ICM applications can establish intents by using the ICM functions. The value of this field MUST be one of the following predefined values, or a printer driver defined value greater than or equal to the value of 0x00000100.

Name/Value	Meaning
DMICM_SATURATE 0x00000001	Color matching SHOULD optimize for color saturation.
DMICM_CONTRAST 0x00000002	Color matching SHOULD optimize for color contrast.
DMICM_COLORIMETRIC 0x00000003	Color matching SHOULD optimize to match the exact color requested.
DMICM_ABS_COLORIMETRIC 0x00000004	Color matching SHOULD optimize to match the exact color requested without **white point** mapping.

dmMediaType (4 bytes): If the DM_MEDIATYPE bit is set in the **dmFields** field, the value of this field specifies the type of media to print on. The value of this field MUST be one of the following predefined values or else a printer driver-defined value greater than or equal to the value of 0x00000100.

Name/Value	Meaning
DMMEDIA_STANDARD 0x00000001	Plain paper
DMMEDIA_TRANSPARENCY 0x00000002	Transparent film
DMMEDIA_GLOSSY 0x00000003	Glossy paper

dmDitherType (4 bytes): If the DM_DITHERTYPE bit is set in the **dmFields** field, the value of this field specifies how **dithering** is to be done. The value of this field MUST be one of the following predefined values or else a printer driver-defined value greater than or equal to 0x00000100.

Name/Value	Meaning
DMDITHER_NONE 0x00000001	No dithering.
DMDITHER_COARSE 0x00000002	Dithering with a coarse brush.

Name/Value	Meaning
DMDITHER_FINE 0x00000003	Dithering with a fine brush.
DMDITHER_LINEART 0x00000004	Line art dithering, a special dithering method that produces well defined borders between black, white, and gray scaling.
DMDITHER_ERRORDIFFUSION 0x00000005	Windows 95/98/ME: Error diffusion dithering.
DMDITHER_RESERVED6 0x00000006	Same as DMDITHER_LINEART.
DMDITHER_RESERVED7 0x00000007	Same as DMDITHER_LINEART.
DMDITHER_RESERVED8 0x00000008	Same as DMDITHER_LINEART.
DMDITHER_RESERVED9 0x00000009	Same as DMDITHER_LINEART.
DMDITHER_GRAYSCALE 0x0000000A	Device does gray scaling.

reserved5 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved6 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved7 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

reserved8 (4 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

dmDriverExtraData (variable): This field MAY contain implementation-specific printer driver data. Its size in bytes is specified by the value of the **dmDriverExtra** field.<156>

2.2.2.1.1 PostScript Driver Extra Data

Information about PostScript Driver Extra Data can be found in Appendix B: Product Behavior.<157>

2.2.2.1.2 Generic Driver Extra Data

Information about Generic Driver Extra Data can be found in Appendix B: Product Behavior.<158>

2.2.2.1.3 OEM Driver Extra Data

Information about OEM (vendor-supplied) Driver Extra Data can be found in Appendix B: Product Behavior <159>.

2.2.2.1.4 Print Ticket Driver Extra Data

Information about Print Ticket Driver Extra Data can be found in Appendix B: Product Behavior <160>

2.2.2.2 Members in Custom-Marshaled INFO structures

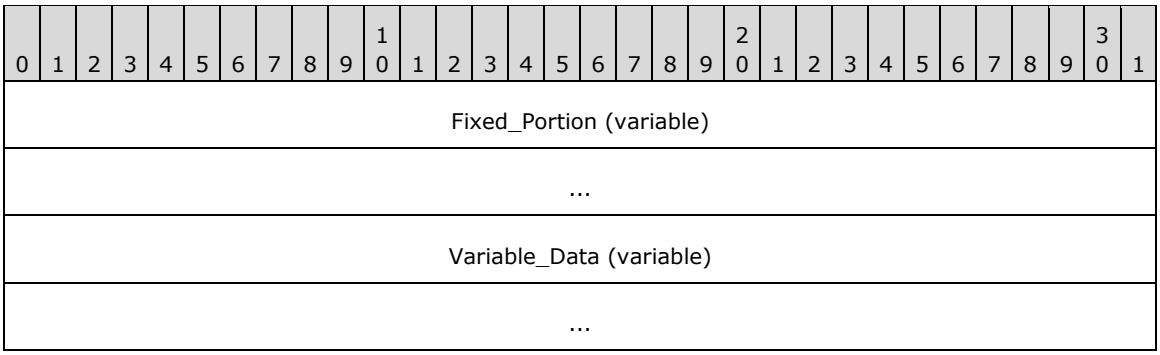
Unless noted otherwise, pointer and string pointer members of IDL-marshaled INFO structure forms are represented in the corresponding custom-marshaled INFO structure forms according to the following rules:

- **MEMBERNAME** is derived by trimming the leading "p" from the pointer, string pointer, or multisz pointer member of the IDL-marshaled INFO structure.
- The custom-marshaled INFO structure form contains an **Offset** member whose name is derived by appending "Offset" to **MEMBERNAME**.
- The **Offset** member is a 32-bit unsigned integer that specifies the number of bytes from the start of the structure to the start of the bytes making up the pointed-to data, string, or multisz. That data, string, or multisz area is represented in the custom-marshaled structure by a member whose name is derived by appending "Array" to **MEMBERNAME**. The length of that member is variable and includes the terminating null character for string data or the two terminating null characters for multisz data, respectively.
- If the pointer, string pointer, or multisz pointer member in the IDL-marshaled structure form is optional (that is, it can be NULL), it can be represented by a zero **Offset** in a custom-marshaled structure. The corresponding **Array** member is then considered optional and is present only if the **Offset** is not zero.

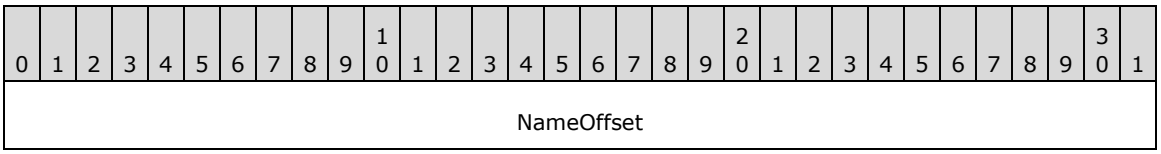
Unless noted otherwise, all other members of IDL-marshaled INFO structure forms are represented identically in the corresponding custom-marshaled INFO structure forms.

2.2.2.3 DATATYPES_INFO_1

The DATATYPES_INFO_1 structure contains information about the data type used to record a print job.

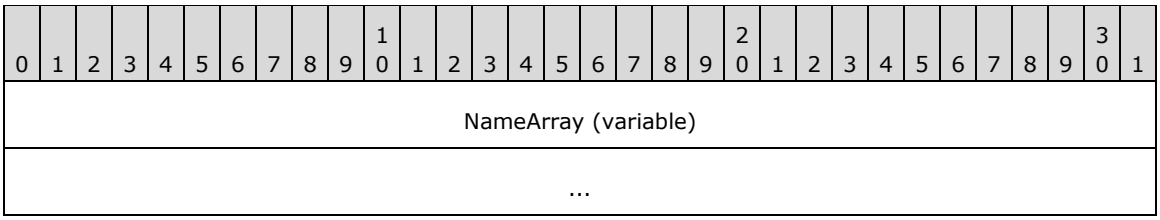


Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.



NameOffset (4 bytes): A 32-bit unsigned integer that specifies the number of bytes from the start of the structure to the **NameArray** member.

Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

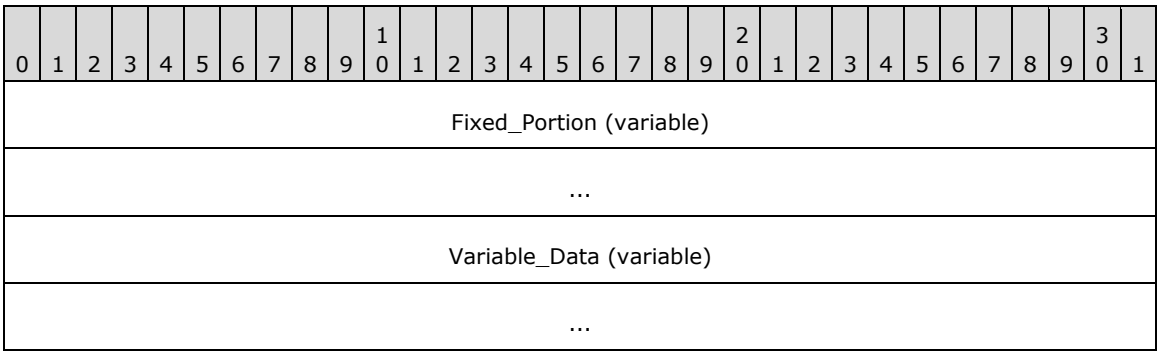


NameArray (variable): A string that specifies the data type used to record a print job. The location of this buffer is determined by the value of the **NameOffset** member. For rules governing data type names, see section 2.2.4.2.

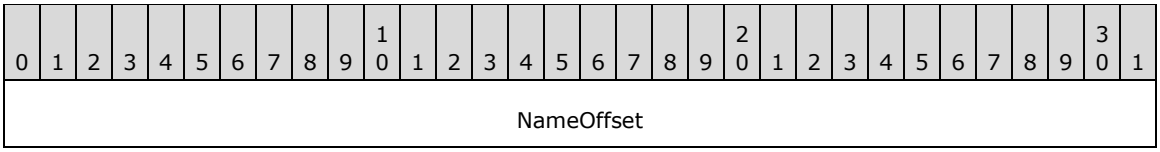
2.2.2.4 _DRIVER_INFO

2.2.2.4.1 _DRIVER_INFO_1

The _DRIVER_INFO_1 structure specifies printer driver information. It is a custom-marshaled form of the DRIVER_INFO_1 (section 2.2.1.5.1) structure.

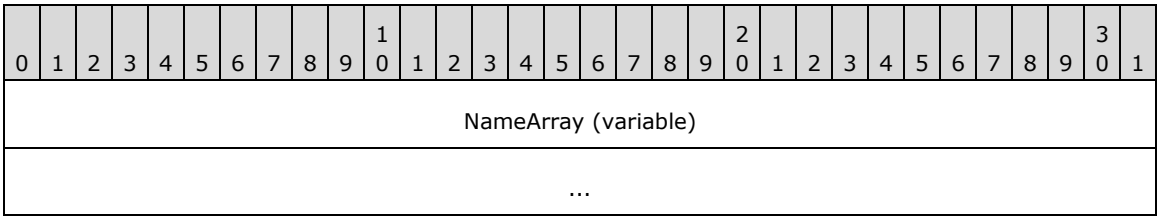


Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.



NameOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the **NameArray** member.

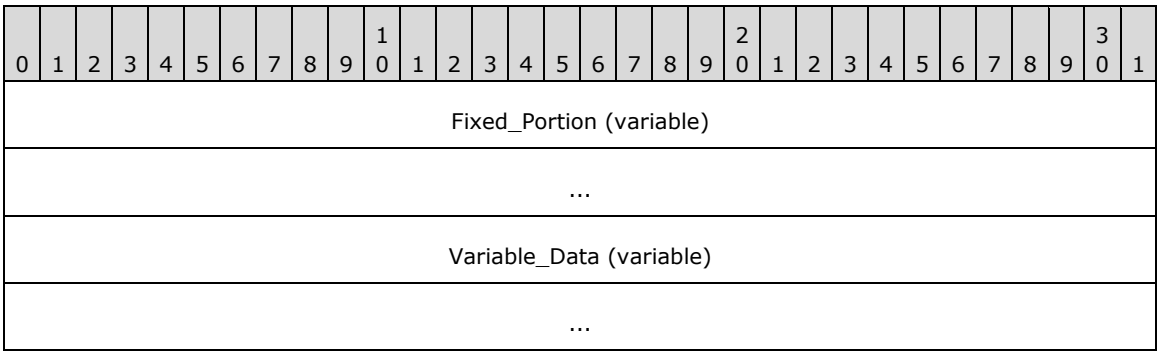
Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.



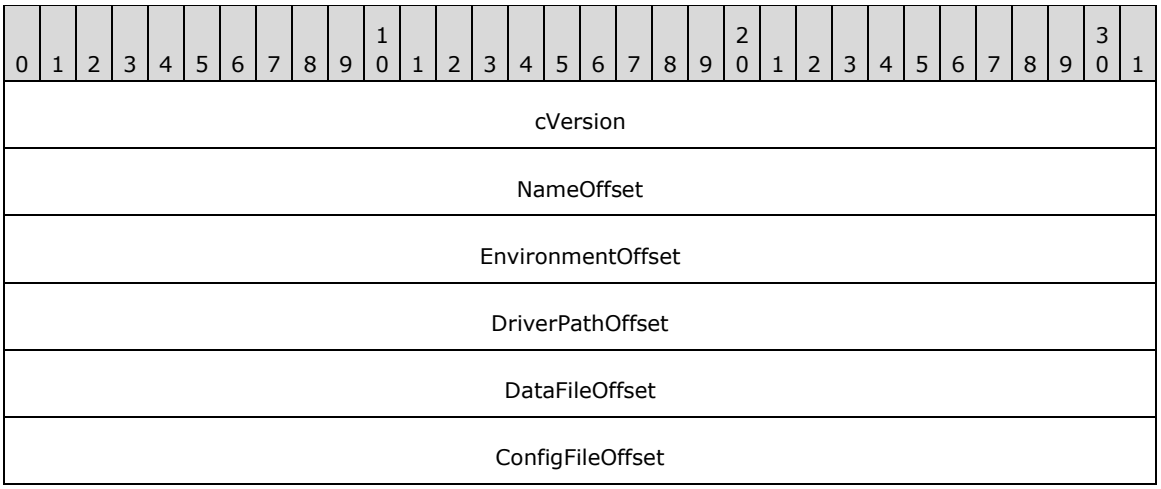
NameArray (variable): This is a string that MUST specify the Printer Driver Name. The location of this buffer is determined by the value of the **NameOffset** member.

2.2.2.4.2 _DRIVER_INFO_2

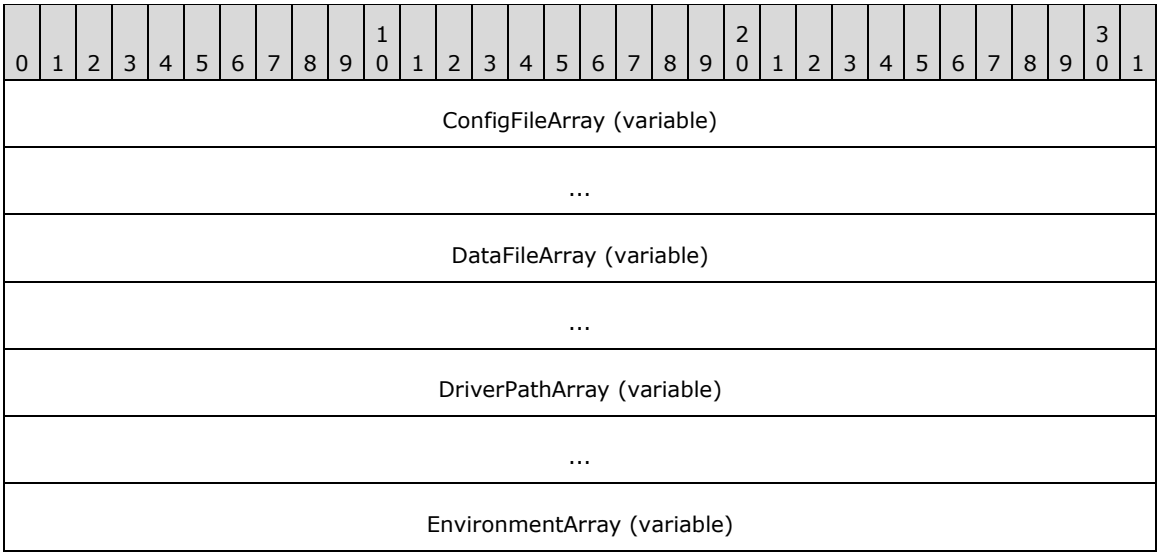
The _DRIVER_INFO_2 structure specifies printer driver information. It is a custom-marshaled form of the DRIVER_INFO_2 (section 2.2.1.5.2) structure.



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.



...
NameArray (variable)
...

2.2.2.4.3 _DRIVER_INFO_3

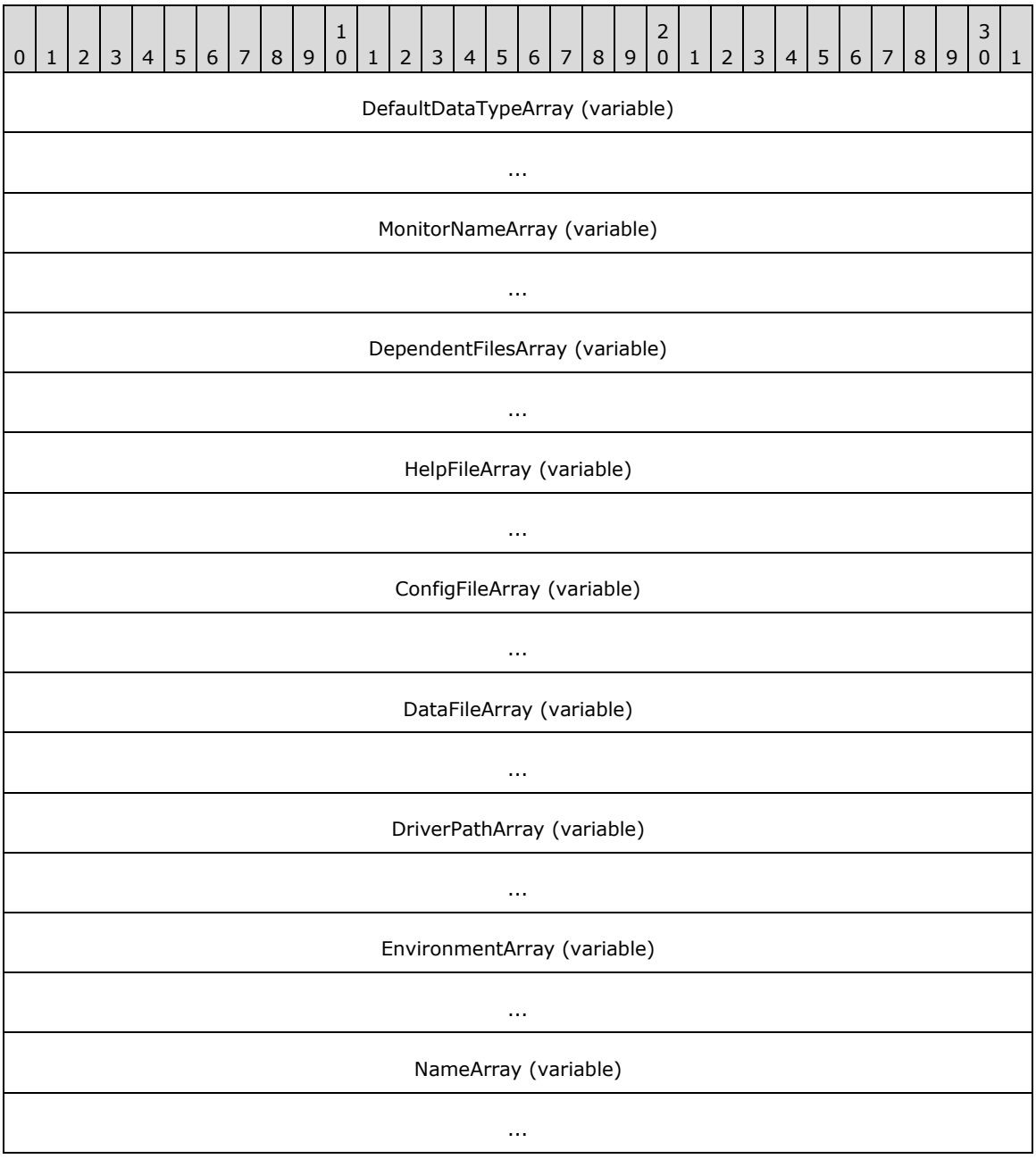
The _DRIVER_INFO_3 structure specifies printer driver information. It is a custom-marshaled form of the RPC_DRIVER_INFO_3 (section 2.2.1.5.3) structure.<161>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

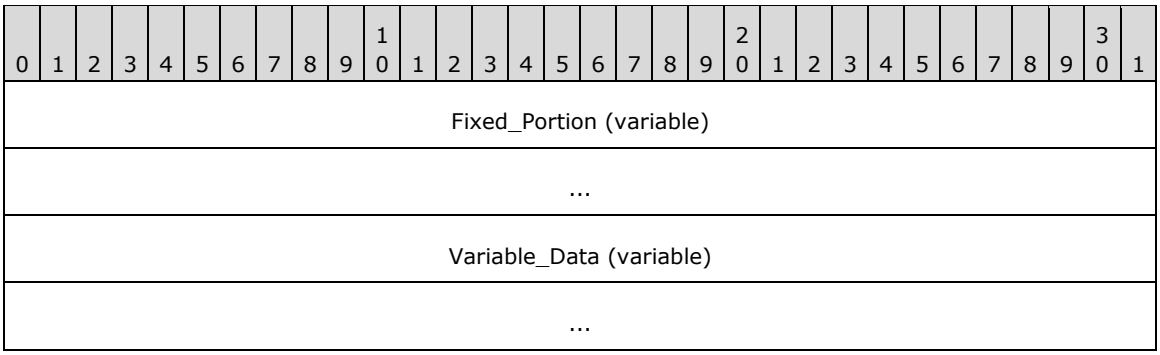
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cVersion																															
NameOffset																															
EnvironmentOffset																															
DriverPathOffset																															
DataFileOffset																															
ConfigFileOffset																															
HelpFileOffset																															
DependentFilesOffset																															
MonitorNameOffset																															
DefaultDataTypeOffset																															

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

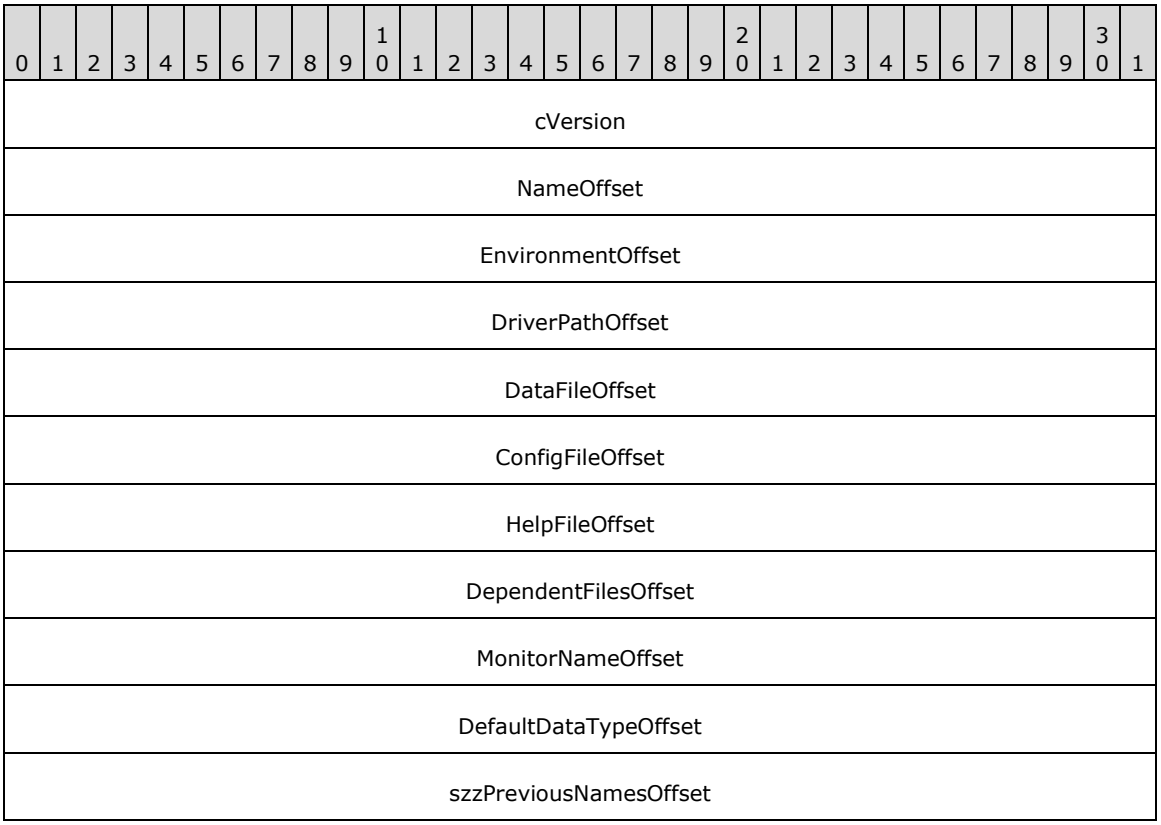


2.2.2.4.4 _DRIVER_INFO_4

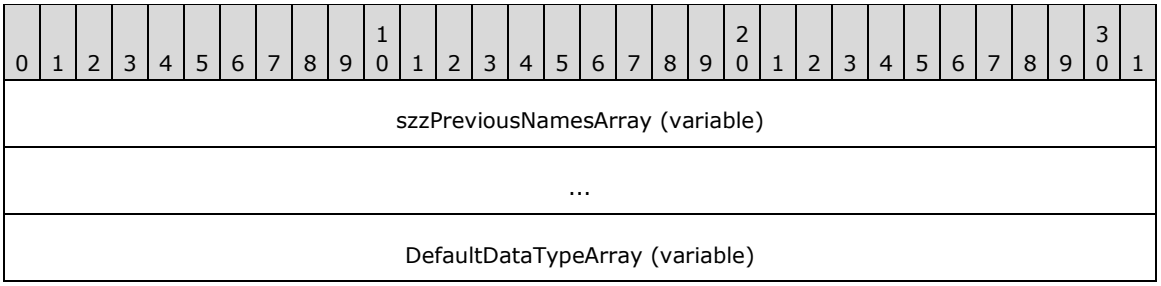
The _DRIVER_INFO_4 structure specifies printer driver information.<162> It is a custom-marshaled form of the RPC_DRIVER_INFO_4 (section 2.2.1.5.4) structure.



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.



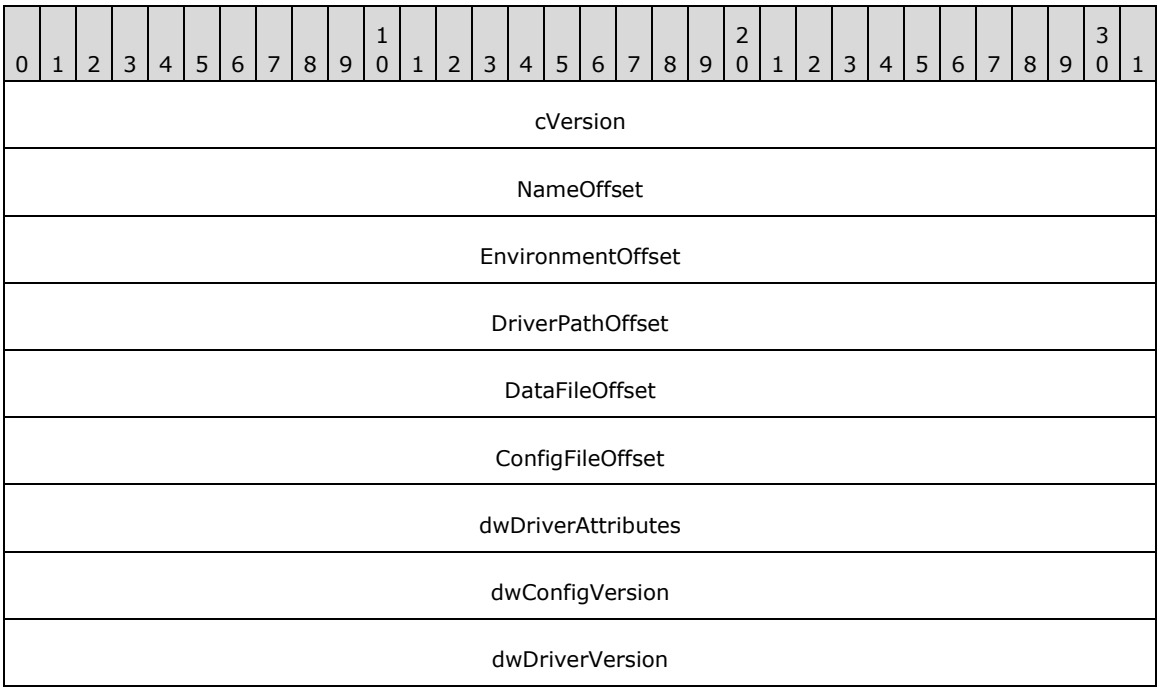
...
MonitorNameArray (variable)
...
DependentFilesArray (variable)
...
HelpFileArray (variable)
...
ConfigFileArray (variable)
...
DataFileArray (variable)
...
DriverPathArray (variable)
...
EnvironmentArray (variable)
...
NameArray (variable)
...

2.2.2.4.5 _DRIVER_INFO_5

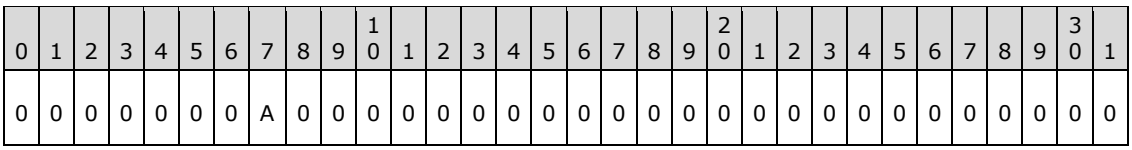
The _DRIVER_INFO_5 structure specifies printer driver information.<163>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



dwDriverAttributes (4 bytes): A bit field that specifies attributes of the printer driver.



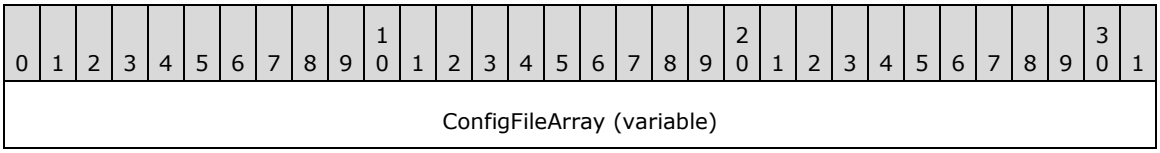
Where the bits are defined as:

Value	Description
A PRINTER_DRIVER_PACKAGE_AWARE	The printer driver is part of a driver package.

dwConfigVersion (4 bytes): The number of times the printer driver configuration file has been upgraded (replaced with a newer binary) or downgraded (replaced with an older binary) since the system was restarted.

dwDriverVersion (4 bytes): The number of times the printer driver executable file has been upgraded (replaced with a newer binary) or downgraded (replaced with an older binary) since the system was restarted.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.



...
DataFileArray (variable)
...
DriverPathArray (variable)
...
EnvironmentArray (variable)
...
NameArray (variable)
...

Members not defined in this section are identical to members in the _DRIVER_INFO_4 structure (section 2.2.2.4.4).

2.2.2.4.6 _DRIVER_INFO_6

The DRIVER_INFO_6 structure specifies printer driver information.<164> It is a custom-marshaled form of the RPC_DRIVER_INFO_6 (section 2.2.1.5.5) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cVersion																															
NameOffset																															
EnvironmentOffset																															
DriverPathOffset																															

DataFileOffset
ConfigFileOffset
HelpFileOffset
DependentFilesOffset
MonitorNameOffset
DefaultDataTypeOffset
szzPreviousNamesOffset
ftDriverDate.dwLowDateTime
ftDriverDate.dwHighDateTime
PaddingForAlignment
dwlDriverVersion
...
MfgNameOffset
OEMUrlOffset
HardwareIDOffset
ProviderOffset

PaddingForAlignment (4 bytes): 4 bytes of padding to align the **dwlDriverVersion** field on an 8-byte boundary. The contents of this field MUST be ignored.

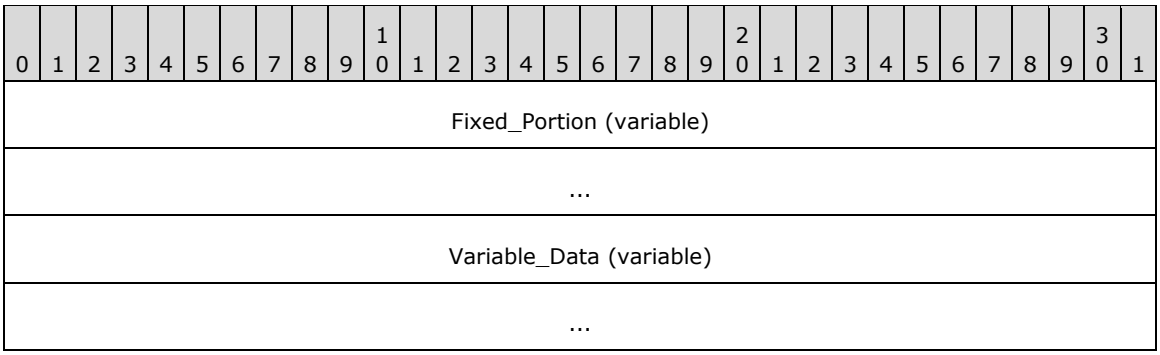
Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ProviderArray (variable)																															
...																															
HardwareIDArray (variable)																															
...																															
OEMUrlArray (variable)																															

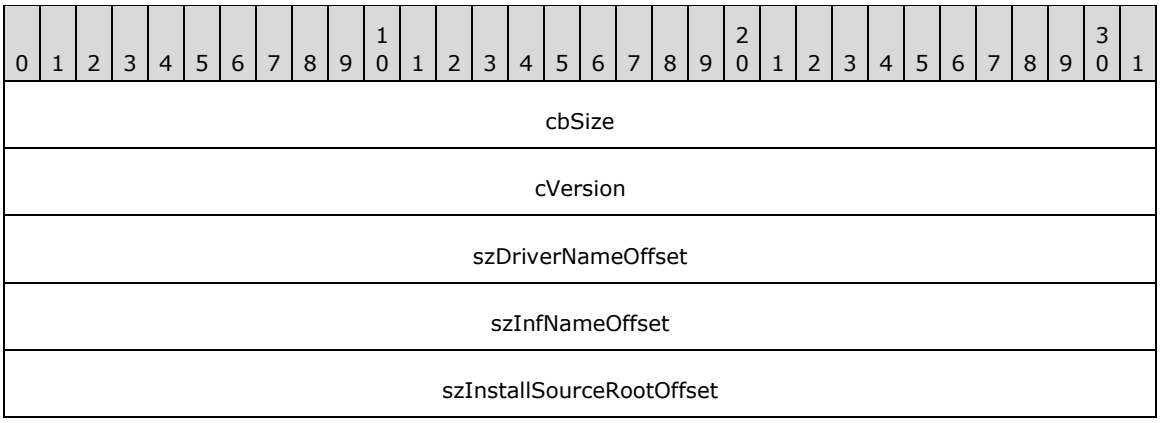
...
MfgNameArray (variable)
...
szzPreviousNamesArray (variable)
...
DefaultDataTypeArray (variable)
...
MonitorNameArray (variable)
...
DependentFilesArray (variable)
...
HelpFileArray (variable)
...
ConfigFileArray (variable)
...
DataFileArray (variable)
...
DriverPathArray (variable)
...
EnvironmentArray (variable)
...
NameArray (variable)
...

2.2.2.4.7 _DRIVER_INFO_7

The _DRIVER_INFO_7 structure specifies printer driver information.<165>



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



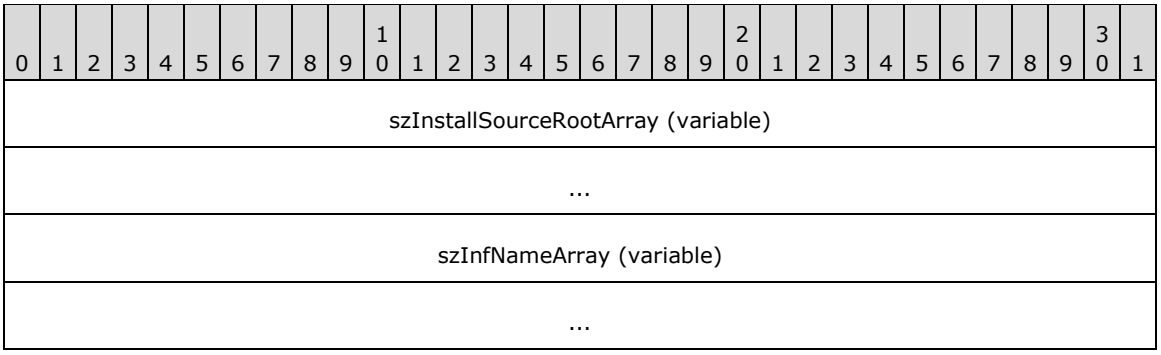
cbSize (4 bytes): The value of this member MUST specify the size, in bytes, of the _DRIVER_INFO_7 data structure.

szDriverNameOffset (4 bytes): This member is synonymous with the **NameOffset** member.

szInfNameOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **szInfNameArray** member.

szInstallSourceRootOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **szInstallSourceRootArray** member.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.



szDriverNameArray (variable)
...

szInstallSourceRootArray (variable): This member MUST be a string that specifies the path from which the driver MUST be installed. The location of this buffer is determined by the value of the **szInstallSourceRootOffset** member.

szInfNameArray (variable): This member MUST be a string that specifies the name of the driver's installation configuration file. The location of this buffer is determined by the value of the **szInfNameOffset** member.<166>

szDriverNameArray (variable): This member is synonymous with the **NameArray** member.

2.2.2.4.8 _DRIVER_INFO_8

The _DRIVER_INFO_8 structure specifies printer driver information. It is a custom-marshaled form of the RPC_DRIVER_INFO_8 (section 2.2.1.5.6) structure.<167>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

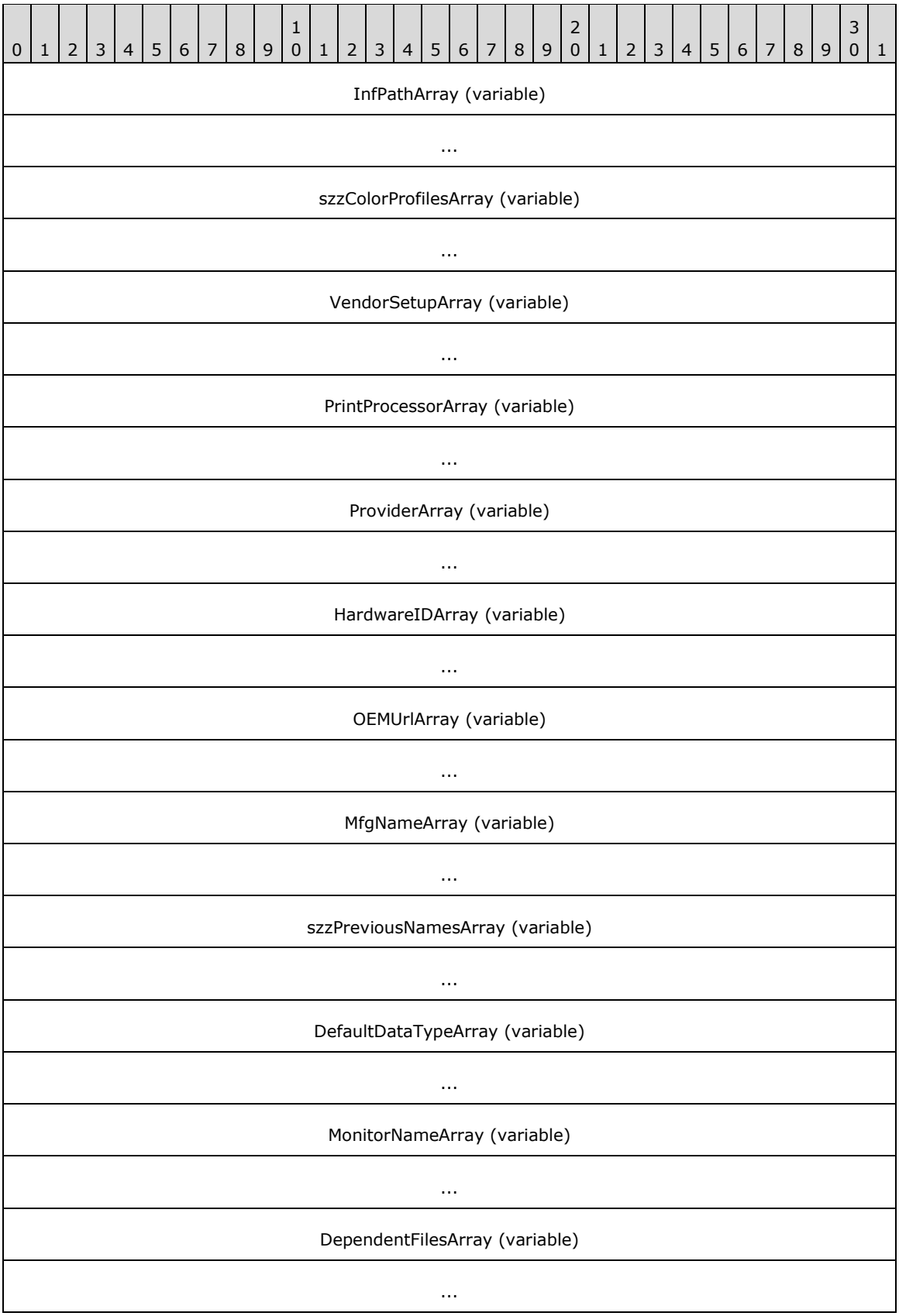
Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cVersion																															
NameOffset																															
EnvironmentOffset																															
DriverPathOffset																															
DataFileOffset																															
ConfigFileOffset																															
HelpFileOffset																															
DependentFilesOffset																															

MonitorNameOffset
DefaultDataTypeOffset
szzPreviousNamesOffset
ftDriverDate.dwLowDateTime
ftDriverDate.dwHighDateTime
PaddingForAlignment
dwlDriverVersion
...
MfgNameOffset
OEMUrlOffset
HardwareIDOffset
ProviderOffset
PrintProcessorOffset
VendorSetupOffset
szzColorProfilesOffset
InfPathOffset
dwPrinterDriverAttributes
szzCoreDependenciesOffset
ftMinInboxDriverVerDate
...
dwlMinInboxDriverVerVersion
...

PaddingForAlignment (4 bytes): 4 bytes of padding to align the **dwlDriverVersion** field on an 8-byte boundary. The contents of this field MUST be ignored.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.



HelpFileArray (variable)
...
ConfigFileArray (variable)
...
DataFileArray (variable)
...
DriverPathArray (variable)
...
EnvironmentArray (variable)
...
NameArray (variable)
...
szzCoreDependenciesArray (variable)
...

2.2.2.4.9 _DRIVER_INFO_101

The DRIVER_INFO_101 structure specifies printer driver information.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cVersion																															
NameOffset																															
EnvironmentOffset																															
FileInfoOffset																															
dwFileCount																															
MonitorNameOffset																															
DefaultDataTypeOffset																															
szzPreviousNamesOffset																															
ftDriverDate																															
...																															
dwlDriverVersion																															
...																															
MfgNameOffset																															
OEMUrlOffset																															
HardwareIDOffset																															
ProviderOffset																															

FileInfoOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **FileInfoArray** member.

dwFileCount (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of DRIVER_FILE_INFO structures in the **FileInfoArray** member.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ProviderArray (variable)																															
...																															

HardwareIDArray (variable)
...
OEMUrlArray (variable)
...
MfgNameArray (variable)
...
szzPreviousNamesArray (variable)
...
DefaultDataTypeArray (variable)
...
MonitorNameArray (variable)
...
FileInfoArray (variable)
...
EnvironmentArray (variable)
...
NameArray (variable)
...

FileInfoArray (variable): This member MUST be an array of DRIVER_FILE_INFO structures. The number of elements in the array MUST be the same as the value of the **dwFileCount** member.

Note: Member definitions for all members not defined in this section are identical to members in DRIVER_INFO_8 (section 2.2.2.4.8).

2.2.2.4.10 _DRIVER_FILE_INFO

The _DRIVER_FILE_INFO structure specifies information about a file belonging to a printer driver.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															

...
Variable_Data (variable)
...

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FileNameOffset																															
FileType																															
FileVersion																															

FileNameOffset (4 bytes): A 32-bit unsigned integer that specifies the number of bytes from the start of the _DRIVER_INFO_101 (section 2.2.2.4.9) structure that contains this structure to the start of the **FileNameArray** field.

FileType (4 bytes): A 32-bit unsigned integer that specifies the file type using one of the constant values from the following table.

Value	Meaning
0x00000000	The file is a rendering driver module executable.
0x00000001	The file is a configuration module executable.
0x00000002	The file is a driver data file.
0x00000003	The file is a driver help file.
0x00000004	The file is a dependent file with a type other than the preceding file types.

FileVersion (4 bytes): The version of the printer driver file. The actual value is specific to the implementation of the printer driver. Since printer driver are developed by third parties, it is not practical to list all possible values.

A print client MAY use this field to detect a change to the printer driver on a print server.<168>

Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

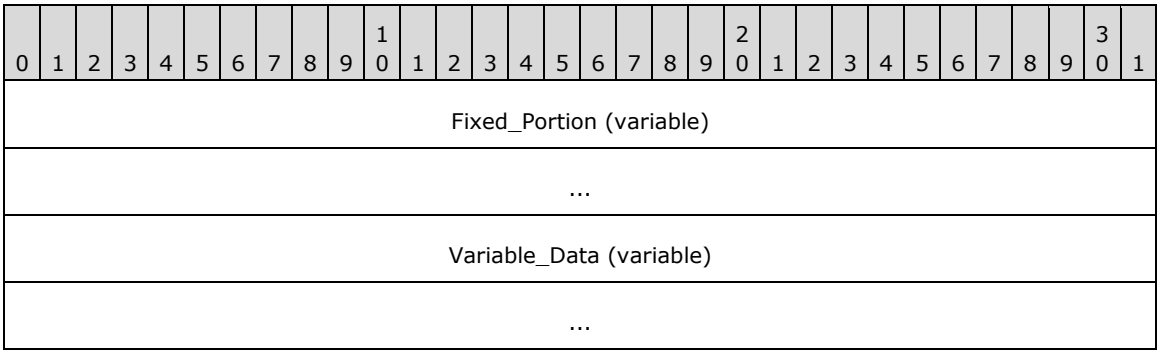
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FileNameArray (variable)																															
...																															

FileNameArray (variable): A null-terminated string that specifies the name of the file in Unicode UTF-16LE characters. The location of this buffer is determined by the value of the **FileNameOffset** field.

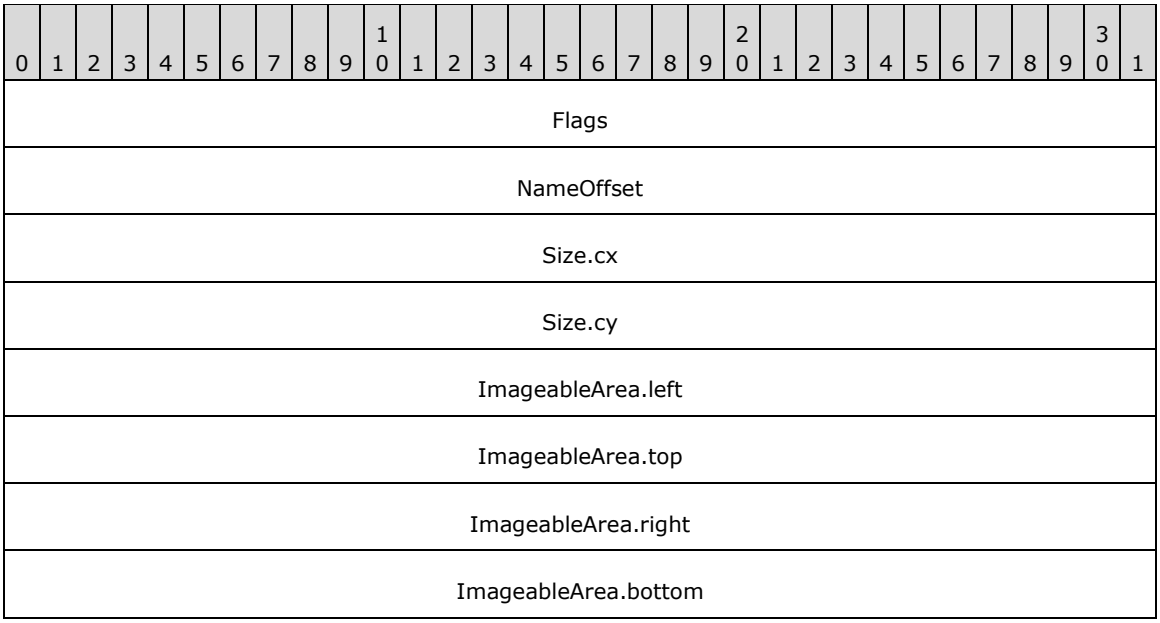
2.2.2.5 _FORM_INFO

2.2.2.5.1 _FORM_INFO_1

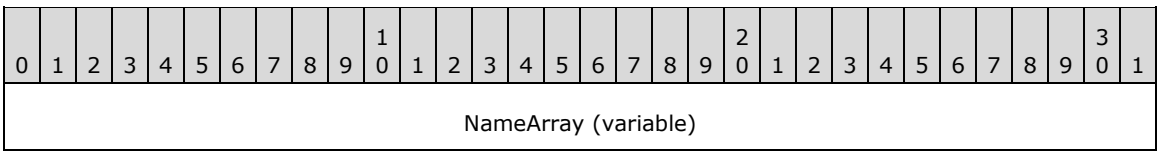
The _FORM_INFO_1 structure specifies printer media information. It is a custom-marshaled form of the FORM_INFO_1 (section 2.2.1.6.1) structure.



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.



...

All members not defined in this section are specified in sections 2.2.1.3.2 and 2.2.2.3.

2.2.2.5.2 _FORM_INFO_2

The _FORM_INFO_2 structure specifies printer media information. It is a custom-marshaled form of the RPC_FORM_INFO_2 (section 2.2.1.6.2) structure.<169>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Flags																															
NameOffset																															
Size.cx																															
Size.cy																															
ImageableArea.left																															
ImageableArea.top																															
ImageableArea.right																															
ImageableArea.bottom																															
KeywordOffset																															
StringType																															
MuiDllOffset																															
dwResourceID																															
DisplayNameOffset																															

wLangID	unused

unused (2 bytes): A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NameArray (variable)																															
...																															
KeywordArray (variable)																															
...																															
MuiDllArray (variable)																															
...																															
DisplayNameArray (variable)																															
...																															

2.2.2.6 _JOB_INFO

2.2.2.6.1 _JOB_INFO_1

The _JOB_INFO_1 structure specifies print job information. It is a custom-marshaled form of the JOB_INFO_1 (section 2.2.1.7.1) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
JobId																															
PrinterNameOffset																															
MachineNameOffset																															
UserNameOffset																															
DocumentOffset																															
DatatypeOffset																															
StatusOffset																															
Status																															
Priority																															
Position																															
TotalPages																															
PagesPrinted																															
Submitted.wYear																	Submitted.wMonth															
Submitted.wDayOfWeek																	Submitted.wDay															
Submitted.wHour																	Submitted.wMinute															
Submitted.wSecond																	Submitted.wMilliseconds															

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
StatusArray (variable)																															
...																															
DatatypeArray (variable)																															
...																															
DocumentArray (variable)																															

...
UserNameArray (variable)
...
MachineNameArray (variable)
...
PrinterNameArray (variable)
...

Fields that are not defined in this section are specified in section 2.2.1.3.3.

2.2.2.6.2 _JOB_INFO_2

The _JOB_INFO_2 structure specifies print job information. It is a custom-marshaled form of the JOB_INFO_2 (section 2.2.1.7.2) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
JobId																															
PrinterNameOffset																															
MachineNameOffset																															
UserNameOffset																															
DocumentOffset																															
NotifyNameOffset																															
DatatypeOffset																															

PrintProcessorOffset	
ParametersOffset	
DriverNameOffset	
DevModeOffset	
StatusOffset	
SecurityDescriptorOffset	
Status	
Priority	
Position	
StartTime	
UntilTime	
TotalPages	
Size	
Submitted.wYear	Submitted.wMonth
Submitted.wDayOfWeek	Submitted.wDay
Submitted.wHour	Submitted.wMinute
Submitted.wSecond	Submitted.wMilliseconds
Time	
PagesPrinted	

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

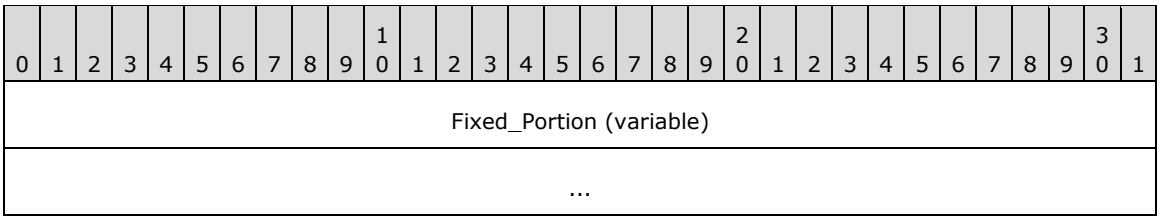
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SecurityDescriptorArray (variable)																															
...																															
StatusArray (variable)																															

...
DevModeArray (variable)
...
DriverNameArray (variable)
...
ParametersArray (variable)
...
PrintProcessorArray (variable)
...
DatatypeArray (variable)
...
NotifyNameArray (variable)
...
DocumentArray (variable)
...
UserNameArray (variable)
...
MachineNameArray (variable)
...
PrinterNameArray (variable)
...

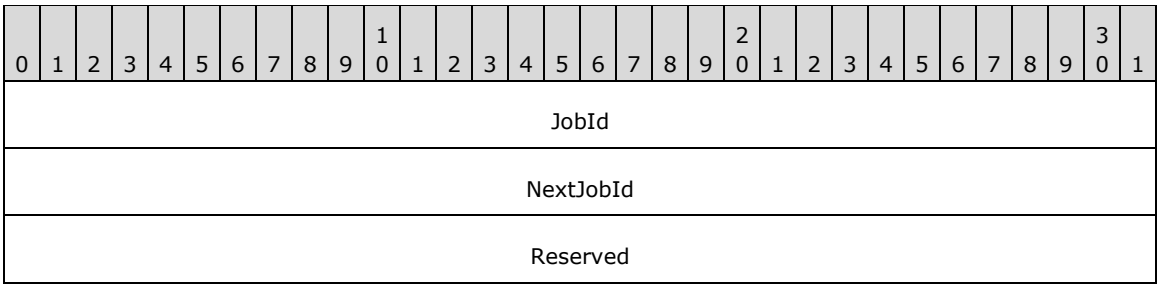
Fields that are not defined in this section are specified in section 2.2.1.3.3.

2.2.2.6.3 _JOB_INFO_3

The _JOB_INFO_3 structure specifies information about the order of print jobs, and it is used to alter the order of print jobs.<170> It is a custom-marshaled form of the JOB_INFO_3 (section 2.2.1.7.3) structure.



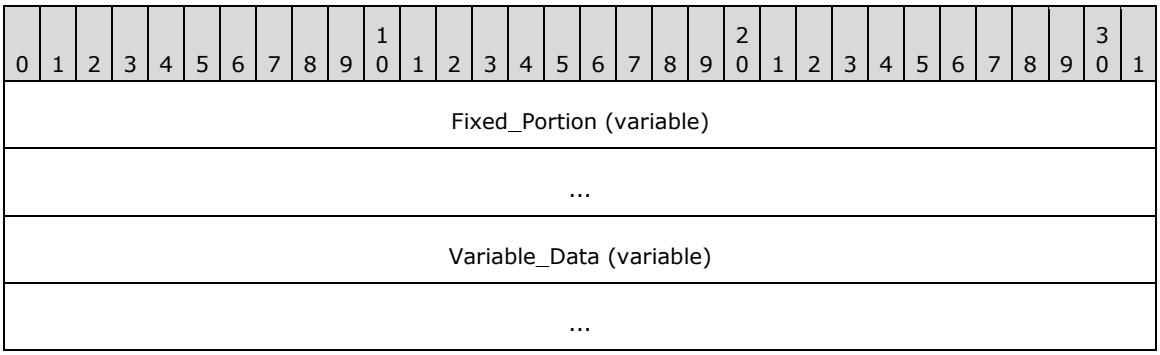
Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



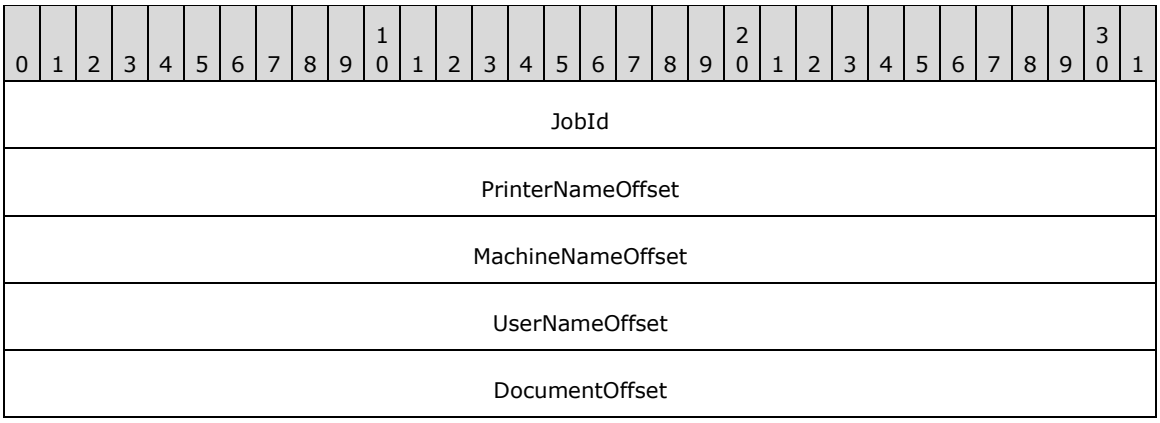
Fields that are not defined in this section are specified in sections 2.2.1.7.3, and 2.2.1.3.

2.2.2.6.4 _JOB_INFO_4

The _JOB_INFO_4 structure specifies print job information. It is a custom-marshaled form of the JOB_INFO_4 (section 2.2.1.7.4) structure.<171>

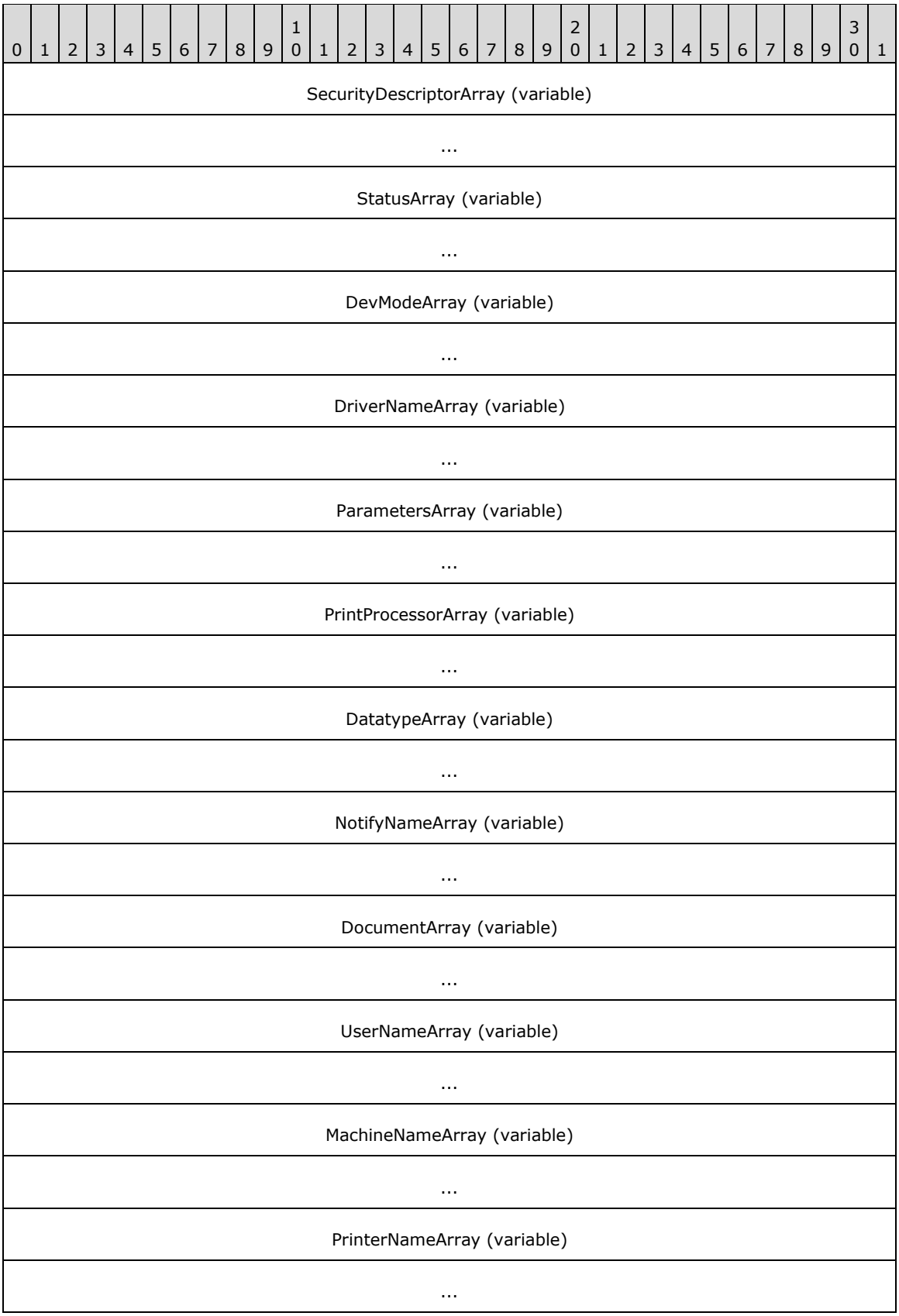


Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



NotifyNameOffset	
DatatypeOffset	
PrintProcessorOffset	
ParametersOffset	
DriverNameOffset	
DevModeOffset	
StatusOffset	
SecurityDescriptorOffset	
Status	
Priority	
Position	
StartTime	
UntilTime	
TotalPages	
Size	
Submitted.wYear	Submitted.wMonth
Submitted.wDayOfWeek	Submitted.wDay
Submitted.wHour	Submitted.wMinute
Submitted.wSecond	Submitted.wMilliseconds
Time	
PagesPrinted	
SizeHigh	

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

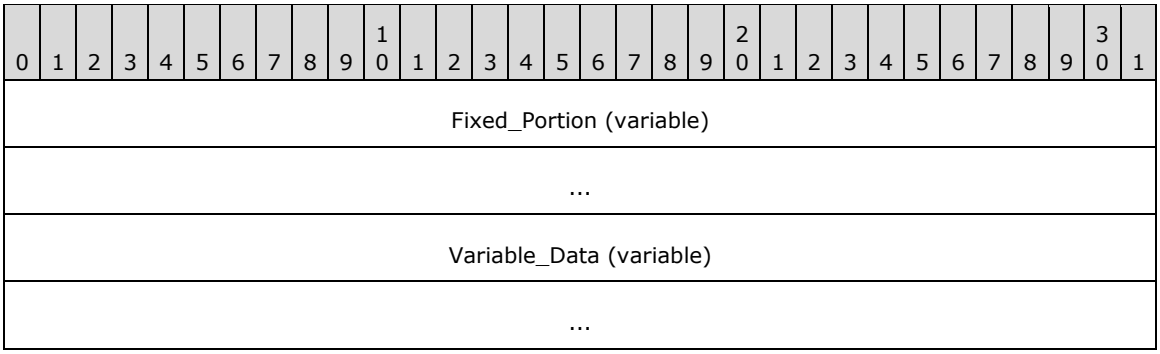


Fields that are not defined in this section are specified in section 2.2.1.3.3.

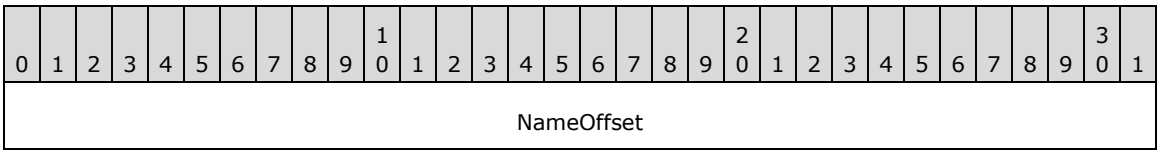
2.2.2.7 _MONITOR_INFO

2.2.2.7.1 _MONITOR_INFO_1

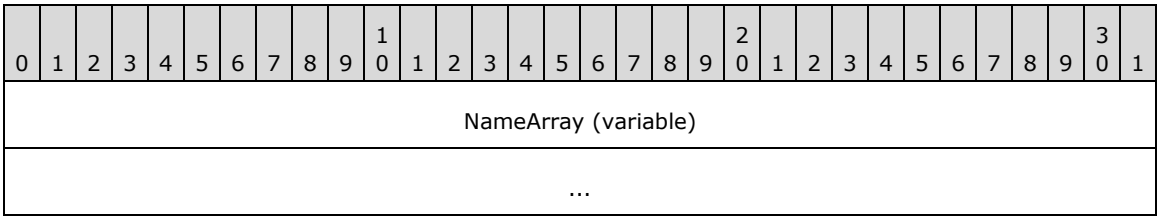
The _MONITOR_INFO_1 structure identifies an installed port monitor. It is a custom-marshaled form of the MONITOR_INFO_1 (section 2.2.1.8.1) structure.



Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.

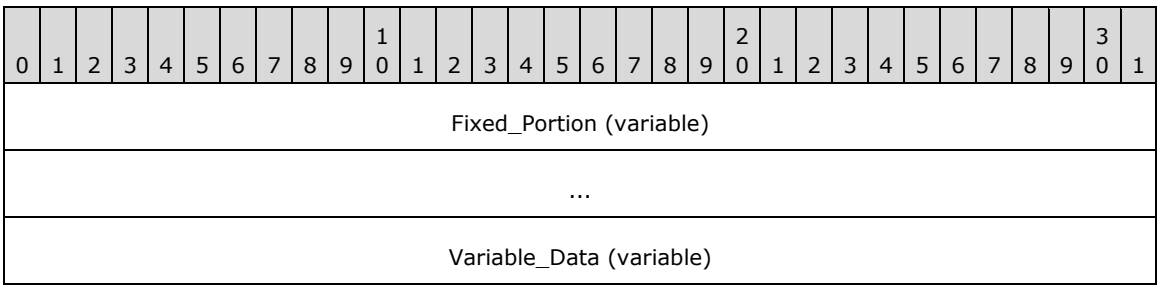


Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.



2.2.2.7.2 _MONITOR_INFO_2

The _MONITOR_INFO_2 structure is used to identify a port monitor. It is a custom-marshaled form of the MONITOR_INFO_2 (section 2.2.1.8.2) structure.



...

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NameOffset																															
EnvironmentOffset																															
DLLNameOffset																															

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
DLLNameArray (variable)																															
...																															
EnvironmentArray (variable)																															
...																															
NameArray (variable)																															
...																															

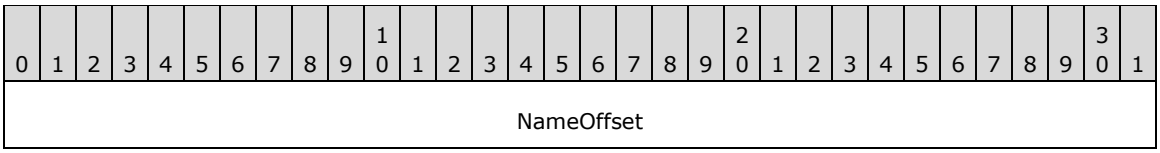
2.2.2.8 _PORT_INFO

2.2.2.8.1 _PORT_INFO_1

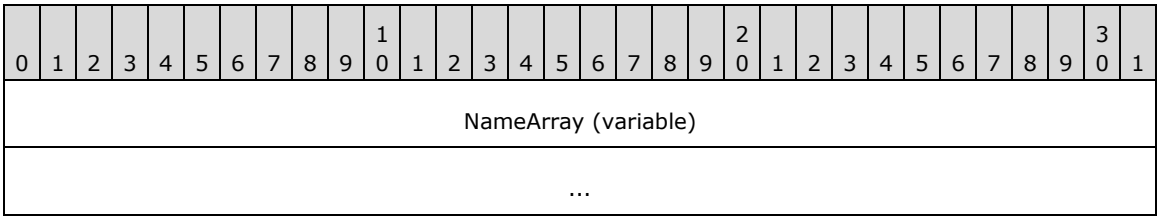
The _PORT_INFO_1 structure specifies information about a printer port. It is a custom-marshaled form of the PORT_INFO_1 (section 2.2.1.9.1) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.

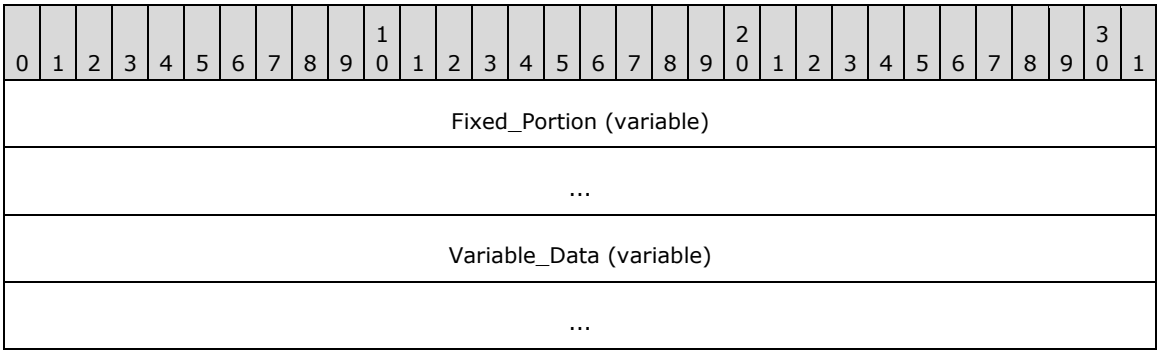


Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

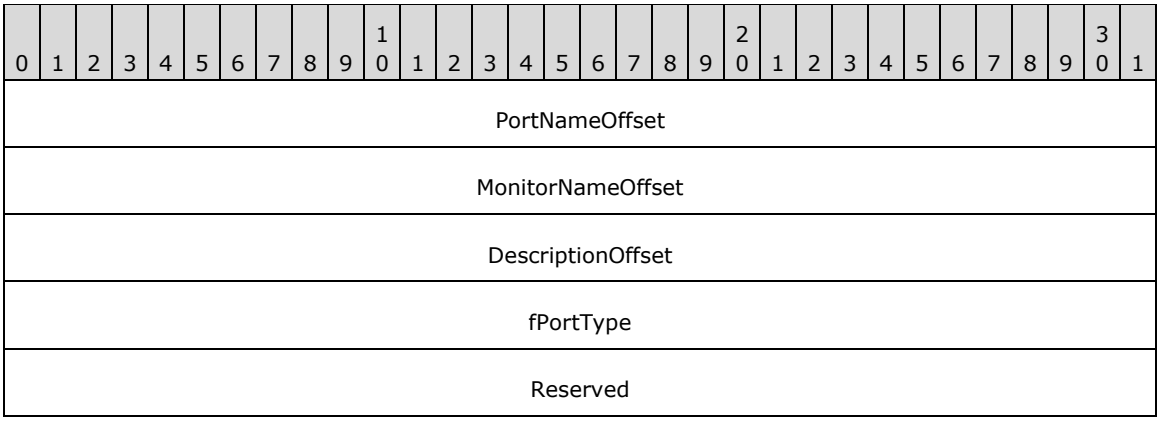


2.2.2.8.2 _PORT_INFO_2

The _PORT_INFO_2 structure specifies information about a printer port. It is a custom-marshaled form of the PORT_INFO_2 (section 2.2.1.9.2) structure.



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.



Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
DescriptionArray (variable)																															
...																															
MonitorNameArray (variable)																															
...																															
PortNameArray (variable)																															
...																															

2.2.2.9 _PRINTER_INFO

2.2.2.9.1 _PRINTER_INFO_STRESS

The _PRINTER_INFO_STRESS structure specifies printer diagnostic information. It is a custom-marshaled form of the PRINTER_INFO_STRESS (section 2.2.1.10.1) structure.<172> This form of the _PRINTER_INFO_STRESS structure corresponds to an information **Level** value of 0x00000000.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
PrinterNameOffset																															
ServerNameOffset																															
cJobs																															
cTotalJobs																															
cTotalBytes																															

stUpTime.wYear	stUpTime.wMonth
stUpTime.wDayOfWeek	stUpTime.wDay
stUpTime.wHour	stUpTime.wMinute
stUpTime.wSecond	stUpTime.wMilliseconds
MaxcRef	
cTotalPagesPrinted	
dwGetVersion	
fFreeBuild	
cSpooling	
cMaxSpooling	
cRef	
cErrorOutOfPaper	
cErrorNotReady	
cJobError	
dwNumberOfProcessors	
dwProcessorType	
dwHighPartTotalBytes	
cChangeID	
dwLastError	
Status	
cEnumerateNetworkPrinters	
cAddNetPrinters	
wProcessorArchitecture	wProcessorLevel
cRefIC	
dwReserved2	

dwReserved3

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
PrinterNameArray (variable)																															
...																															
ServerNameArray (variable)																															
...																															

2.2.2.9.2 _PRINTER_INFO_1

The _PRINTER_INFO_1 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_1 (section 2.2.1.10.2) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Flags																															
DescriptionOffset																															
NameOffset																															
CommentOffset																															

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CommentArray (variable)																															
...																															
NameArray (variable)																															
...																															
DescriptionArray (variable)																															
...																															

2.2.2.9.3 _PRINTER_INFO_2

The _PRINTER_INFO_2 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_2 (section 2.2.1.10.3) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ServerNameOffset																															
PrinterNameOffset																															
ShareNameOffset																															
PortNameOffset																															
DriverNameOffset																															
CommentOffset																															

LocationOffset
DevModeOffset
SepFileOffset
PrintProcessorOffset
DatatypeOffset
ParametersOffset
SecurityDescriptorOffset
Attributes
Priority
DefaultPriority
StartTime
UntilTime
Status
cJobs
AveragePPM

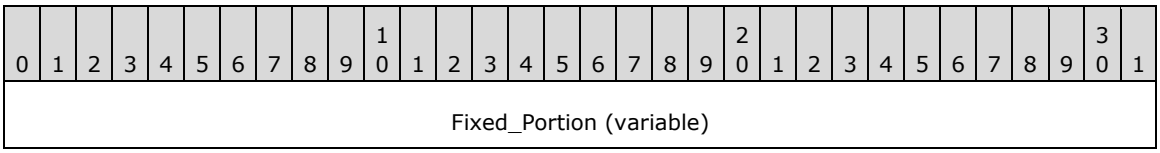
Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SecurityDescriptorArray (variable)
...
ParametersArray (variable)
...
DatatypeArray (variable)
...
PrintProcessorArray (variable)

...
SepFileArray (variable)
...
DevModeArray (variable)
...
LocationArray (variable)
...
CommentArray (variable)
...
DriverNameArray (variable)
...
PortNameArray (variable)
...
ShareNameArray (variable)
...
PrinterNameArray (variable)
...
ServerNameArray (variable)
...

2.2.2.9.4 _PRINTER_INFO_3

The _PRINTER_INFO_3 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_3 (section 2.2.1.10.4) structure.



...
Variable_Data (variable)
...

Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SecurityDescriptorOffset																															

Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SecurityDescriptorArray (variable)																															
...																															

2.2.2.9.5 _PRINTER_INFO_4

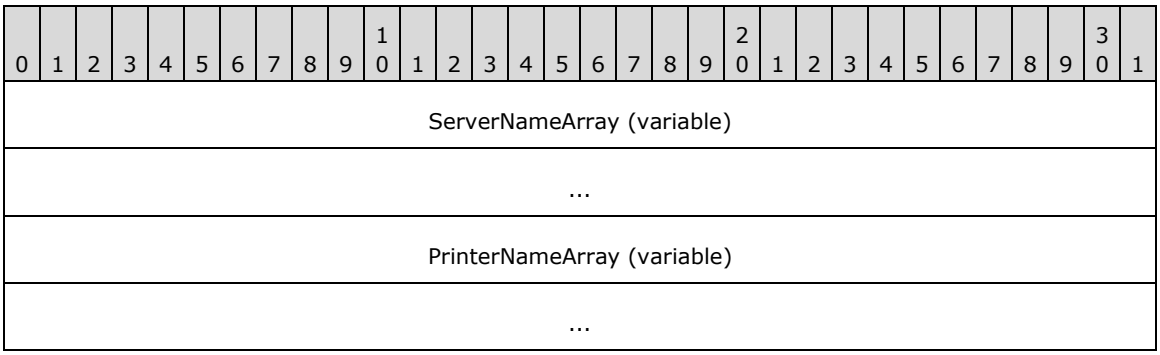
The _PRINTER_INFO_4 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_4 (section 2.2.1.10.5) structure.<173>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Fixed_Portion (variable)																															
...																															
Variable_Data (variable)																															
...																															

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

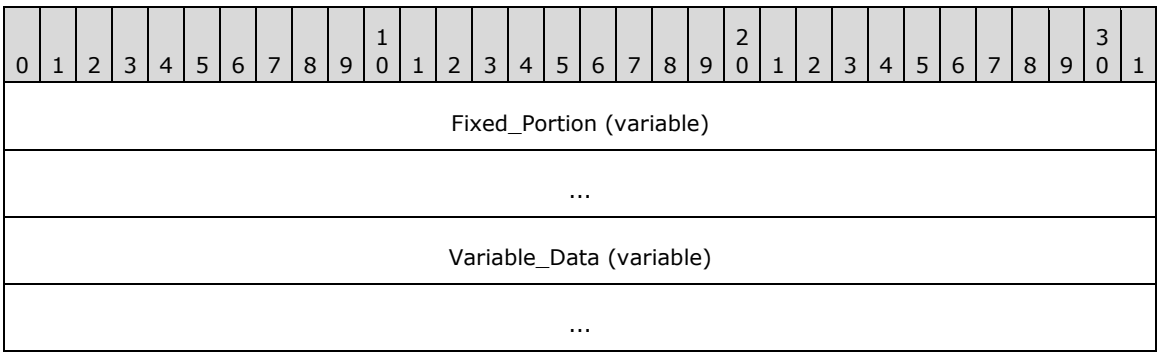
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
PrinterNameOffset																															
ServerNameOffset																															
Attributes																															

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

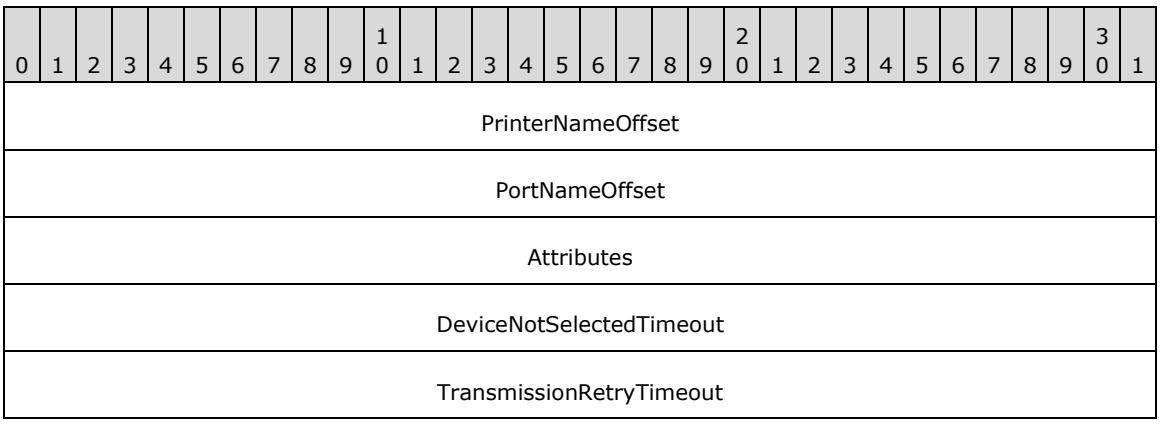


2.2.2.9.6 _PRINTER_INFO_5

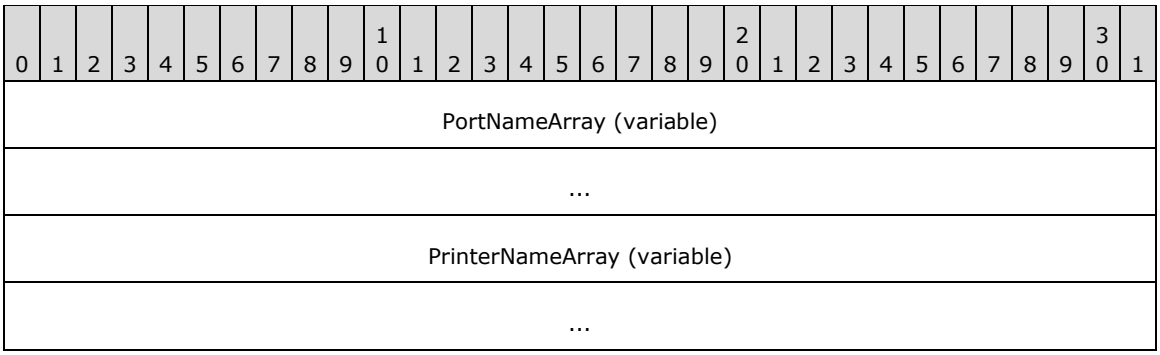
The _PRINTER_INFO_5 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_5 (section 2.2.1.10.6) structure.<174>



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

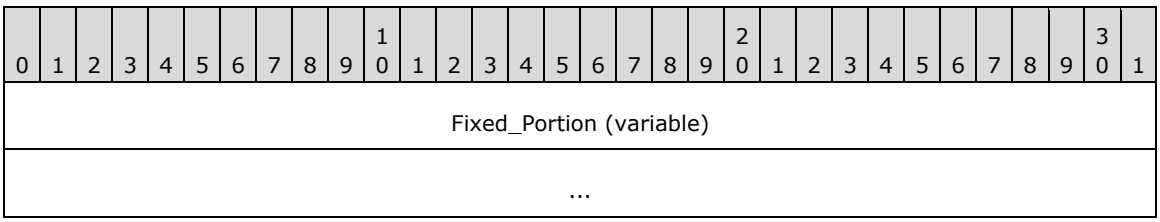


Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

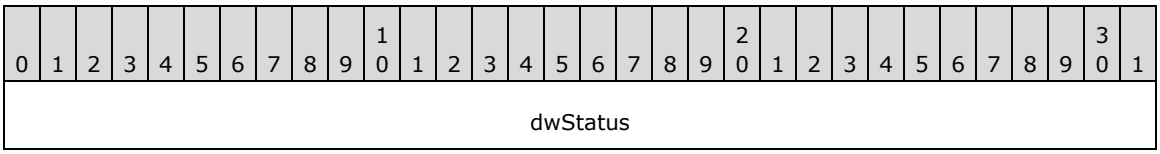


2.2.2.9.7 _PRINTER_INFO_6

The _PRINTER_INFO_6 structure specifies printer information. It is a custom-marshaled form of the PRINTER_INFO_6 (section 2.2.1.10.7) structure.<175>



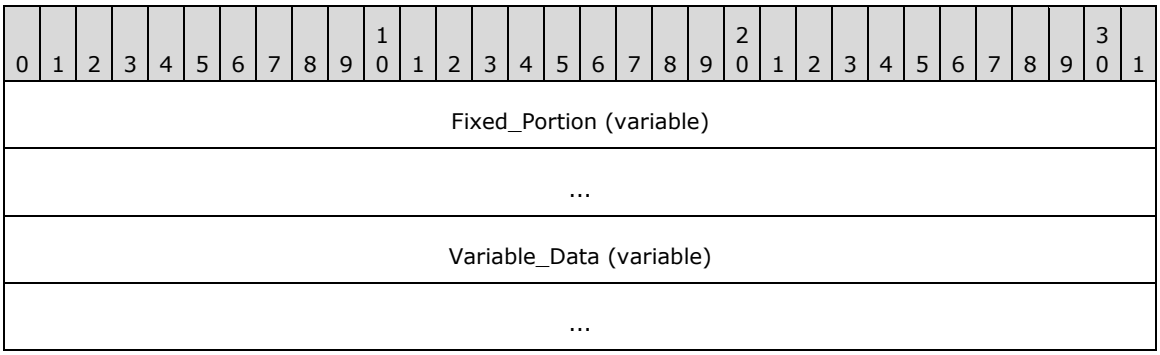
Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.



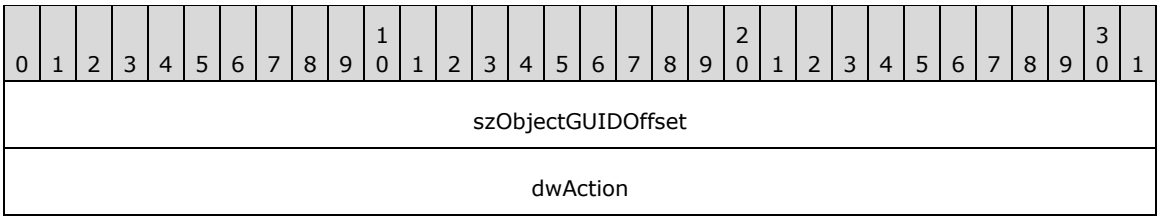
dwStatus (4 bytes): The new printer status, which is a value specified in Status and Attribute Values (section 2.2.3.12).

2.2.2.9.8 _PRINTER_INFO_7

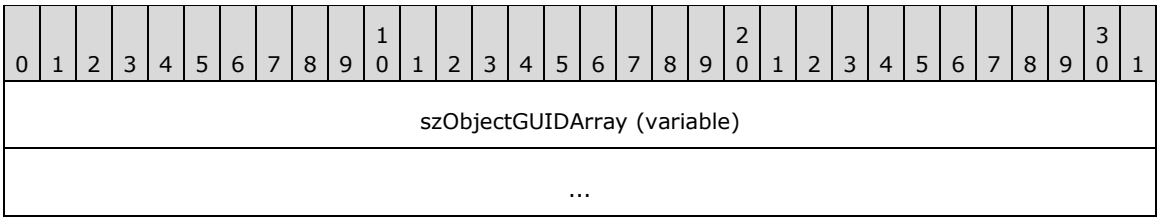
The _PRINTER_INFO_7 structure specifies printer information.<176> It is a custom-marshaled form of the PRINTER_INFO_7 (section 2.2.1.10.8) structure.



Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

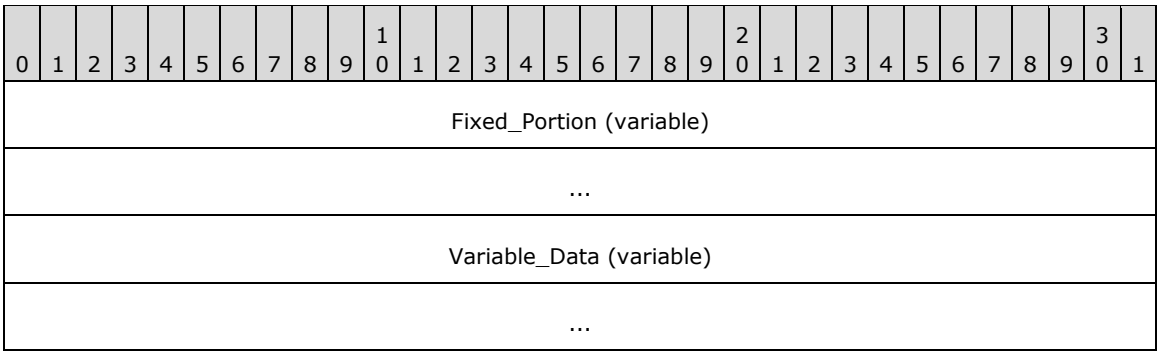


Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

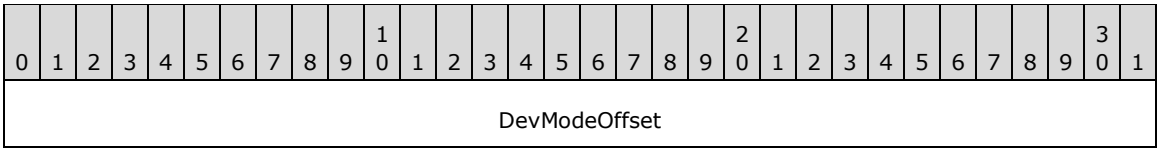


2.2.2.9.9 _PRINTER_INFO_8

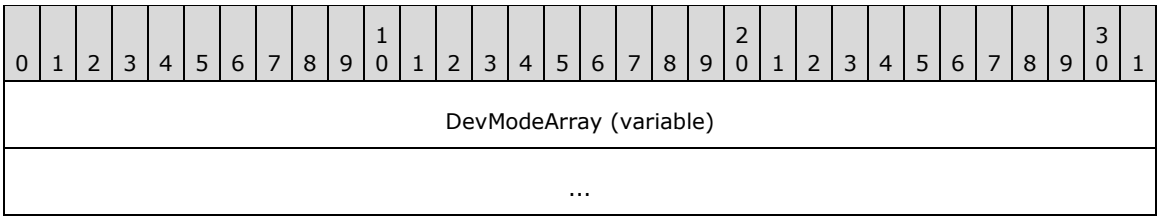
The _PRINTER_INFO_8 structure specifies printer information.<177> It is a custom-marshaled form of the PRINTER_INFO_8 (section 2.2.1.10.9) structure.



Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.

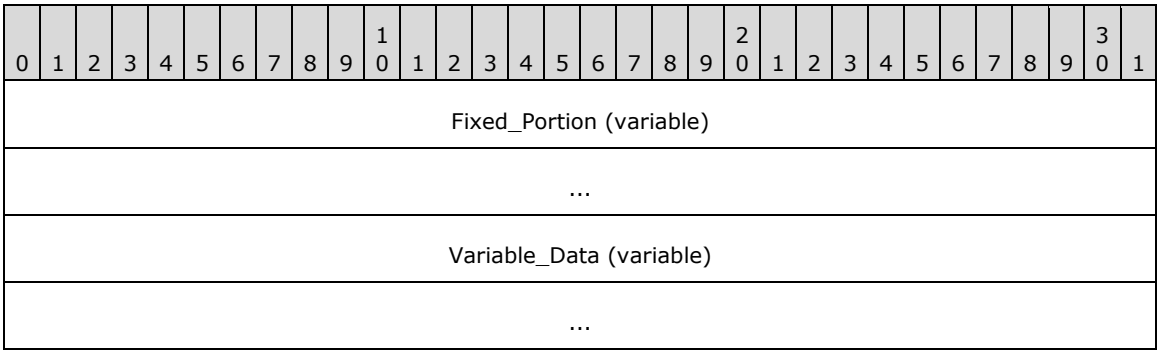


Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.

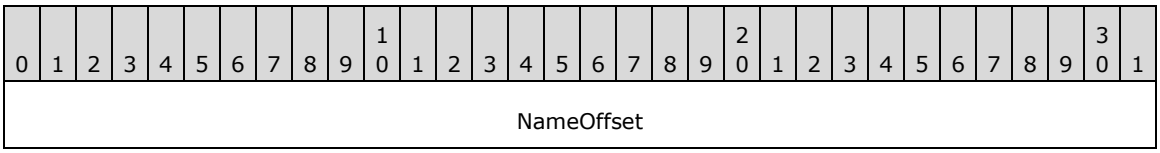


2.2.2.10 PRINTPROCESSOR_INFO_1

The PRINTPROCESSOR_INFO_1 structure specifies printer information.

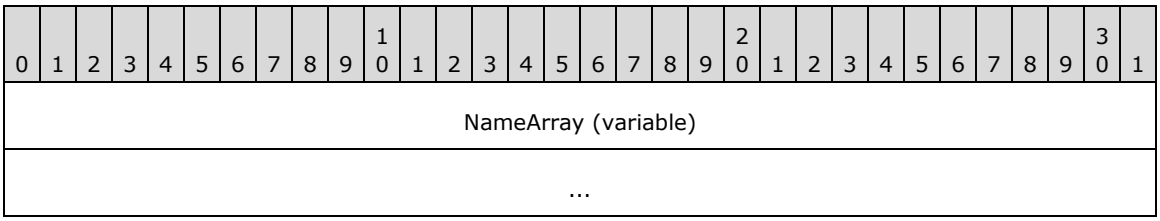


Fixed_Portion (variable): An array of one or more fixed-size fields, which are defined as follows.



NameOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **NameArray** member.

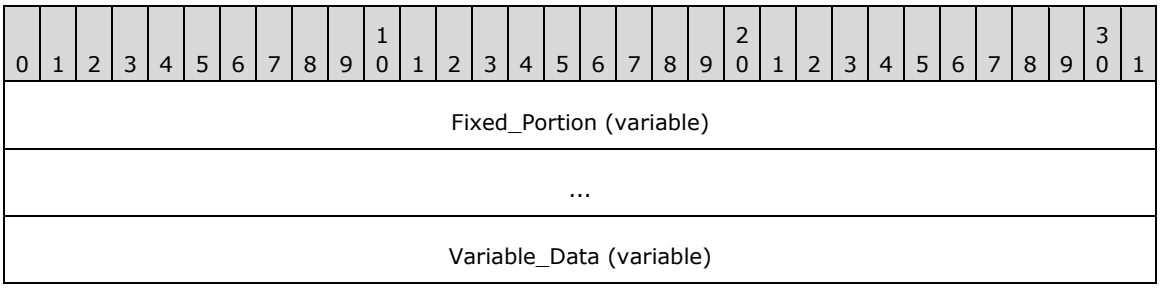
Variable_Data (variable): An array of zero or more optional, variable-size fields, which are defined as follows.



NameArray (variable): This member MUST contain a string that specifies the print processor name. The location of this buffer is determined by the value of the **NameOffset** member. For rules governing print processor names, see section 2.2.4.11.

2.2.2.11 PRINTER_ENUM_VALUES

The PRINTER_ENUM_VALUES structure specifies the value name, type, and data for a printer configuration value.<178>



...

Fixed_Portion (variable): An array of one or more groups of fixed-size fields, which are defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ValueNameOffset																															
cbValueName																															
dwType																															
DataOffset																															
cbData																															

ValueNameOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **ValueNameArray** member.

cbValueName (4 bytes): The value of this member MUST specify the size of the **ValueNameArray**, in bytes.

dwType (4 bytes): The value of this member MUST specify the data type of the data in the **DataArray** member. For a list of the possible type codes, see section 2.2.3.9. For rules governing **registry** type values, see section 2.2.4.15.

DataOffset (4 bytes): This member is a 32-bit unsigned integer that MUST specify the number of bytes from the start of the structure to the start of the **Data** member.

cbData (4 bytes): The value of this member MUST specify the number of bytes retrieved in the **DataArray** buffer.

Variable_Data (variable): An array of zero or more groups of optional, variable-size fields, which are defined as follows.

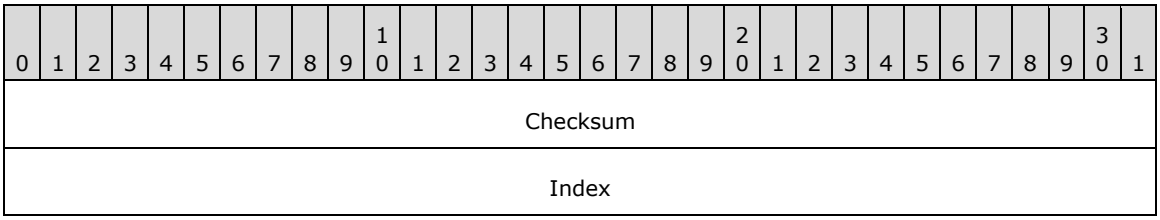
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ValueNameArray (variable)																															
...																															
DataArray (variable)																															
...																															

ValueNameArray (variable): This member MUST contain a string that specifies the value name. The location of this buffer is determined by the value of the **ValueNameOffset** member. For rules governing value names, see section 2.2.4.18.

DataArray (variable): This member MUST contain the data for the retrieved value.

2.2.2.12 UNIVERSAL_FONT_ID

The UNIVERSAL_FONT_ID structure identifies a font.



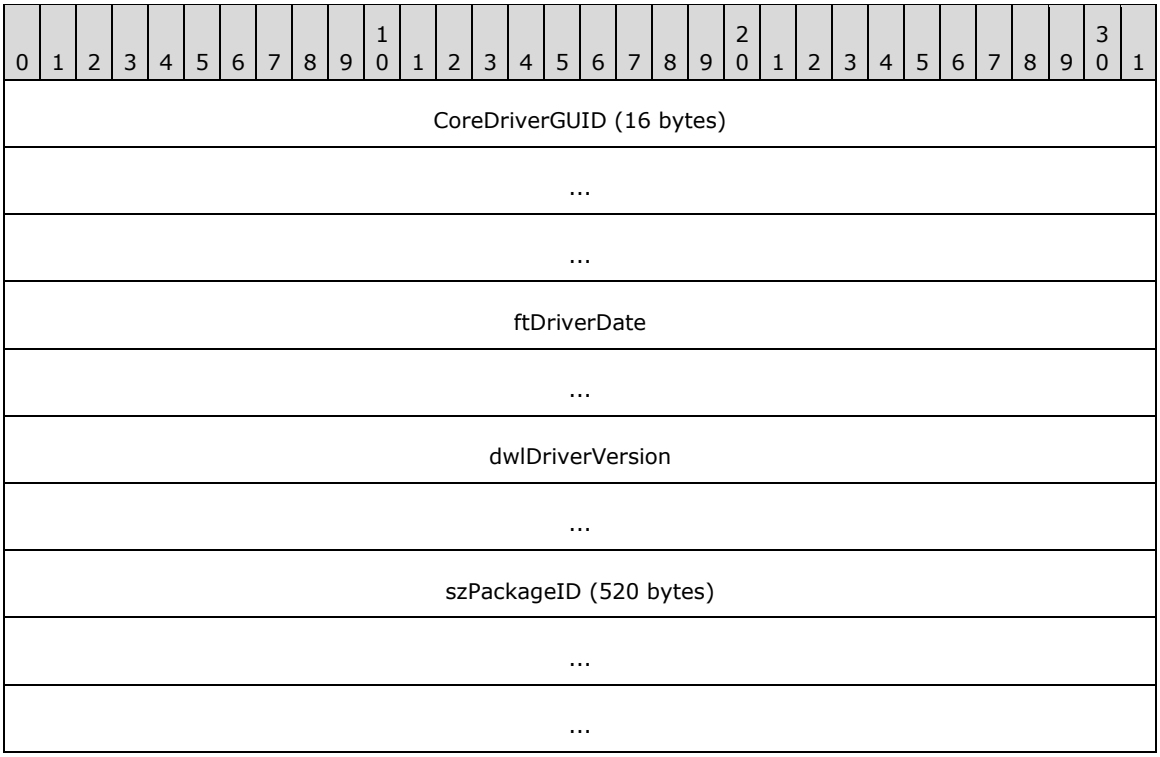
Checksum (4 bytes): A 32-bit unsigned integer that is the implementation-specific **checksum** of the font. <179>

Index (4 bytes): A 32-bit unsigned integer that is an index associated with the font. The meaning of this field is determined by the type of font.

Note: The UNIVERSAL_FONT_ID structure is equivalent to the **enhanced metafile format (EMF)** UniversalFontId object ([MS-EMF] section 2.2.27).

2.2.2.13 CORE_PRINTER_DRIVER

The CORE_PRINTER_DRIVER structure defines information that identifies a specific core printer driver. See the RpcGetCorePrinterDrivers (section 3.1.4.4.9) method for an example of its use.<180>



CoreDriverGUID (16 bytes): A GUID value that MUST uniquely identify the package.

ftDriverDate (8 bytes): A FILETIME value that MUST specify the date this package was published.

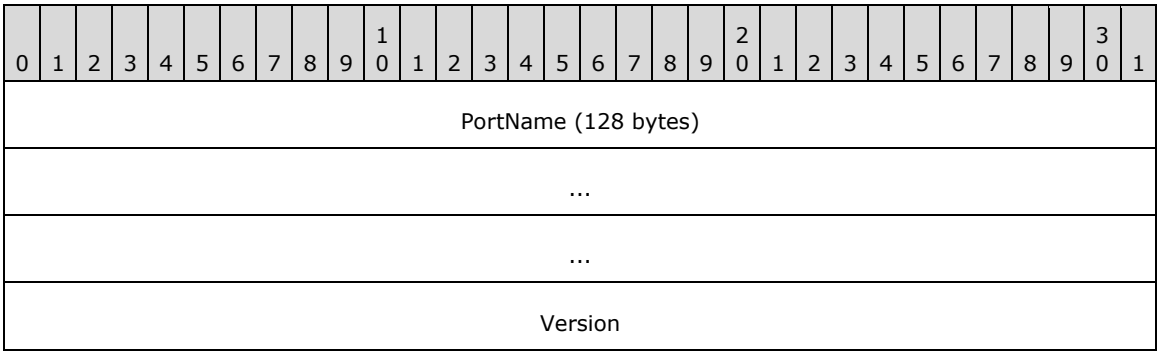
dwlDriverVersion (8 bytes): A 64-bit value that MUST specify the version of the core printer driver that MAY<181> be used to match the driver version in the driver installation control file.

szPackageID (520 bytes): A **string** that MUST specify the package name.

2.2.2.14 TCPMON Structures

2.2.2.14.1 CONFIG_INFO_DATA_1

The CONFIG_INFO_DATA_1 structure specifies printer configuration data.

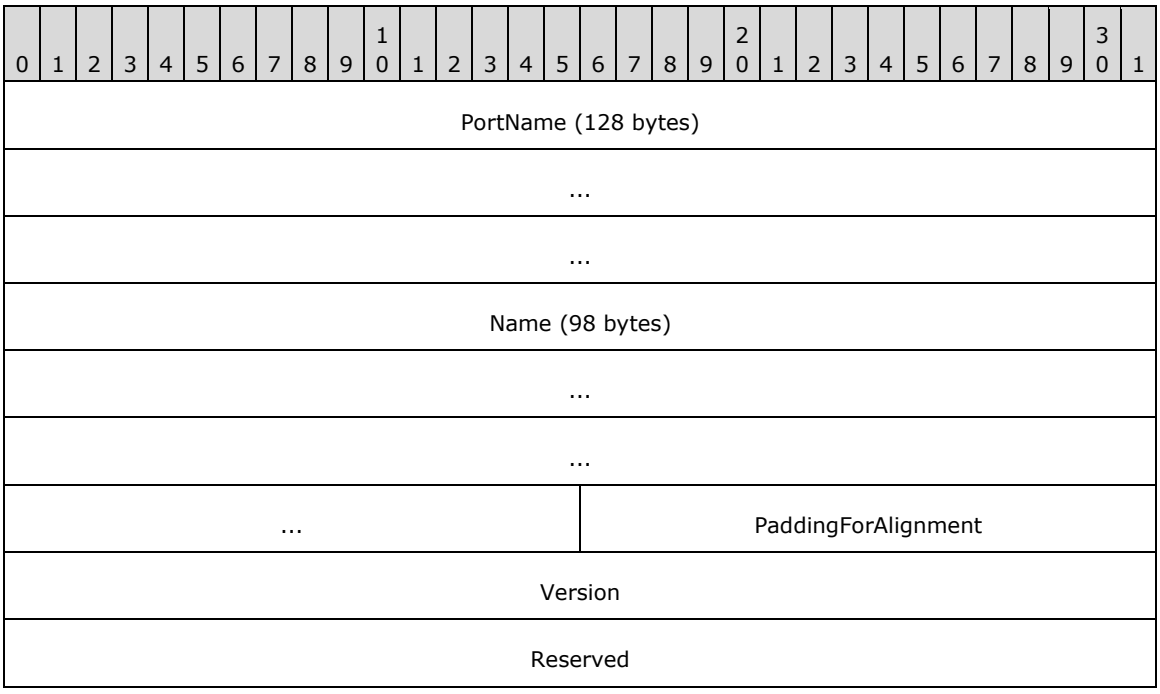


PortName (128 bytes): A null-terminated Unicode string, which is the name of the port to be queried for configuration information.

Version (4 bytes): A 32-bit unsigned integer that is the level of the **PORT_DATA** structure that will contain the configuration information. This value MUST be 0x00000001 or 0x00000002.

2.2.2.14.2 DELETE_PORT_DATA_1

The DELETE_PORT_DATA_1 structure specifies the port to be deleted.



PortName (128 bytes): A null-terminated Unicode string, which is the name of the port to be deleted.

Name (98 bytes): A null-terminated Unicode string, which is the server name for the port to be deleted.

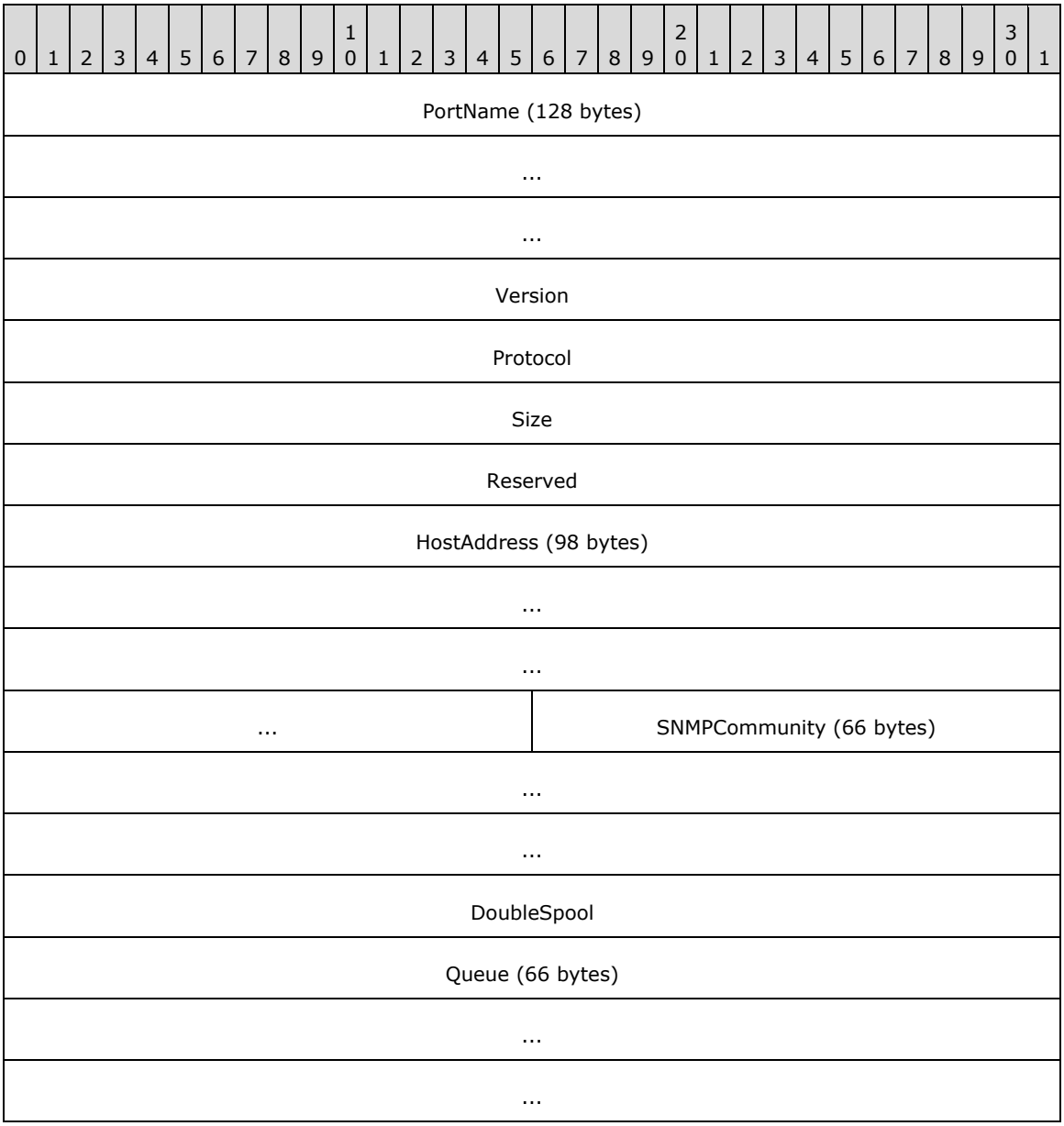
PaddingForAlignment (2 bytes): 2 bytes of padding to align **Version** field on a 4-byte boundary. The contents of this field MUST be ignored.

Version (4 bytes): A 32-bit unsigned integer that is the version of this structure. This value MUST be 0x00000001.

Reserved (4 bytes): A 32-bit unsigned integer that is set to zero.

2.2.2.14.3 PORT_DATA_1

The PORT_DATA_1 structure specifies **PORT_DATA** level 1 data.



...	IPAddress (32 bytes)
...	
...	
...	HardwareAddress (26 bytes)
...	
...	
DeviceType (514 bytes)	
...	
...	
...	PaddingForAlignment
PortNumber	
SNMPEnabled	
SNMPDevIndex	

PortName (128 bytes): A null-terminated Unicode string, which is the name of the port.

Version (4 bytes): A 32-bit unsigned integer that specifies the version number of the PORT_DATA_1 structure. This value MUST be 0x00000001.

Protocol (4 bytes): A 32-bit unsigned integer that specifies the protocol to use for the port. This value MUST be either **PROTOCOL_RAWTCP_TYPE** (0x00000001), indicating that the port expects RAW print data, or **PROTOCOL_LPR_TYPE** (0x00000002), indicating that the port expects to be driven as an LPR port.

Size (4 bytes): A 32-bit unsigned integer that specifies the size, in bytes, of the PORT_DATA_1 structure.

Reserved (4 bytes): A 32-bit unsigned integer that is set to zero.

HostAddress (98 bytes): A null-terminated Unicode string, which is the IP address or host name of the printer.

SNMPCommunity (66 bytes): A null-terminated Unicode string, which is the Simple Network Management Protocol (SNMP) [RFC1157] community name of the printer.

DoubleSpool (4 bytes): A 32-bit unsigned integer that, if nonzero, indicates that double spooling is enabled or, if zero, indicates that double spooling is disabled.

Queue (66 bytes): A null-terminated Unicode string, which is the LPR queue name.

IPAddress (32 bytes): A null-terminated Unicode string, which is the **IPv4** address of the printer.

HardwareAddress (26 bytes): A null-terminated Unicode string, which is the MAC address of the printer.

DeviceType (514 bytes): A null-terminated Unicode string, which is the generic SNMP device description (**object identifier (OID)** 1.3.6.1.2.1.1.1).

PaddingForAlignment (2 bytes): 2 bytes of padding to align the **PortNumber** field on a 4-byte boundary. The contents of this field MUST be ignored.

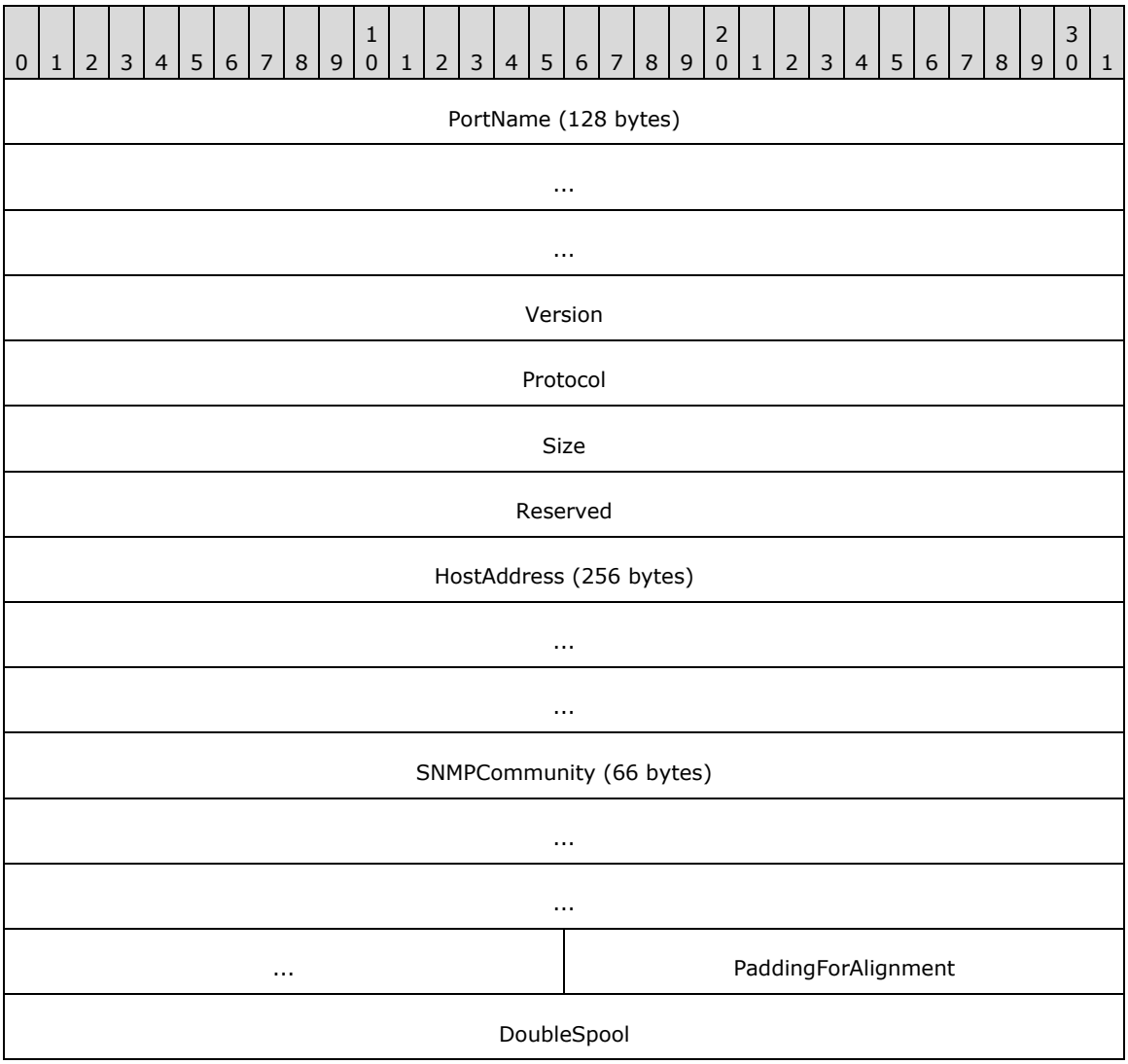
PortNumber (4 bytes): A 32-bit unsigned integer that is the port number of the device.

SNMPEnabled (4 bytes): A 32-bit unsigned integer that MUST be nonzero if the device supports SNMP.

SNMPDevIndex (4 bytes): A 32-bit unsigned integer that is the SNMP device index.

2.2.2.14.4 PORT_DATA_2

The PORT_DATA_2 structure specifies **PORT_DATA** level 2 data.<182>



Queue (66 bytes)	
...	
...	
...	DeviceType (514 bytes)
...	
...	
PortNumber	
SNMPEnabled	
SNMPDevIndex	
PortMonitorMibIndex	

PortName (128 bytes): A null-terminated Unicode string, which is the name of the port.

Version (4 bytes): A 32-bit unsigned integer that specifies the version number of the PORT_DATA_2 structure. This value MUST be 0x00000002.

Protocol (4 bytes): A 32-bit unsigned integer that specifies the protocol to use for the port. This value MUST be either **PROTOCOL_RAWTCP_TYPE** (0x00000001), indicating that the port expects RAW print data, or **PROTOCOL_LPR_TYPE** (0x00000002), indicating that the port expects to be driven as an LPR port.

Value	Meaning
PROTOCOL_RAWTCP_TYPE 0x00000001	The port expects RAW print data.
PROTOCOL_LPR_TYPE 0x00000002	The port expects to be driven as an LPR port.

Size (4 bytes): A 32-bit unsigned integer that specifies the size, in bytes, of the PORT_DATA_2 structure.

Reserved (4 bytes): A 32-bit unsigned integer that is set to zero.

HostAddress (256 bytes): A null-terminated Unicode string, which is the IP address or host name of the printer.

SNMPCommunity (66 bytes): A null-terminated Unicode string, which is the Simple Network Management Protocol (SNMP) [RFC1157] community name of the printer.

PaddingForAlignment (2 bytes): 2 bytes of padding to align the **DoubleSpool** field on a 4-byte boundary. The contents of this field MUST be ignored.

DoubleSpool (4 bytes): A 32-bit unsigned integer that, if nonzero, indicates double spooling is enabled or, if zero, indicates that double spooling is disabled.

Queue (66 bytes): A null-terminated Unicode string, which is the LPR queue name.

DeviceType (514 bytes): A null-terminated Unicode string, which is the generic SNMP device description (object identifier (OID) 1.3.6.1.2.1.1.1).

PortNumber (4 bytes): A 32-bit unsigned integer that is the port number of the device.

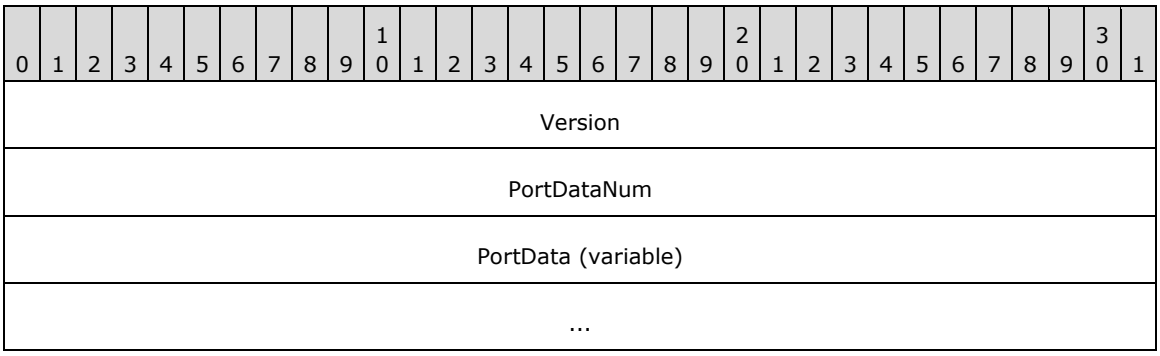
SNMPEnabled (4 bytes): A 32-bit unsigned integer that, if nonzero, indicates that the device supports SNMP.

SNMPDevIndex (4 bytes): A 32-bit unsigned integer that is the SNMP device index.

PortMonitorMibIndex (4 bytes): A 32-bit unsigned integer that specifies the index in the network devices PWG port Monitor MIB for the current **TCPMON** port. This index is used to query the IEEE 1284 device ID for the attached printer. For details, see [IEEE1284].

2.2.2.14.5 PORT_DATA_LIST_1

The PORT_DATA_LIST_1 structure specifies an array of PORT_DATA_2 structures.<183>



Version (4 bytes): A 32-bit unsigned integer that specifies the version number of the PORT_DATA_LIST_1 structure. This value MUST be 0x00000001.

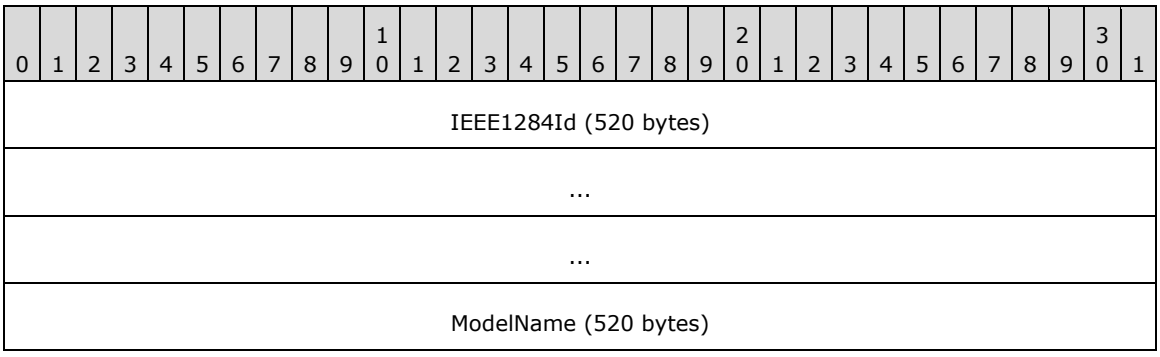
PortDataNum (4 bytes): A 32-bit unsigned integer that is the number of PORT_DATA_2 structures contained in the **PortData** array of this PORT_DATA_LIST_1 structure.

PortData (variable): An array of PORT_DATA_2 structures.

2.2.2.15 WSDMON Structures

2.2.2.15.1 WSD_DRIVER_DATA

The WSD_DRIVER_DATA structure holds information on the discovered printer.



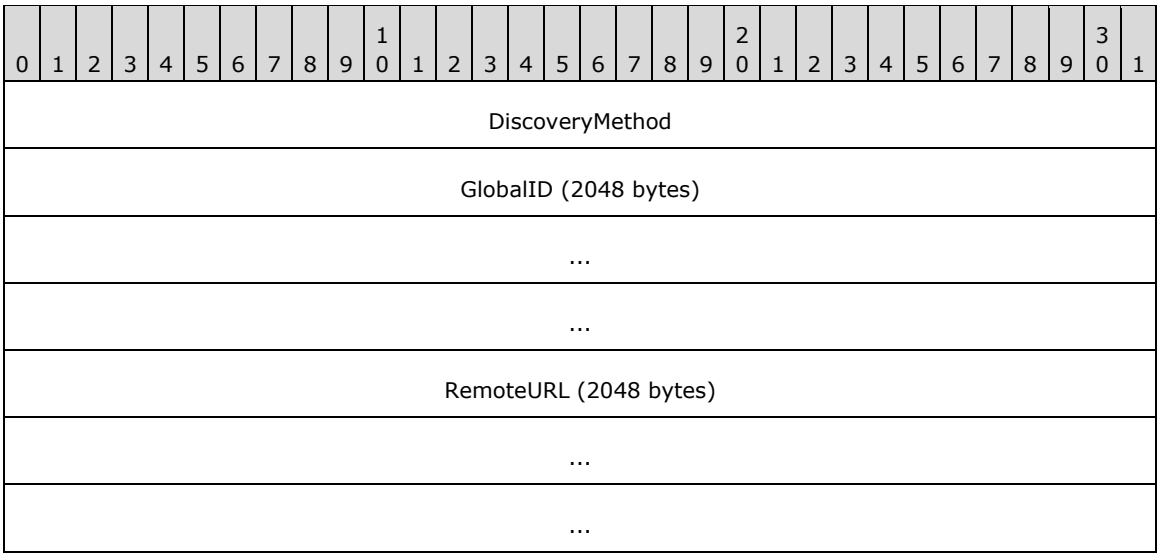
...
...

IEEE1284Id (520 bytes): A null-terminated Unicode string, which is the IEEE 1284 device ID for the discovered printer (for details, see [IEEE1284]). This can be used to generate the correct **Universal Plug and Play (UPnP)** ID for the printer driver.

ModelName (520 bytes): A null-terminated Unicode string, which is the name of the printer model discovered.

2.2.2.15.2 WSD_BACKUP_PORT_DATA

The WSD_BACKUP_PORT_DATA structure specifies information about the **Web Services for Devices (WSD)** backup port.



DiscoveryMethod (4 bytes): A 32-bit unsigned integer that specifies how the **WSD** port was initially discovered.

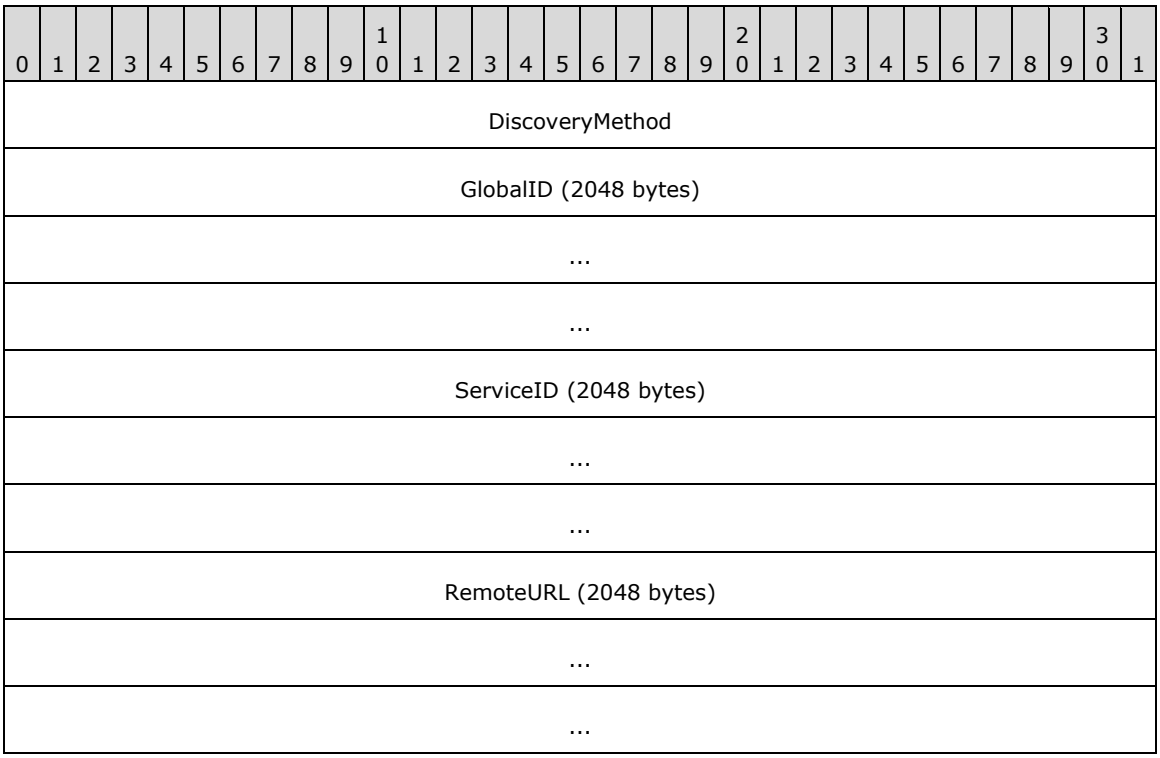
Value	Meaning
kMulticast 0x00000000	The WSD port was initially discovered by using **multicast discovery**.
kDirected 0x00000001	The WSD port was initially discovered by using **directed discovery**.

GlobalID (2048 bytes): A null-terminated **Unicode string** which, if **DiscoveryMethod** is kMulticast, specifies the **PKEY_PNPX_GlobalIdentify** of the device attached to the WSD port; otherwise, this field MUST be a NULL pointer.

RemoteURL (2048 bytes): A null-terminated Unicode string which, if **DiscoveryMethod** is kDirected, specifies the **URL** of the device attached to the WSD port; otherwise, this field MUST be a NULL pointer.

2.2.2.15.3 WSD_BACKUP_PORT_DATA_EX

The WSD_BACKUP_PORT_DATA_EX structure specifies information to restore for the Web Services for Devices (WSD) port.<184>



DiscoveryMethod (4 bytes): A 32-bit unsigned integer that specifies how the WSD port was initially discovered.

Value	Meaning
kMulticast 0x00000000	The WSD port was discovered by using multicast discovery.
kDirected 0x00000001	The WSD port was discovered by using directed discovery.

GlobalID (2048 bytes): A null-terminated Unicode string that specifies the **PKEY_PNPX_GlobalIdentify** of the device attached to the WSD port.

ServiceID (2048 bytes): A null-terminated Unicode string that specifies the **PKEY_PNPX_ServiceId** of the **WS-Print** printer service within the device represented by the WSD port.

RemoteURL (2048 bytes): A null-terminated Unicode string which, if **DiscoveryMethod** is kDirected, specifies the URL of the device attached to the WSD port; otherwise, this field MUST be NULL.

2.2.3 Constants

2.2.3.1 Access Values

The following table lists access level values that a print client can specify when opening a print job object, port object, printer object, or print server object. The type of object that an access value applies to is indicated in its name, as follows:

- Access values that are named starting with "JOB" are used for opening print job objects.
- Access values that are named starting with "PRINTER" are used for opening printer objects and port objects.
- Access values that are named starting with "SERVER" are used for opening print server objects.

The ACCESS_MASK data type ([MS-DTYP] section 2.4.3) specifies standard access rights and reserves a range of values for protocol-specific use. The following table defines printing-specific access rights in the reserved range and in combination with standard access rights. For more information concerning access rights, see [MS-AZOD] section 1.1.1.5.

For completeness, the table lists all defined access values and identifies those that MUST NOT be used with the Print System Remote Protocol.

Name/value	Description
JOB_ACCESS_ADMINISTER 0x00000010	Printing-specific authorization to cancel, pause, resume, or restart the job ([MS-DTYP] ACCESS_MASK Bit 27).
JOB_ACCESS_READ 0x00000020	Printing-specific read rights for the spool file ([MS-DTYP] ACCESS_MASK Bit 26).<185>
JOB_EXECUTE 0x00020010	Access rights for jobs combining **RC** (Read Control) of ACCESS_MASK with printing-specific **JOB_ACCESS_ADMINISTER**. This value MUST NOT be passed over the wire. If it is, the server SHOULD return ERROR_ACCESS_DENIED.
JOB_READ 0x00020020	Access rights for jobs combining **RC** (Read Control) of ACCESS_MASK with printing-specific **JOB_ACCESS_READ**.
JOB_WRITE 0x00020010	Access rights for jobs combining **RC** (Read Control) of ACCESS_MASK with printing-specific **JOB_ACCESS_ADMINISTER**. This value MUST NOT be passed over the wire. If it is, the server SHOULD return ERROR_ACCESS_DENIED.
JOB_ALL_ACCESS 0x000F0030	Access rights for printers to perform all administrative tasks and basic printing operations except SYNCHRONIZE ([MS-DTYP] ACCESS_MASK Bit 'SY'). Combines STANDARD_RIGHTS_REQUIRED (ACCESS_MASK Bits 'RC', 'DE', 'WD', 'WO'), JOB_ACCESS_ADMINISTER (ACCESS_MASK Bit 27), and JOB_ACCESS_READ (ACCESS_MASK Bit 26).
PRINTER_ACCESS_ADMINISTER 0x00000004	Printing-specific access rights for printers to perform administrative tasks ([MS-DTYP] ACCESS_MASK Bit 29).
PRINTER_ACCESS_USE 0x00000008	Printing-specific access rights for printers to perform basic printing operations ([MS-DTYP] ACCESS_MASK Bit 28).
PRINTER_ACCESS_MANAGE_LIMITED 0x00000040	Printing-specific access rights for printers to perform printer data management operations ([MS-DTYP] ACCESS_MASK Bit 25).<186>
PRINTER_ALL_ACCESS	Access rights for printers to perform all administrative tasks and basic

Name/value	Description
0x000F000C	printing operations except synchronization. Combines **WO** (Write Owner), **WD** (Write DACL), **RC** (Read Control), and **DE** (Delete) of ACCESS_MASK with printing-specific **PRINTER_ACCESS_ADMINISTER** and printing-specific **PRINTER_ACCESS_USE**.
PRINTER_EXECUTE 0x00020008	Access rights for printers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **PRINTER_ACCESS_USE**.
PRINTER_READ 0x00020008	Access rights for printers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **PRINTER_ACCESS_USE**.
PRINTER_WRITE 0x00020008	Access rights for printers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **PRINTER_ACCESS_USE**.
SERVER_ACCESS_ADMINISTER 0x00000001	Printing-specific access rights to administer print servers ([MS-DTYP] ACCESS_MASK Bit 31).
SERVER_ACCESS_ENUMERATE 0x00000002	Printing-specific access rights to enumerate print servers ([MS-DTYP] ACCESS_MASK Bit 30).
SERVER_ALL_ACCESS 0x000F0003	Access rights for print servers to perform all administrative tasks and basic printing operations except synchronization. Combines **WO** (Write Owner), **WD** (Write DACL), **RC** (Read Control), and **DE** (Delete) of ACCESS_MASK with printing-specific **SERVER_ACCESS_ADMINISTER** and printing-specific **SERVER_ACCESS_ENUMERATE**.
SERVER_EXECUTE 0x00020002	Access rights for print servers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **SERVER_ACCESS_ENUMERATE**.
SERVER_READ 0x00020002	Access rights for print servers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **SERVER_ACCESS_ENUMERATE**.
SERVER_WRITE 0x00020003	Access rights for print servers combining **RC** (Read Control) of ACCESS_MASK with printing-specific **SERVER_ACCESS_ADMINISTER** and printing-specific **SERVER_ACCESS_ENUMERATE**.
SPECIFIC_RIGHTS_ALL 0x0000FFFF	All specific rights.<187> This value MUST NOT be passed over the wire. It SHOULD only be used locally, as a mask to determine the protocol-specific subset of access values.
STANDARD_RIGHTS_ALL 0x001F0000	Combines **SY** (Synchronize), **WO** (Write Owner), **WD** (Write DACL), **RC** (Read Control), and **DE** (Delete) of ACCESS_MASK.<188> This value MUST NOT be passed over the wire. It SHOULD only be used locally, as a mask to determine the standard set of access values.
STANDARD_RIGHTS_EXECUTE 0x00020000	Standard rights, set to **RC** (Read Control) of ACCESS_MASK.
STANDARD_RIGHTS_READ 0x00020000	Standard read rights, set to **RC** (Read Control) of ACCESS_MASK.
STANDARD_RIGHTS_REQUIRED 0x000F0000	Standard rights, combines **WO** (Write Owner), **WD** (Write DACL), **RC** (Read Control), and **DE** (Delete) of ACCESS_MASK.
STANDARD_RIGHTS_WRITE 0x00020000	Standard write rights, set to **RC** (Read Control) of ACCESS_MASK.
SYNCHRONIZE	The right to use the object for synchronization, set to **SY** (Synchronize)

Name/value	Description
0x00100000	of ACCESS_MASK.<189> This value MUST NOT be passed over the wire. If it is, the server SHOULD return ERROR_ACCESS_DENIED.
DELETE 0x00010000	The right to delete an object, set to **DE** (Delete) of ACCESS_MASK.
READ_CONTROL 0x00020000	The right to read the information in the object's **security descriptor**, not including the information in the **system access control list (SACL)**, set to **RC** (Read Control) of ACCESS_MASK.
WRITE_DAC 0x00040000	The right to modify the **discretionary access control list (DACL)** in the object's security descriptor, set to **WD** (Write DACL) of ACCESS_MASK.
WRITE_OWNER 0x00080000	The right to change the owner in the object's security descriptor, set to **WO** (Write Owner) of ACCESS_MASK.
GENERIC_READ 0x80000000	**GR** (Generic Read) of ACCESS_MASK: For server object, access is mapped to **SERVER_READ**. For printer object, access is mapped to **PRINTER_READ**. For print job, access is mapped to **JOB_READ**.
GENERIC_WRITE 0x40000000	**GW** (Generic Write) of ACCESS_MASK: For server object, access is mapped to **SERVER_WRITE**. For printer object, access is mapped to **PRINTER_WRITE**. For print job, access is mapped to **JOB_WRITE**.
GENERIC_EXECUTE 0x20000000	**GX** (Generic Execute) of ACCESS_MASK: For server object, access is mapped to **SERVER_EXECUTE**. For printer object, access is mapped to **PRINTER_EXECUTE**. For print job, access is mapped to **JOB_EXECUTE**.
GENERIC_ALL 0x10000000	**GA** (Generic All) of ACCESS_MASK: For server object, access is mapped to **SERVER_ALL_ACCESS**. For printer object, access is mapped to **PRINTER_ALL_ACCESS**. For print job, access is mapped to **JOB_ALL_ACCESS**.

2.2.3.2 Change Notification Flags

Change Notification Flags specify change notification information and options.<190>

The following bit flag is set by a print server in the **Flags** member of an RPC_V2_NOTIFY_INFO structure (section 2.2.1.13.3).

Name/value	Description
PRINTER_NOTIFY_INFO_DISCARDED 0x00000001	An overflow or error has occurred, and notifications have been lost. The print server MUST NOT send additional notifications until the client has made a call to RpcRouterRefreshPrinterChangeNotification (section 3.1.4.10.5).

The following bit flags are set by a print client in the variable pointed to by the *pdwResult* parameter in a call to RpcRouterReplyPrinterEx (section 3.2.4.1.4). They are used to indicate the results of processing the state of an RPC_V2_NOTIFY_INFO structure.

These flags can be combined to specify multiple results.

Name/Value	Description
PRINTER_NOTIFY_INFO_DISCARDNOTED 0x00010000	The client acknowledges receiving and processing the **PRINTER_NOTIFY_INFO_DISCARDED** notification.
PRINTER_NOTIFY_INFO_COLORMISMATCH 0x00080000	The value of the *dwColor* parameter in a call to RpcRouterReplyPrinterEx does not match the value the client previously passed to the server in a call to RpcRouterRefreshPrinterChangeNotification.

The following bit flag is set by a print client in the **Reserved** member of an RPC_V2_NOTIFY_OPTIONS structure (section 2.2.1.13.1).

Name/value	Description
PRINTER_NOTIFY_OPTIONS_REFRESH 0x00000001	Refreshed data is requested from the server for all monitored members.

2.2.3.3 Job Notification Values

Job Notification Values specify types of changes in the **Data** member of an RPC_V2_NOTIFY_INFO_DATA (section 2.2.1.13.4) structure.<191>**Reserved** members are specified by Notification Data Type Values (section 2.2.3.5), and **String** values are specified in JOB_INFO (section 2.2.2.6) structures. The print server passes these notification values to a print client using RpcRouterReplyPrinterEx (section 3.2.4.1.4).

Constant/value	Description
JOB_NOTIFY_FIELD_PRINTER_NAME 0x0000	Specifies that the printer name for the print job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new printer name (see **pPrinterName** in JOB_INFO structures).
JOB_NOTIFY_FIELD_MACHINE_NAME 0x0001	Specifies that the server name for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new server name (see **pMachineName** in JOB_INFO structures).
JOB_NOTIFY_FIELD_PORT_NAME 0x0002	Specifies that the port for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains a string specifying the new port the job is printed on (see **pPortName** in JOB_INFO structures).
JOB_NOTIFY_FIELD_USER_NAME 0x0003	Specifies that the user name for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new user name (see **pUserName** in JOB_INFO structures).
JOB_NOTIFY_FIELD_NOTIFY_NAME 0x0004	Specifies that the notify name for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new notify name (see **pNotifyName** in JOB_INFO

Constant/value	Description
	structures).
JOB_NOTIFY_FIELD_DATATYPE 0x0005	Specifies that the default data type for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new default data type (see **pDatatype** in JOB_INFO structures).
JOB_NOTIFY_FIELD_PRINT_PROCESSOR 0x0006	Specifies that the print processor associated with the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new print processor name (see **pPrintProcessor** in JOB_INFO structures).
JOB_NOTIFY_FIELD_PARAMETERS 0x0007	Specifies that the default print processor parameters for the job have changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new print processor parameters value (see **pParameters** in JOB_INFO structures).
JOB_NOTIFY_FIELD_DRIVER_NAME 0x0008	Specifies that the printer driver for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new printer driver name (see **pDriverName** in JOB_INFO structures).
JOB_NOTIFY_FIELD_DEVMODE 0x0009	Specifies that the default DEVMODE (section 2.2.2.1) structure for the job has changed. **Reserved** contains **TABLE_DEVMODE**, and **Data.DevMode** contains the new **DEVMODE** structure (see **pDevMode** in JOB_INFO structures).
JOB_NOTIFY_FIELD_STATUS 0x000A	Specifies that the status for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Status**.
JOB_NOTIFY_FIELD_STATUS_STRING 0x000B	The textual representation for the job status has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new status string (see **pStatus** in JOB_INFO structures).
JOB_NOTIFY_FIELD_SECURITY_DESCRIPTOR 0x000C	Security descriptor for the job has changed. **Reserved** contains **TABLE_SECURITYDESCRIPTOR**, and **Data.SecurityDescriptor** contains the new security descriptor (see **pSecurityDescriptor** in JOB_INFO structures).
JOB_NOTIFY_FIELD_DOCUMENT 0x000D	Specifies that the document name for the job has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new document name (see **pDocument** in JOB_INFO structures).
JOB_NOTIFY_FIELD_PRIORITY 0x000E	Specifies that the current priority for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Priority** value.
JOB_NOTIFY_FIELD_POSITION 0x000F	Specifies that the position in the queue for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Position** value.
JOB_NOTIFY_FIELD_SUBMITTED 0x0010	Specifies that the submitted time for the job has changed. **Reserved** contains **TABLE_TIME**, and **Data.SystemTime** contains the new **Submitted** value.

Constant/value	Description
JOB_NOTIFY_FIELD_START_TIME 0x0011	Specifies that the earliest start time for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **StartTime** value.
JOB_NOTIFY_FIELD_UNTIL_TIME 0x0012	Specifies that the latest print time for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **UntilTime** value.
JOB_NOTIFY_FIELD_TIME 0x0013	Specifies that the total print time for the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Time** value.
JOB_NOTIFY_FIELD_TOTAL_PAGES 0x0014	Specifies that the total number of pages of the job has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **cTotalPages** value.
JOB_NOTIFY_FIELD_PAGES_PRINTED 0x0015	Specifies that the number of pages that have been printed has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **PagesPrinted** value.
JOB_NOTIFY_FIELD_TOTAL_BYTES 0x0016	Specifies that the total number of bytes of the job has changed. **Reserved** contains **TABLE_DWORD**, **Data.dwData[0]** contains the new **Size**, and **Data.dwData[1]** contains the new **SizeHigh** value.
JOB_NOTIFY_FIELD_BYTES_PRINTED 0x0017	Specifies that the total number of bytes that have been printed has changed. **Reserved** contains **TABLE_DWORD**, **Data.dwData[0]** contains the 32 low-order bits of the number of bytes printed, and **Data.dwData[1]** contains the 32 high-order bits of the number of bytes printed.

2.2.3.4 Server Notification Values

Server notification values specify types of changes in the **Data** member of an RPC_V2_NOTIFY_INFO_DATA structure (section 2.2.1.13.4). **Reserved** members are specified by notification data type values (section 2.2.3.5). The print server passes these notification values to a print client using RpcRouterReplyPrinterEx (section 3.2.4.1.4).

Constant/value	Description
SERVER_NOTIFY_FIELD_PRINT_DRIVER_ISOLATION_GROUP 0x0000	Specifies that the printer driver isolation group setting for the print server has changed. **Reserved** contains **TABLE_STRING** and **Data.String** contains the updated value of the print server's "PrintDriverIsolationGroups" configuration data. For details, see Server Handle Key Values (section 2.2.3.10). <192>

2.2.3.5 Notification Data Type Values

Constant/value	Description
TABLE_DWORD 0x0001	The notification data member MUST contain a two-DWORD array.
TABLE_STRING 0x0002	The notification data member MUST contain a string.
TABLE_DEVMODE 0x0003	The notification data member MUST be a non-NULL pointer to a DEVMODE_CONTAINER structure (section 2.2.1.2.1).
TABLE_TIME 0x0004	The notification data member MUST contain a SYSTEMTIME_CONTAINER structure (section 2.2.1.2.16).
TABLE_SECURITYDESCRIPTOR 0x0005	The notification data member MUST contain a SECURITY_CONTAINER structure (section 2.2.1.2.13).

2.2.3.6 Printer Change Flags

Printer Change Flags specify changes to a printer configuration. These flags can be combined to specify multiple changes.

2.2.3.6.1 Printer Change Flags for Use with a Printer Handle

These flags are for use with a printer handle, and all other bits MUST be ignored when used with a printer handle:

Constant/value	Description
PRINTER_CHANGE_SET_PRINTER 0x00000002	Printer object properties were configured.
PRINTER_CHANGE_DELETE_PRINTER 0x00000004	A printer object was deleted.
PRINTER_CHANGE_PRINTER 0x000000FF	A printer object changed in some way. A client can use this value to indicate that it accepts all change notifications regarding printers. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_ADD_JOB 0x00000100	A print job was added.
PRINTER_CHANGE_SET_JOB 0x00000200	Print job properties were configured.
PRINTER_CHANGE_DELETE_JOB 0x00000400	A print job was deleted.
PRINTER_CHANGE_WRITE_JOB 0x00000800	A print job was written.
PRINTER_CHANGE_JOB 0x0000FF00	A print job changed in some way. A client can use this value to indicate that it accepts all change

Constant/value	Description
	notifications regarding print jobs. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_SET_PRINTER_DRIVER 0x20000000	A printer driver was specified.<193>
PRINTER_CHANGE_TIMEOUT 0x80000000	Returned by RpcWaitForPrinterChange (section 3.1.4.10.1) if the implementation-specific timeout has expired.
PRINTER_CHANGE_ALL 0x7777FFFF	A change was made to one or more printer-related objects, including print job, form, port, **processor**, or printer driver, or to the printer object itself. A client can use this value to indicate that it is interested in all change notifications. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.<194>
PRINTER_CHANGE_ALL_2 0x7F77FFFF	Identical with PRINTER_CHANGE_ALL (0x7777FFFF).<195>

For more information about the rules governing printer change values, see section 2.2.4.13.

2.2.3.6.2 Printer Change Flags for Use with a Server Handle

These flags are for use with a server handle, and all other bits MUST be ignored when used with a server handle:

Constant/value	Description
PRINTER_CHANGE_ADD_PRINTER_DRIVER 0x10000000	A printer driver was added.
PRINTER_CHANGE_DELETE_PRINTER_DRIVER 0x40000000	A printer driver was deleted.
PRINTER_CHANGE_PRINTER_DRIVER 0x70000000	A printer driver was changed in some way. A client can use this to indicate that it accepts all change notifications regarding printer drivers. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_ADD_FORM 0x00010000	A form was added.
PRINTER_CHANGE_DELETE_FORM 0x00040000	A form was deleted.
PRINTER_CHANGE_SET_FORM 0x00020000	Form properties were configured.
PRINTER_CHANGE_FORM 0x00070000	A form was changed in some way. A client can use this to indicate that it accepts all change notifications regarding forms. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_ADD_PORT 0x00100000	A port was added.

Constant/value	Description
PRINTER_CHANGE_CONFIGURE_PORT 0x00200000	Port properties were configured.
PRINTER_CHANGE_DELETE_PORT 0x00400000	A port was deleted.
PRINTER_CHANGE_PORT 0x00700000	A port was changed in some way. A client can use this to indicate that it accepts all change notifications regarding ports. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_ADD_PRINT_PROCESSOR 0x01000000	A print processor was added.
PRINTER_CHANGE_DELETE_PRINT_PROCESSOR 0x04000000	A print processor was deleted.
PRINTER_CHANGE_PRINT_PROCESSOR 0x07000000	The properties for a print processor were updated. A client can use this to indicate that it accepts all change notifications regarding print processors. The server SHOULD set only the individual flags corresponding to the changes that actually occurred.
PRINTER_CHANGE_ADD_PRINTER 0x00000001	A printer object was added.
PRINTER_CHANGE_FAILED_CONNECTION_PRINTER 0x00000008	A connection to a printer object failed.<196>
PRINTER_CHANGE_SERVER 0x08000000	A change was made to one or more of the monitored server configuration settings.<197>

For more information about the rules governing printer change values, see section 2.2.4.13.

2.2.3.7 Printer Enumeration Flags

Printer Enumeration Flags specify types of printers to enumerate. These flags can be combined to specify multiple printer types.

Constant/value	Description
PRINTER_ENUM_LOCAL 0x00000002	Enumerate local printer objects.
PRINTER_ENUM_CONNECTIONS 0x00000004	Enumerate printer connections previously added through RpcAddPerMachineConnection.
PRINTER_ENUM_NAME 0x00000008	Enumerate printers on the print server, network **domain**, or a specific **print provider**.
PRINTER_ENUM_REMOTE 0x00000010	Enumerate network printers and other print servers that are in the same domain as the print server.
PRINTER_ENUM_SHARED 0x00000020	Only enumerate printers with the shared attribute set. This flag MUST be combined with one or more of the other flags.

Constant/value	Description
PRINTER_ENUM_NETWORK 0x00000040	Enumerate network printers that are in the same domain as the print server.
PRINTER_ENUM_EXPAND 0x00004000	Indicates that the printer object contains further enumerable child objects. When a server enumerates print servers (see RpcEnumPrinters (section 3.1.4.2.1) the server can set this bit for each enumerated server whose name matches the server's **domain name**.
PRINTER_ENUM_CONTAINER 0x00008000	Indicates that the printer object is capable of containing enumerable objects. One such object is a print provider, which is a print server that contains printers.
PRINTER_ENUM_ICON1 0x00010000	Indicates that, where appropriate, an application treats the printer object as a top-level network name, such as Windows network. A GUI application can <198> choose to display an icon of choice for this type of object.
PRINTER_ENUM_ICON2 0x00020000	Indicates that, where appropriate, an application treats an object as a network domain name. A GUI application can<199> choose to display an icon of choice for this type of object.
PRINTER_ENUM_ICON3 0x00040000	Indicates that, where appropriate, an application treats an object as a print server. A GUI application can<200> choose to display an icon of choice for this type of object.
PRINTER_ENUM_ICON8 0x00800000	Indicates that, where appropriate, an application treats an object as a print server. A GUI application can<201> choose to display an icon of choice for this type of object.
PRINTER_ENUM_HIDE 0x01000000	Indicates that an application cannot display the printer object.<202>

2.2.3.8 Printer Notification Values

Printer notification values specify types of changes in printer data and/or state for which print clients can be notified.

The following constants can be used in the *fdwOptions* parameters of the RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3) and RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4) methods. They specify the categories of printers for which change notifications are returned.

Name/value	Description
0x00000000	Return notifications for 2D printers only.
PRINTER_NOTIFY_CATEGORY_ALL 0x00010000	Return notifications for both 2D and 3D printers.<203><204>
PRINTER_NOTIFY_CATEGORY_3D 0x00020000	Return notifications for 3D printers only.<205>

The following constants can be used in the **Data** member of an RPC_V2_NOTIFY_INFO_DATA (section 2.2.1.13.4) structure.<206> **Reserved** members are specified by Notification Data Type Values (section 2.2.3.5), and **String** values are specified in PRINTER_INFO (section 2.2.1.10) structures. The print server passes these notification values to a print client using RpcRouterReplyPrinterEx (section 3.2.4.1.4).

Name/value	Description
PRINTER_NOTIFY_FIELD_ATTRIBUTES 0x000D	Specifies that printer attributes have changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Attributes** value.
PRINTER_NOTIFY_FIELD_AVERAGE_PPM 0x0015	Specifies that the average pages per minute for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **AveragePPM** value.
PRINTER_NOTIFY_FIELD_BRANCH_OFFICE_PRINTING 0x001C	Specifies that the EnableBranchOfficePrinting printer data value (section 2.2.3.11) for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new value for the EnableBranchOfficePrinting printer data value.<207>
PRINTER_NOTIFY_FIELD_BYTES_PRINTED 0x0019	Specifies that the number of bytes that have been printed has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **cTotalBytes** value.
PRINTER_NOTIFY_FIELD_CJOBS 0x0014	Specifies that the number of print jobs that are queued for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **cJobs** value.
PRINTER_NOTIFY_FIELD_COMMENT 0x0005	Specifies that the printer comment has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new comment (see **pComment** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_DATATYPE 0x000B	Specifies that the printer default data type has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new comment (see **pDatatype** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_DEFAULT_PRIORITY 0x000F	Specifies that the default priority for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **DefaultPriority**.
PRINTER_NOTIFY_FIELD_DEVMODE 0x0007	Specifies that the default _DEVMODE (section 2.2.2.1) structure for the printer has changed. **Reserved** contains **TABLE_DEVMODE**, and **Data.DevMode** contains the new **DEVMODE** structure (see **pDevMode** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_DRIVER_NAME 0x0004	Specifies that the printer driver for the printer has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new printer driver name (see **pDriverName** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_LOCATION 0x0006	Specifies that the printer location has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new location description value (see **pLocation** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_OBJECT_GUID 0x001A	Specifies that the printer object GUID has changed.<208>

Name/value	Description
	Reserved contains **TABLE_STRING**, and **Data.String** contains the new printer GUID value (see **pszObjectGUID** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_PAGES_PRINTED 0x0017	Specifies that the number of pages that have been printed for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **PagesPrinted** value.
PRINTER_NOTIFY_FIELD_PARAMETERS 0x000A	Specifies that the default print processor parameters for the printer have changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new print processor parameters value (see **pParameters** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_PORT_NAME 0x0003	Specifies that the default port name for the printer has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new port name value (see **pPortName** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_PRINTER_NAME 0x0001	Specifies that the printer name has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new printer name value (see **pPrinterName** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_PRINT_PROCESSOR 0x0009	Specifies that the print processor associated with the printer has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new print processor name value (see **pPrintProcessor** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_PRIORITY 0x000E	Specifies that the current priority for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Priority** value.
PRINTER_NOTIFY_FIELD_SECURITY_DESCRIPTOR 0x000C	Specifies that the security descriptor for the printer has changed. **Reserved** contains **TABLE_SECURITYDESCRIPTOR**, and **Data.SecurityDescriptor** contains the new security descriptor value (see **pSecurityDescriptor** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_SEPFILE 0x0008	Specifies that the separator page for the printer has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new separator page value (see **pSepFile** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_SERVER_NAME 0x0000	Specifies that the server name for the printer has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new server name value (see **pServerName** in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_SHARE_NAME 0x0002	Specifies that the printer share name has changed. **Reserved** contains **TABLE_STRING**, and **Data.String** contains the new share name value (see **pShareName**

Name/value	Description
	in PRINTER_INFO structures).
PRINTER_NOTIFY_FIELD_START_TIME 0x0010	Specifies that the earliest start time for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **StartTime** value.
PRINTER_NOTIFY_FIELD_STATUS 0x0012	Specifies that the status for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **Status** value.
PRINTER_NOTIFY_FIELD_TOTAL_BYTES 0x0018	Specifies that the total number of bytes that have been printed on the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **cTotalBytes** value.
PRINTER_NOTIFY_FIELD_TOTAL_PAGES 0x0016	Specifies that the total number of pages that have been printed on the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **cTotalPagesPrinted** value.
PRINTER_NOTIFY_FIELD_UNTIL_TIME 0x0011	Specifies that the latest print time for the printer has changed. **Reserved** contains **TABLE_DWORD**, and **Data.dwData[0]** contains the new **UntilTime** value.

2.2.3.9 Registry Type Values

Registry type name/value	Description
REG_NONE 0x00000000	No value type is defined.
REG_SZ 0x00000001	A string.
REG_EXPAND_SZ 0x00000002	A string that can contain unexpanded references to environment variables, for example, "%PATH%".
REG_BINARY 0x00000003	Binary data in any form.
REG_DWORD 0x00000004	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN 0x00000004	A 32-bit number in little-endian format; equivalent to REG_DWORD.
REG_DWORD_BIG_ENDIAN 0x00000005	A 32-bit number in **big-endian** format.
REG_LINK 0x00000006	Symbolic link to a registry key.
REG_MULTI_SZ	A REG_MULTI_SZ structure as specified in [MS-RRP] section 2.2.6.

Registry type name/value	Description
0x00000007	
REG_RESOURCE_LIST 0x00000008	A **device driver** resource list.
REG_QWORD 0x0000000B	A 64-bit number.
REG_QWORD_LITTLE_ENDIAN 0x0000000B	A 64-bit number in little-endian format; equivalent to REG_QWORD.

2.2.3.10 Server Handle Key Values

Server Handle Key Values are used to store printer configuration data. The values named in the following table MUST be supported by print servers as follows:

- In a call to RpcGetPrinterData or RpcGetPrinterDataEx, the *pValueName* parameter identifies the data that MUST be returned in the *pData* parameter. The value pointed to by *pValueName* MUST be one of the strings specified in the "Server handle key value name" column.

  The specified registry type values are defined in section 2.2.3.9.

- If the "Read-write" column is checked, a print server SHOULD keep track of the value set by a call to RpcSetPrinterData or RpcSetPrinterDataEx in the *pData* parameter, and it SHOULD return the same value in a subsequent call to RpcGetPrinterData or RpcGetPrinterDataEx.

- The "Meaning" column describes the printer configuration data that is associated with the server handle key value name, and in some cases it specifies print server behavior. If the "Behavior optional" column is checked, that behavior is implementation-specific and not mandatory.

Server handle key value name/registry type	Read-write	Behavior optional	Meaning
"Architecture" REG_SZ			A string that specifies the OS Environment Name (section 2.2.4.4), as determined by the processor architecture.
"BeepEnabled" REG_DWORD	X	X	If this value is nonzero, the print server issues a beep sound on the local console.
"DefaultSpoolDirectory" REG_SZ	X		The **UNC** path for the directory in which the print server stores spooled print jobs.
"DNSMachineName" REG_SZ			**Domain Name System (DNS)** computer name.
"DsPresent" REG_DWORD			0x0001 if the print server is joined to a domain with directory services, zero if not.
"DsPresentForUser" REG_DWORD			0x0001 if the user is logged on to a domain with directory services, zero if not.
"EventLog" REG_DWORD	X	X	A bit mask specifying which events for a print server to log in its internal event log. It is a bitwise OR of zero or more of the Event Log

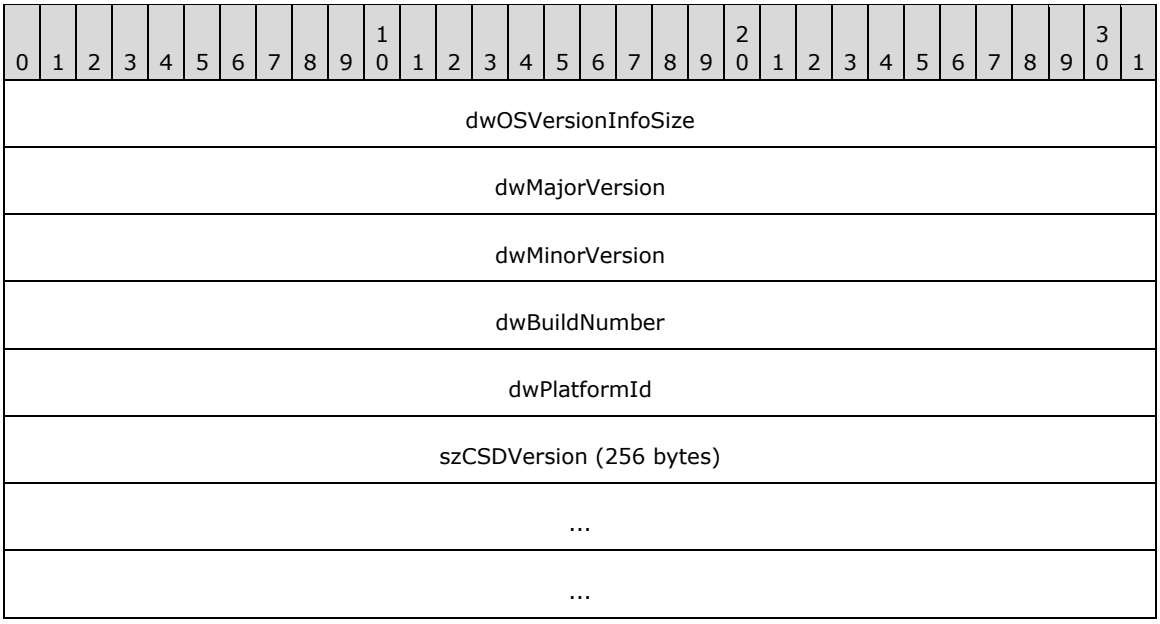
Server handle key value name/registry type	Read-write	Behavior optional	Meaning
			Flags (section 2.2.3.10.4).
"MajorVersion" REG_DWORD			The major OS version. See **dwMajorVersion** in SPLCLIENT_INFO Members (section 2.2.1.3.7) for details.
"MinorVersion" REG_DWORD			The minor OS version. See **dwMinorVersion** in SPLCLIENT_INFO Members (section 2.2.1.3.7) for details.
"NetPopup" REG_DWORD	X	X	If this value is nonzero, the print server MAY alert the print client of the status of a print job.<209>
"NetPopupToComputer" REG_DWORD	X	X	If this value is nonzero, the print server MAY alert the print client of changes to the status of a print job.<210>
"OSVersion" REG_BINARY			OS version information, in the form of an OSVERSIONINFO structure (section 2.2.3.10.1).
"OSVersionEx" REG_BINARY			Extended OS version information, in the form of an OSVERSIONINFOEX structure (section 2.2.3.10.2).
"PortThreadPriority" REG_DWORD	X	X	The current priority of the thread on which the print server sends data to printers. This value MUST be one of the Thread Priority values (section 2.2.3.10.6) constants. This key name MAY be used remotely.<211>
"PortThreadPriorityDefault" REG_DWORD		X	The default priority of the thread on which the print server sends data to printers. This value MUST be one of the Thread Priority values constants. This key name MAY be used remotely.<212>
"RemoteFax" REG_DWORD			0x0001 if the Microsoft Fax printer driver supports remote clients, zero otherwise. If this value is zero, print client connections to the Microsoft Fax printer driver SHOULD NOT be made.
"RestartJobOnPoolEnabled" REG_DWORD	X		A nonzero value indicates that **SPLREG_RESTART_JOB_ON_POOL_ERROR** is enabled. This key name MAY be used remotely.<213>
"RestartJobOnPoolError" REG_DWORD	X		The minimum time, in seconds, when a print job is restarted on another port after an error occurs. This key name MAY be used remotely.<214>
"RetryPopup" REG_DWORD	X	X	If this value is nonzero, the print server MAY offer the print client an option to retry a print job.<215> This key name MAY be used remotely.<216>
"SchedulerThreadPriority"	X	X	The current priority of the thread on which the print server schedules jobs for sending to

Server handle key value name/registry type	Read-write	Behavior optional	Meaning
REG_DWORD			printers. This value MUST be one of the Thread Priority values constants. This key name MAY be used remotely.<217>
"SchedulerThreadPriorityDefault" REG_DWORD		X	The default priority of the thread on which the print server schedules jobs for sending to printers. This value MUST be one of the Thread Priority values constants. This key name MAY be used remotely.<218>
"W3SvcInstalled" REG_DWORD			0x0001 if the web printing services are installed on the machine that hosts the print server.
"PrintDriverIsolationGroups" REG_SZ	X	X	A string that specifies groups of printer driver names.<219> The printer drivers in each group are executed in the same process, but printer drivers in different groups are executed in separate processes. The format of the string is as follows: ▪ Printer driver names within a group are separated from each other by a single backslash "\". ▪ Groups are separated from each other by two backslashes "\\". ▪ The last group is terminated by a null character.
"PrintDriverIsolationTimeBeforeRecycle" REG_DWORD	X	X	A time in milliseconds that specifies the maximum time span a printer driver isolation process should be used for before it is shut down and restarted; the shut down and restart sequence reclaims memory potentially leaked by drivers.<220>
"PrintDriverIsolationMaxobjsBeforeRecycle" REG_DWORD	X	X	A count that specifies the maximum number of operations a printer driver isolation process should be used for before it is shut down and restarted; the shut down and restart sequence reclaims memory potentially leaked by drivers.<221>
"PrintDriverIsolationIdleTimeout" REG_DWORD	X	X	A time in milliseconds that specifies the maximum time a printer driver isolation process should remain idle before it is shut down.<222>
"PrintDriverIsolationExecutionPolicy" REG_DWORD		X	An integer that specifies if printer driver isolation is enabled on the print server. 0x00000000 indicates that printer driver isolation is disabled. 0x00000001 indicates that printer driver isolation is enabled.<223>
"PrintDriverIsolationOverrideCompat"		X	An integer value that specifies if the print server overrides the printer driver's indication

Server handle key value name/registry type	Read-write	Behavior optional	Meaning
REG_DWORD			of printer driver isolation compatibility.<224> 0x00000001 indicates that the print server runs the printer driver in isolation mode, even if they do not indicate that they are compatible through the PRINTER_DRIVER_SANDBOX_ENABLED driver attribute flag.
"V4DriverDisallowPrinterUIApp" REG_DWORD	X	X	An integer value that specifies if printer drivers with a driver version (see *cVersion* in section 2.2.1.3.1) of 0x00000004 are enabled to run their **printer UI applications**, if any. <225> If the value is not configured, or if the value is set to 0x00000000, running of printer UI applications is enabled. If the value is set to 0x00000001, running of printer UI applications is disabled. This value has no effect on drivers with a driver version other than 0x00000004. By default this value is not configured.

2.2.3.10.1 OSVERSIONINFO

The OSVERSIONINFO structure specifies operating system (OS) version information for use with Server Handle Key Values (section 2.2.3.10).



dwOSVersionInfoSize (4 bytes): The size of the OSVERSIONINFO structure in bytes.

dwMajorVersion (4 bytes): The major OS version. See **dwMajorVersion** in SPLCLIENT_INFO Members (section 2.2.1.3.7) for details.

dwMinorVersion (4 bytes): The minor OS version. See **dwMinorVersion** in SPLCLIENT_INFO Members (section 2.2.1.3.7) for details.

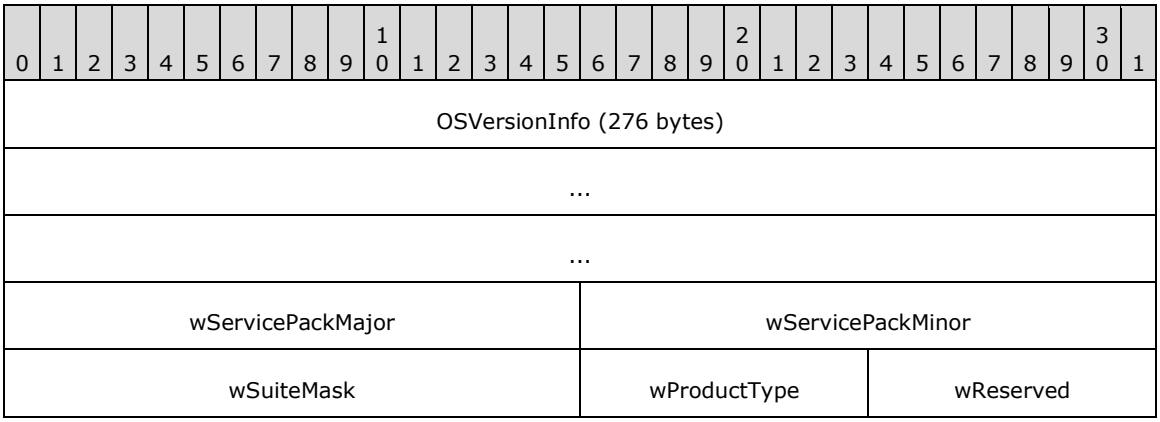
dwBuildNumber (4 bytes): The build number of the OS.<226>

dwPlatformId (4 bytes): The OS platform. See **wProcessorArchitecture** in SPLCLIENT_INFO Members (section 2.2.1.3.7) for details.

szCSDVersion (256 bytes): A maintenance string for Microsoft Product Support Services (PSS) use.

2.2.3.10.2 OSVERSIONINFOEX

The OSVERSIONINFOEX structure specifies extended operating system (OS) version information for use with Server Handle Key Values (section 2.2.3.10).



OSVersionInfo (276 bytes): An OSVERSIONINFO structure (section 2.2.3.10.1), which specifies basic OS version information.

wServicePackMajor (2 bytes): The major version number of the latest Service Pack installed on the system. For example, for Service Pack 3, the major version number is 3. If no Service Pack has been installed, the value is zero.

wServicePackMinor (2 bytes): The minor version number of the latest Service Pack installed on the system. For example, for Service Pack 3, the minor version number is 0.

wSuiteMask (2 bytes): A value that identifies the product suites available on the system, consisting of Product Suite Flags (section 2.2.3.10.5).

wProductType (1 byte): Additional information about the OS, which MUST be an OS_TYPE enumeration (section 2.2.3.10.3) value.

wReserved (1 byte): A field that SHOULD be initialized to zero when sent and MUST be ignored on receipt.

2.2.3.10.3 OS_TYPE Enumeration

The OS_TYPE enumeration specifies information about the operating system (OS) type for use with Server Handle Key Values (section 2.2.3.10).

```
typedef  enum
{
  VER_NT_WORKSTATION = 0x00000001,
  VER_NT_DOMAIN_CONTROLLER = 0x00000002,
  VER_NT_SERVER = 0x00000003
```

```
} OS_TYPE;
```

VER_NT_WORKSTATION: The OS is a Windows NT operating system workstation.<227>

VER_NT_DOMAIN_CONTROLLER: The OS is a Windows NT **domain controller**.<228>

VER_NT_SERVER: The OS is a Windows NT server.<229> A server that is also a domain controller is reported as **VER_NT_DOMAIN_CONTROLLER**, not **VER_NT_SERVER**.

2.2.3.10.4 Event Log Flags

The Event Log Flags specify events for a print server to log in its internal event log, for use with Server Handle Key Values (section 2.2.3.10). These flags can be combined to specify multiple options.

Constant/value	Description
EVENTLOG_ERROR_TYPE 0x00000001	An event that indicates a significant problem such as loss of data or loss of functionality. For example, a service fails to load during startup.
EVENTLOG_WARNING_TYPE 0x00000002	An event that is not necessarily significant, but may indicate a possible future problem. For example, disk space is low. If an application can recover from an event without loss of functionality or data, it can classify the event as a warning.
EVENTLOG_INFORMATION_TYPE 0x00000004	An event that describes the successful operation of an application, driver, or service. For example, a network driver loads successfully.
EVENTLOG_AUDIT_SUCCESS 0x00000008	An event that records an audited security access attempt that is successful. For example, a user's attempt to log onto a system is successful.
EVENTLOG_AUDIT_FAILURE 0x00000010	An event that records an audited security access attempt that fails. For example, a user's attempt to access a network drive fails.

2.2.3.10.5 Product Suite Flags

The Product Suite Flags are implementation-specific values for the product suites that are available on the operating system (OS). They are used with Server Handle Key Values (section 2.2.3.10).<230>

2.2.3.10.6 Thread Priority Values

The Thread Priority values specify priorities for threads on which print servers schedule jobs or send data to printers, for use with Server Handle Key Values (section 2.2.3.10).

Constant/value	Description
THREAD_PRIORITY_LOWEST 0xFFFFFFFE	The thread can be scheduled after threads with any other priority.
THREAD_PRIORITY_BELOW_NORMAL 0xFFFFFFFF	The thread can be scheduled after threads with normal priority and before those with lowest priority.
THREAD_PRIORITY_NORMAL 0x00000000	The thread can be scheduled after threads with above normal priority and before those with below normal priority. Threads have normal priority by default.

Constant/value	Description
THREAD_PRIORITY_ABOVE_NORMAL 0x00000001	The thread can be scheduled after threads with highest priority and before those with normal priority.
THREAD_PRIORITY_HIGHEST 0x00000002	The thread can be scheduled before threads with any other priority.

2.2.3.11 Printer Data Values

Printer Data Values are used to store printer configuration data. The values named in the following table MUST be supported on print servers for each printer as follows:

- In a call to RpcGetPrinterData (section 3.1.4.2.7) or RpcGetPrinterDataEx (section 3.1.4.2.19), the *pValueName* parameter identifies the data that MUST be returned in the *pData* parameter. The key name pointed to by *pKeyName* MUST be NULL, and the value pointed to by *pValueName* MUST be one of the strings specified in the "Printer data value name" column.

  The specified registry type values are defined in section 2.2.3.9.

- If the "Read-write" column is checked, a print server SHOULD keep track of the value set by a call to RpcSetPrinterData (section 3.1.4.2.8) or RpcSetPrinterDataEx (section 3.1.4.2.18) in the *pData* parameter, and it SHOULD return the same value in a subsequent call to RpcGetPrinterData or RpcGetPrinterDataEx.

- The "Meaning" column describes the printer configuration data that is associated with the printer data value name, and in some cases it specifies print server behavior. If the "Behavior optional" column is checked, that behavior is implementation-specific and not mandatory.<231>

Printer data value name / registry type	Read-write	Behavior optional	Meaning
"HardwareId" REG_SZ		X	A string that identifies compatible printer drivers for the printer. The value of this identifier is implementation-specific.<232>
"EnableBranchOfficePrinting" REG_DWORD		X	An integer that specifies whether print clients connect directly to the printer. If the value exists and contains a REG_DWORD value of 0x00000001, print clients render print jobs and send printed data directly to the printer instead of using the Job Management Methods (section 3.1.4.3) and Document Printing Methods (section 3.1.4.9) on the print server.<233>
"SeparatorFileData"		X	If the printer driver for the printer has a driver version of 0x00000004, and the EnableBranchOfficePrinting printer data value exists and contains a REG_DWORD value of 0x00000001, this value contains the contents of the separator page, if any, associated with the printer.<234>
"V4_Driver_Hardware_IDs" REG_MULTI_SZ			A multisz containing one or more curly-braced GUID strings that identify the compatible printer drivers with a driver version (see *cVersion* in section 2.2.1.3.1) of 0x00000004.<235> This value is required for printers with drivers having a driver version of 0x00000004.

Printer data value name / registry type	Read-write	Behavior optional	Meaning
"XpsFormat" REG_BINARY		X	A binary value that specifies the XML Paper Specification (XPS) formats that the server supports: Microsoft XPS [MSFT-XMLPAPER] and/or OpenXPS [ECMA-388].<236> The binary value MUST be either one of the following two: ▪ A single DWORD (4 bytes) if only one XPS format is supported. ▪ A pair of DWORDs (8 bytes) if both XPS formats are supported. Each DWORD MUST contain one of the following two values: ▪ 0x00000001 for Microsoft XPS. ▪ 0x00000002 for OpenXPS. When both formats are supported, the two DWORD values can be specified in any order.
"MergedData" REG_BINARY			A binary value that specifies the contents of the GPD ([MSDN-GPDFiles]) or PPD ([PS-PPD4.3]) file for a printer driver with a driver version of 0x00000004.<237> This value is required for printers with printer drivers having a driver version of 0x00000004.
"MergedDataName" REG_SZ			A string value that specifies the name of the GPD ([MSDN-GPDFiles] or PPD ([PS-PPD4.3]) file for a printer driver with a driver version of 0x00000004.<238> This value is required for printers with printer drivers having a driver version of 0x00000004.
"BranchOfficeLoggingEnabled" REG_DWORD		X	An integer that specifies whether branch office print remote logging is enabled, which means that print clients operating in **branch office print mode** SHOULD log printing-related Windows Events on the print server.<239> If this value is not configured, or if it is set to 0x00000001, the client SHOULD create printing events on the server as well as the client. If the value is set to zero, the client SHOULD NOT create printing events on the server.
"BranchOfficeOfflineLogSize" REG_DWORD		X	The maximum size, in megabytes (MB), of the **Branch Office Print Remote Log Offline Archive** (section 3.2.1).<240> This value MUST be 1 MB or more. If the print client is unable to contact the print server when processing a branch office print job, and branch office print remote logging is enabled, the printing event SHOULD be written to the **Branch Office Print Remote Log Offline Archive**. If the offline archive exceeds this maximum size, printing events SHOULD

Printer data value name / registry type	Read-write	Behavior optional	Meaning
			be discarded, and the server, when it can be contacted, SHOULD be informed of the overflow condition. If this value is not configured, an implementation-defined default value SHOULD<241> be used. If this value is set to zero, the client SHOULD NOT create printing events on the server.
"MinimumSupportedClientBuild" REG_DWORD		X	An integer value that specifies the minimum print client version required in order to connect to this printer. This is expressed as a Windows build number and is provided by printer drivers having a version of 0x00000004 which have known dependencies that cannot be met on clients with versions prior to this build number. If the build number of the print client is less than this value, the client MUST NOT create a connection to this printer. If this value is not supplied or is zero, any client, regardless of build number, can connect to this printer.

2.2.3.12 Status and Attribute Values

Printer status	Description
PRINTER_STATUS_BUSY 0x00000200	The printer is busy.
PRINTER_STATUS_DOOR_OPEN 0x00400000	The printer door is open.
PRINTER_STATUS_ERROR 0x00000002	The printer is in an error state.
PRINTER_STATUS_INITIALIZING 0x00008000	The printer is initializing.
PRINTER_STATUS_IO_ACTIVE 0x00000100	The printer is in an active input or output state.
PRINTER_STATUS_MANUAL_FEED 0x00000020	The printer is in a manual feed state.
PRINTER_STATUS_NOT_AVAILABLE 0x00001000	The printer is not available for printing.
PRINTER_STATUS_NO_TONER 0x00040000	The printer is out of toner.
PRINTER_STATUS_OFFLINE 0x00000080	The printer is offline.
PRINTER_STATUS_OUTPUT_BIN_FULL 0x00000800	The printer's output bin is full.
PRINTER_STATUS_OUT_OF_MEMORY	The printer has run out of memory.

Printer status	Description
0x00200000	
PRINTER_STATUS_PAGE_PUNT 0x00080000	The printer cannot print the current page.
PRINTER_STATUS_PAPER_JAM 0x00000008	Paper is stuck in the printer.
PRINTER_STATUS_PAPER_OUT 0x00000010	The printer is out of paper.
PRINTER_STATUS_PAPER_PROBLEM 0x00000040	The printer has an unspecified paper problem.
PRINTER_STATUS_PAUSED 0x00000001	The printer is paused.
PRINTER_STATUS_PENDING_DELETION 0x00000004	The printer is being deleted as a result of a client's call to RpcDeletePrinter. No new jobs can be submitted on existing printer objects for that printer.
PRINTER_STATUS_POWER_SAVE 0x01000000	The printer is in power-save mode.<242>
PRINTER_STATUS_PRINTING 0x00000400	The printer is printing.
PRINTER_STATUS_PROCESSING 0x00004000	The printer is processing a print job.
PRINTER_STATUS_SERVER_OFFLINE 0x02000000	The printer is offline.<243>
PRINTER_STATUS_SERVER_UNKNOWN 0x00800000	The printer status is unknown.<244>
PRINTER_STATUS_TONER_LOW 0x00020000	The printer is low on toner.
PRINTER_STATUS_USER_INTERVENTION 0x00100000	The printer has an error that requires the user to do something.
PRINTER_STATUS_WAITING 0x00002000	The printer is waiting.
PRINTER_STATUS_WARMING_UP 0x00010000	The printer is warming up.

Printer attribute	Description
PRINTER_ATTRIBUTE_DEFAULT 0x00000004	Indicates the printer is the default printer in the system.
PRINTER_ATTRIBUTE_DIRECT 0x00000002	Job is sent directly to the printer (it is not spooled).

Printer attribute	Description
PRINTER_ATTRIBUTE_DO_COMPLETE_FIRST 0x00000200	If set and printer is set for print-while-spooling, any jobs that have completed spooling are scheduled to print before jobs that have not completed spooling.<245>
PRINTER_ATTRIBUTE_ENABLE_BIDI 0x00000800	Indicates whether bidirectional communications are enabled for the printer.<246>
PRINTER_ATTRIBUTE_ENABLE_DEVQ 0x00000080	Setting this flag causes mismatched documents to be held in the queue.<247>
PRINTER_ATTRIBUTE_FAX 0x00004000	If set, printer is a fax printer.
PRINTER_ATTRIBUTE_KEEPPRINTEDJOBS 0x00000100	If set, jobs are kept after they are printed. If cleared, jobs are deleted.<248>
PRINTER_ATTRIBUTE_LOCAL 0x00000040	Printer is a local printer.
PRINTER_ATTRIBUTE_NETWORK 0x00000010	Printer is a network printer connection.
PRINTER_ATTRIBUTE_PUBLISHED 0x00002000	Indicates whether the printer is published in the directory service (DS).<249>
PRINTER_ATTRIBUTE_QUEUED 0x00000001	If set, the printer spools and starts printing after the last page is spooled. If cleared, and **PRINTER_ATTRIBUTE_DIRECT** is not set, the printer spools and prints while spooling.
PRINTER_ATTRIBUTE_RAW_ONLY 0x00001000	Indicates that only RAW data type print jobs MUST be spooled.<250>
PRINTER_ATTRIBUTE_SHARED 0x00000008	Printer is shared.
PRINTER_ATTRIBUTE_TS 0x00008000	Printer is a redirected terminal server printer.
PRINTER_ATTRIBUTE_WORK_OFFLINE 0x00000400	Indicates whether the printer is currently connected. If the printer is not currently connected, print jobs will continue to spool.<251>

Job status	Description
JOB_STATUS_BLOCKED_DEVQ 0x00000200	Printer driver cannot print the job.<252>
JOB_STATUS_COMPLETE 0x00001000	Job has been delivered to the printer.
JOB_STATUS_DELETED 0x00000100	Job has been deleted.<253>
JOB_STATUS_DELETING 0x00000004	Job is being deleted.

Job status	Description
JOB_STATUS_ERROR 0x00000002	An error is associated with the job.
JOB_STATUS_OFFLINE 0x00000020	Printer is offline.
JOB_STATUS_PAPEROUT 0x00000040	Printer is out of paper.
JOB_STATUS_PAUSED 0x00000001	Job is paused.
JOB_STATUS_PRINTED 0x00000080	Job has printed.
JOB_STATUS_PRINTING 0x00000010	Job is printing.
JOB_STATUS_RESTART 0x00000800	Job has been restarted.<254>
JOB_STATUS_SPOOLING 0x00000008	Job is spooling.
JOB_STATUS_USER_INTERVENTION 0x00000400	Printer has an error that requires the user to do something.<255>

2.2.3.13 BIDI_TYPE Enumeration

The BIDI_TYPE enumeration specifies the type of data transferred in a bidirectional operation.

```
typedef  enum
{
  BIDI_NULL = 0x00000000,
  BIDI_INT = 0x00000001,
  BIDI_FLOAT = 0x00000002,
  BIDI_BOOL = 0x00000003,
  BIDI_STRING = 0x00000004,
  BIDI_TEXT = 0x00000005,
  BIDI_ENUM = 0x00000006,
  BIDI_BLOB = 0x00000007
} BIDI_TYPE;
```

BIDI_NULL: No bidirectional data.

BIDI_INT: Bidirectional data is an integer.

BIDI_FLOAT: Bidirectional data is a floating-point number.

BIDI_BOOL: Bidirectional data is a Boolean value.

BIDI_STRING: Bidirectional data is a string.

BIDI_TEXT: Bidirectional data is text data.

BIDI_ENUM: Bidirectional data is an enumeration.

BIDI_BLOB: Bidirectional data is a data **BLOB**.

2.2.4 Rules for Members

The following sections specify rules for the common string and flag members that are passed as parameters, or are parts of structures that are passed as parameters, to methods in this protocol.

2.2.4.1 Access Values

Access values specify the access rights that a caller is requesting.

2.2.4.2 Datatype Names

A Datatype Name is a string that contains the name of a data type. It MUST uniquely identify a format for print data that is supported by a print processor. The string MUST NOT be empty.

The data type SHOULD be specified by the implementation.<256>

2.2.4.3 Driver Names

A Driver Name is a string that contains the name of a printer driver. The string MUST NOT be empty.

An implementation MAY restrict the length of driver name strings.<257>

2.2.4.4 Environment Names

An Environment Name is a string that contains the name of the operating system environment. The string MUST permit white space and MUST NOT be empty.

Environment name strings MAY be specified by the implementation.<258>

2.2.4.5 Form Names

A Form Name is a string that contains the name of a printer form. It MUST uniquely identify a printer form on the system. The string MUST NOT be empty.

An implementation MAY restrict the length of form name strings.<259>

2.2.4.6 Job Control Values

Job control values specify actions such as pause and cancel (see RpcSetJob (section 3.1.4.3.1) for a list of job control values and their meanings). A job control value MUST NOT be zero if a job control action is required.

2.2.4.7 Key Names

A Key Name is a string that contains the name of a **printer key**. It MUST uniquely identify a path under the main registry key where printer configuration data is kept. The string MUST permit a backslash ("\") as delimiter for paths with one or more subkeys.

An implementation MAY restrict the length of key name strings.<260>

2.2.4.8 Monitor Names

A Monitor Name is a string that contains the name of a port monitor. It MUST uniquely identify a port monitor on the system. The string MUST NOT be empty.

An implementation MAY restrict the length of monitor name strings.<261>

2.2.4.9 Path Names

A Path Name is a string that contains the file name, or full path and file name, including subdirectories, for the identified file. The string MUST permit white space and MUST NOT be empty.

If the path name string identifies a network-addressable file, it MUST be a **DNS**, **NetBIOS**, IPv4, or universal naming convention (UNC) name, or it SHOULD be an **IPv6** name.<262> The string SHOULD be of the form "\\ServerName\ShareName" and MUST identify a unique shared folder on the machine.

The pattern followed by path name strings of local files SHOULD be specified by the implementation.<263>

An implementation SHOULD restrict the length of path name strings.<264>

For further information on **DNS** names, see [RFC819] section 2. **NetBIOS** names are specified in [RFC1001] section 14.

2.2.4.10 Port Names

A Port Name is a string that contains the name of a printer port. The string MUST NOT be empty.

The pattern followed by port name strings MAY be specified by the port monitor implementation that they belong to. <265>

2.2.4.11 Print Processor Names

A Print Processor Name specifies a string that contains the name of a print processor. It MUST uniquely identify a print processor for a given operating system environment. The string SHOULD NOT be empty.

An implementation MAY restrict the length of print processor name strings.<266>

2.2.4.12 Print Provider Names

A Print Provider Name is a string that contains the name of a print provider. It MUST uniquely identify a print provider on the system. The string MUST NOT be empty.

An implementation MAY restrict the length of print provider strings.<267>

2.2.4.13 Printer Change Values

Printer Change values specify printing-related events that occur on a print server. These values consist of Printer Change Flags (section 2.2.3.6).

2.2.4.14 Printer Names

A Printer Name is a string that contains the name of a facility for output. It MUST uniquely identify a destination that is local or on a print server. The string MUST NOT be empty.

In RpcOpenPrinter and RpcOpenPrinterEx parameters, all instances of printer names MUST follow either the **PRINTER_NAME** or **PRINTER_NAME_EX** pattern. In any other context, printer names MUST follow the **PRINTER_NAME** pattern.

The **PRINTER_NAME** pattern is defined as follows

```
UNICODE NOCOMMA NOBACKSLASH = <Any UTF-16LE character except ","
    and "\">

UNICODE_NOBACKSLASH = <Any UTF-16LE character, except "\">

PRINTER NAME = (SERVER NAME LOCAL PRINTER NAME) |
    (WEB_PRINT_SERVER "/" "printers" "/" LOCAL_PRINTER_NAME "/"
    ".printer")

WEB_PRINT_SERVER = "http: " "//" host [":" port]

LOCAL_PRINTER_NAME = 1#UNICODE_NOCOMMA_NOBACKSLASH
```

where:

- **SERVER_NAME** is defined in section 2.2.4.16.
- **WEB_PRINT_SERVER** specifies the address of the web print server.
- **LOCAL_PRINTER_NAME** is a string specifying the local printer name or share name of the printer. Printer names MUST NOT contain the characters ',' and '\'.

Basic notational conventions are specified in [RFC2616] section 2, and the terms **host** and **port** are defined in [RFC3986] section 3.2.2.

The **PRINTER_NAME_EX** pattern extends the **PRINTER_NAME** pattern and is defined as follows:

```
PRINTER_NAME_EX = PRINTER_NAME_EX1 | PRINTER_NAME_EX2 |
    PRINTER_NAME_EX3 | PRINTER_NAME_EX4 | PRINTER_NAME_EX5

PRINTER NAME EX1 = SERVER NAME LOCAL PRINTER NAME "," #" "
    "Job " #" " JOB IDENTIFIER

JOB_IDENTIFIER = MUST be between 1 and 2,147,483,648, inclusive.

PRINTER NAME EX2 = SERVER NAME LOCAL PORT NAME "," #" " "Port"

PORT NAME = 1#UNICODE NOCOMMA

PRINTER_NAME_EX3 = SERVER_NAME_NE "\," #" " "XcvPort "
    PORT_NAME

PORT_NAME = 1#UNICODE_NOCOMMA

PRINTER_NAME_EX4 = SERVER_NAME_NE "\," #" " "XcvMonitor "
    PORT_MONITOR_NAME


PORT_MONITOR_NAME = 1#UNICODE_NOBACKSLASH

PRINTER_NAME_EX5 = SERVER_NAME_NE
```

where:

- **SERVER_NAME_NE** is defined in section 2.2.4.16

- **PRINTER_NAME_EX1** specifies the print job identified by **JOB_IDENTIFIER** on the printer specified with **PrinterName**.

  When the **PRINTER_NAME_EX1** form is used as the name parameter with RpcOpenPrinter or RpcOpenPrinterEx, a job object will be returned that can be used with RpcReadPrinter and RpcWritePrinter to read and write job content.

- **JOB_IDENTIFIER** specifies a server-wide unique decimal identifier for the print job.

- **PRINTER_NAME_EX2** specifies the name of the port to be opened.

  When the **PRINTER_NAME_EX2** form is used as the name parameter with RpcOpenPrinter or RpcOpenPrinterEx, a port object is returned that can be used with RpcStartDocPrinter and RpcWritePrinter to print directly to a port without intermediate spooling.

- **PORT_NAME** is a port name, as specified in section 2.2.4.10.

- **PRINTER_NAME_EX3** specifies the name of the port to be opened.

  When the **PRINTER_NAME_EX3** form is used as the name parameter with RpcOpenPrinter or RpcOpenPrinterEx, a port object is returned that can be used with RpcXcvData to communicate directly with a port.

- **PRINTER_NAME_EX4** specifies the name of the port monitor to be opened.

  When this form is used as the name parameter with RpcOpenPrinter or RpcOpenPrinterEx, a port monitor object is returned that can be used with RpcXcvData to communicate directly with a port monitor.

- **PORT_MONITOR_NAME** is a monitor name, as specified in section 2.2.4.8.

- **PRINTER_NAME_EX5** specifies the print server to be opened.

The **PRINTER_NAME_WITH_POSTFIX** pattern extends the **PRINTER_NAME** and **PRINTER_NAME_EX** patterns and is defined as follows:

```
PRINTER_NAME_WITH_POSTFIX = PRINTER_NAME_PREFIX "," PRINTER_NAME_POSTFIX
```

where:

- **PRINTER_NAME_PREFIX** is the **PRINTER_NAME** or **PRINTER_NAME_EX** pattern.

- **PRINTER_NAME_POSTFIX** is an implementation-defined string.

An implementation MAY not use the **PRINTER_NAME_WITH_POSTFIX** pattern and instead use only the **PRINTER_NAME** and **PRINTER_NAME_EX** patterns.<268>

An implementation MAY restrict the length of printer name strings. <269>

2.2.4.15 Registry Type Values

A Registry Type Value specifies the type of a data value in the registry using one of the constants specified in Registry Type Value (section 2.2.3.9).

2.2.4.16 Server Names

A Server Name is a string that contains the name of a print server. It MUST be a **DNS**, **NetBIOS**, IPv4, or Universal Naming Convention (UNC) name, or it MAY be an IPv6 name.<270> An empty server name string is interpreted as identifying the server that the **RPC server endpoint** is bound to.

Server names MUST follow the **SERVER_NAME** pattern, which is defined as follows.

```
SERVER_NAME = "\\" host "\" | ""
```

The **SERVER_NAME_NE** pattern is used where a non-empty server name is required and is defined as follows.

```
SERVER_NAME_NE = "\\" host
```

An implementation MAY restrict the length of server name strings. <271>

For further information on **DNS** names, see [RFC819] section 2. Details about **NetBIOS** names are specified in [RFC1001] section 14. Details about basic notational conventions are specified in [RFC2616] section 2. The definition of **host** is specified in [RFC3986] section 3.2.2.

2.2.4.17 User Names

A User Name is a string that contains the name of a valid **principal**. It MUST be formatted as either an Active Directory name or a name with a fully qualified domain. The string MUST NOT be empty,

For more information on user names, see [MSDN-ADOVRVW] Object Names and Identities.

2.2.4.18 Value Names

A Value Name is a string that contains the name of a value that is kept under a printer key. The string MUST NOT be empty.

An implementation MAY restrict the length of value name strings. <272>

2.3 Directory Service Interaction

2.3.1 Interaction Summary

The Print System Remote Protocol initiates interactions with the Active Directory in the following way:<273>

- The print client calls RpcSetPrinter with a PRINTER_CONTAINER structure that is initialized to contain a Level value of 0x00000007 and a pointer to a PRINTER_INFO_7 structure, which specifies the publishing action for the server to take for a print queue. The possible actions are publishing data about a print queue to the Active Directory, modifying the directory data for a print queue already published in the Active Directory, and removing the data for a print queue from the Active Directory.
- The print client calls RpcSetPrinter with a PRINTER_HANDLE that is a handle to a print queue that the print server has published to the Active Directory, and as a result of the print client request, the print server either changes the printer driver associated with the print queue or modifies print queue configuration settings that the print server has published in the Active Directory.
- The print client calls RpcDeletePrinter with a PRINTER_HANDLE that is a handle to a print queue that the print server has published to the Active Directory.

Print servers use LDAP, as specified in [RFC2251], to access data in the Active Directory. The implementation of LDAP by the Active Directory is described in section 3.1.1.4 of [MS-ADTS].

2.3.2 Directory Service Schema Elements

Print servers access the directory service (DS) schema classes and attributes that are listed in this section. For the syntactic specifications of the <Class> or <Class> <Attribute> pairs, refer to [MS-ADSC] and [MS-ADA3].

- The print server uses the printQueue Active Directory schema class to publish a print queue.
- When creating a print queue object in the Active Directory, the print server MUST specify the following mandatory attributes of the printQueue Active Directory schema object:

Class	Attribute
printQueue	versionNumber uNCName shortServerName serverName printerName

- When creating a print queue object in the Active Directory, the print server SHOULD specify the following optional attributes of the printQueue Active Directory schema object:

Class	Attribute
printQueue	priority printStatus printStartTime printStaplingSupported printSpooling printShareName printSeparatorFile printRateUnit printRate printPagesPerMinute printOwner printOrientationsSupported printNumberUp printNotify printNetworkAddress printMinYExtent printMinXExtent printMemory printMediaSupported printMediaReady printMaxYExtent printMaxXExtent printMaxResolutionSupported printMaxCopies printMACAddress printLanguage printKeepPrintedJobs printFormName

Class	Attribute
	printEndTime
	printDuplexSupported
	printColor
	printCollate
	printBinNames
	printAttributes
	portName
	physicalLocationObject
	operatingSystemVersion
	operatingSystemServicePack
	operatingSystemHotfix
	operatingSystem
	location
	driverVersion
	driverName
	defaultPriority
	bytesPerMinute
	assetNumber

2.3.3 Interaction Details

Print servers access data in the Active Directory through the domain controller, which acts as an LDAP server. Print servers locate the domain controller by using the mechanism described in [MS-ADTS] section 6.3.6.1. The remainder of this section describes the sequences of steps performed by print servers in their interactions with the Active Directory.

Unless otherwise noted, the print server SHOULD detect any LDAP failures that can occur during all of the following operations: publishing a print queue to the Active Directory (see section 2.3.3.1), modifying or deleting a print queue in the Active Directory (see section 2.3.3.2), or searching for print queues in the Active Directory (see section 2.3.3.3). If the print server detects an LDAP failure, it SHOULD retry the complete publish, modification, deletion, or search operation. Retries SHOULD be separated by time intervals decaying from 10 seconds to 2 hours.

Because of LDAP failures, print queue objects in the Active Directory will not necessarily match the state of the print server at all times. Aside from update operations executed at print server initialization or periodically as described in section 2.3.3.4, print servers MAY attempt to maintain consistency between their internal state and the objects in the Active Directory using other processes.

2.3.3.1 Publishing a Print Queue to the Active Directory

Print servers perform the following steps when fulfilling a client request to publish a print queue to the Active Directory.

1. Locate the domain controller as specified in section 2.3.3.
2. Determine the **distinguished name (DN)** for the **container** representing the print server in the directory. This container is created when the print server joins the domain, as described in [MS-ADOD] section 2.7.7.1. The print server uses the DRSR protocol [MS-DRSR] to determine the DN, as follows.

- The print server calls the RPC method IDL_DRSCrackNames as defined in [MS-DRSR] section 4.1.4. The input arguments for this call are the following:

Attribute	Value
hDrs	Context handle returned from calling IDL_DRSBind, as defined in [MS-DRSR] section 4.1.3.
dwInVersion	1
pmsgIn	A request with a string containing one name. This is the name of the computer's domain account (<domain>\<computer>, where <domain> is the name of the domain and <computer> is the name of the computer). The **formatOffered** field is DS_UNKNOWN_NAME, a member of the DS_NAME_FORMAT enumeration defined in [MS-DRSR] section 4.1.4.1.3. The **formatDesired** field is DS_FQDN_1779_NAME, also defined in the DS_NAME_FORMAT enumeration.

  As specified in [MS-DRSR] section 4.1.4, if it is successful, the IDL_DRSCrackNames method returns the object name in the requested format, which is the **fully qualified domain name (FQDN)** in this case.

3. Create the directory service object representing the printer by using the LDAP protocol. The specific steps are as follows:

   - The print server binds to the LDAP server by using the bind operation defined in section 4.2 of [RFC2251]. The **version** field MUST be set to 3, the **name** field MUST be a null value, and the authentication mechanism MUST be the GSS_SPNEGO SASL mechanism, as documented in [MS-ADTS] section 3.1.1.3.4.5.2.

   - The print server generates a **relative distinguished name (RDN)**, as defined in section 3.2 of [RFC2251], for the printer object in the directory. This DN is relative to the DN of the computer's directory object determined previously. There are no restrictions on the RDN generated beyond those requirements stated in [RFC2251]. As specified in [RFC2251], the FQDN for the printer object is the RDN generated here, prepended to the DN of the computer's directory object.

   - The print server adds the printer object to the directory and sets the attributes on the object by using exactly one LDAP add operation and any number of LDAP modify operations, as defined in sections 4.7 and 4.9 of [RFC2251]. The printer object MUST conform to the schema for the printQueue Active Directory object class as discussed in section 2.3.2.

     In accordance with the schema, the print server MUST set the required properties of the printer object as part of the LDAP add request. These required properties MUST be set as follows:

Attribute	Value
versionNumber	4
shortServerName	A string containing the machine name of the print server.
serverName	A string containing the fully qualified DNS name, as defined in [RFC819], of the print server.
printerName	A string containing the name of the printer. This corresponds to the **pPrinterName** field described in section 3.1.4.1.5.
uNCName	A string containing the UNC name of the printer. This is of the form "\\<serverName>\<printerName>", where <serverName> is the value of serverName as defined in this table, and <printerName> is the value of printerName.

     The print server SHOULD publish to the directory any of the optional attributes specified in the schema for the print queue object class. The print server determines the values for both the mandatory and optional attributes in the schema by querying its internal state or the printer

driver associated with the printer, as necessary depending on the print server implementation.<274>

The print server SHOULD also publish to the directory values previously set under certain predefined keys ("DsSpooler", "DsDriver", and "DsUser") by print clients using calls to **RpcSetPrinterDataEx** (section 3.1.4.2.18). If the name of a value under one of these keys matches the name of an attribute in the printQueue object class schema, the print server SHOULD set the value of this attribute in the directory object to the data stored in the printer data value.<275>

For all attribute values the print server determined by querying internal state or a printer driver, as specified earlier in this section, the print server SHOULD save these values into the printer data corresponding to the printer. Print clients can later access these data values using calls to **RpcGetPrinterDataEx** (section 3.1.4.2.19). The print server SHOULD save these values under the predefined "DsSpooler" key if it determined the value by querying internal state or the predefined "DsDriver" key if it queried a printer driver. The print server SHOULD use value names that are the same as the names of the corresponding LDAP attributes.

If the LDAP add operation fails, the print server SHOULD wait for the retry interval specified in section 2.3.3 and then perform all the steps in this section. If the LDAP add operation succeeds but any LDAP modify operation fails, the print server SHOULD retry, as specified in section 2.3.3.2, each failed LDAP modify operation.

- The print server unbinds from the LDAP server by using the unbind operation defined in [RFC2251] section 4.3.

2.3.3.2 Modifying or Deleting a Print Queue in the Active Directory

Print servers perform the following steps when modifying or deleting a print queue in the Active Directory.

- Locate the domain controller and bind to the LDAP server as specified in section 2.3.3.1.
- Determine the fully qualified DN of the print queue object in the directory, either by searching for the print queue as described in section 2.3.3.3, or by using implementation-specific state saved during the publish operation.<276>
- Initiate LDAP modify or delete operations (as defined in sections 4.6 and 4.8 of [RFC2251]) to update or delete the object as desired.<277>
- Unbind from the LDAP server as specified in section 2.3.3.1.

2.3.3.3 Searching for Print Queues in the Active Directory

Print servers also use LDAP to search for print queues in the Active Directory. Print servers perform the following steps to search for print queues:

- Locate the domain controller and bind to the LDAP server as described in section 2.3.3.1.
- Search for print queues with the desired attributes, using the LDAP search operation, as defined in section 4.5 of [RFC2251]. The format of the LDAP search request is described in section 4.5.1 of [RFC2251]. The parameters of the search request are set as follows:

Parameter	Value
baseObject	Print servers MAY specify the default naming context retrieved from the root DSE. The root DSE is defined in [MS-ADTS] section 1.1, and the default naming context attribute is described in [MS-ADTS] section 3.1.1.3.2.3.

Parameter	Value
scope	Print servers SHOULD specify wholeSubtree.
derefAliases	Print servers SHOULD specify neverDerefAliases.
sizeLimit	This value is dependent on the print server implementation and does not affect the protocol.
timeLimit	This value is dependent on the print server implementation and does not affect the protocol.
filter	The filter parameter of the search request MUST contain the requirement that the returned objects be of the print queue object class. In the string representation of filters described in [RFC2254], this requirement is written as "(objectClass = printQueue)". In conjunction with this requirement, the filter can contain any restrictions based on any combination of attributes of the schema for the print queue object class.
attributes	The print server can request any set of attributes in the schema, but it SHOULD NOT fail if any optional attribute is missing.

- Unbind from the LDAP server as described in section 2.3.3.1.

Print servers can also search the **global catalog (GC)** for print queues in other **NCs**.<278> The GC and naming contexts are defined in section 1.1. The Active Directory implementation of the GC is described in [MS-ADTS] section 3.1.1.1.8. Print clients use LDAP to perform searches on the GC in the same way as above.

Print servers can negotiate encryption of LDAP messages as part of the SASL authentication during the binding process, as described in [MS-ADTS] section 5.1.1.1.2.<279>

If an Active Directory print queue object is enumerated by the search, but the mandatory information specified in section 2.3.2 is not present in that print queue object, the print server SHOULD ignore this object and continue to the next enumerated print queue object.

Print servers MAY retry failed LDAP search operations.

2.3.3.4 Initializing the Print Server for Active Directory

Print servers perform the following initialization when Active Directory is available:

- Enumerate through the **List of Printers** and, for each print queue with the PRINTER_ATTRIBUTE_PUBLISHED flag set, verify that the print queue is already published into the Active Directory. If the print queue is not already published, publish the print queue to the Active Directory as described in section 2.3.3.1. If the print queue is already published, verify that the data in the Active Directory is current and, if not, update the Active Directory entry for this print queue as described in section 2.3.3.2. If the print queue is marked as "Delete Pending", delete the Active Directory entry for the print queue as described in section 2.3.3.2. If an Active Directory operation fails for a print queue, the print server SHOULD continue enumerating through the **List of Printers**.

Print servers can also perform the following periodic operations when Active Directory is available, but only when the print server is configured to do so:

- Periodically enumerate through the **List of Printers** and update the Active Directory as described earlier in this section.<280>
- Periodically search for the print queues in the Active Directory (see section 2.3.3.3) and delete (see section 2.3.3.2) all print queues that are not present in the current **List of Printers**.<281>

3 Protocol Details

3.1 Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of a possible data organization that a print server implementation might need to maintain in order to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this specification.

This protocol depends on an abstract data model that maintains information about printers and related objects. These objects represent physical output devices, and they are used in the protocol to communicate with those devices, to print to them, and to manage their configurations.

A print server has a SECURITY_DESCRIPTOR controlling access to the server object. By default, the SECURITY_DESCRIPTOR allows SERVER_ALL_ACCESS to members of the Administrators group and SERVER_EXECUTE access to all other users.

A print server behaves as if it hosted the following objects in the following hierarchy.

Note: The abstract data model specified for this protocol is identical to that specified for the Print System Asynchronous Remote Protocol specified in [MS-PAR]. A print server maintains only one copy of the data underlying the implementation that exposes this protocol or [MS-PAR].

List of Print Server Names: It is assumed that a client of this protocol has obtained the name of at least one print server that supports this protocol before it is invoked. There are various ways a client can build a **list of print server names** ([MS-ADLS], [MS-ADSC], [MS-RAP], and [MS-SMB2]).

The server name that is passed to the print server by the client can differ from the server name that the print server determined upon its own initialization; for example, when aliasing of server names via DNS or directory services (DS) takes place.

Other methods, such as RpcAddPerMachineConnection (section 3.1.4.2.24), pass in an additional server name parameter that identifies a print server that is different from the one handling the API call. To correctly resolve server names, each print server maintains a mapping between server names and server addresses. When composing a response to the client, the print server that handles the API call uses the same server name that the client passed as the parameter in this call.

A print server does not persist the list of print server names between **server restarts**.

List of Form Objects: Form objects represent information about units of physical media on output devices, which are available to print clients. Examples are printer forms, such as sheets of paper. Each form object also contains a data element indicating whether the form object was added by a user and if the form object is specific to a printer.

Form objects are added, removed, accessed, and enumerated using Form Management Methods (section 3.1.4.5).

A print server initializes the **list of form objects** to contain an implementation-specific set of built-in form objects. A print server persists the **list of form objects** between server restarts.

List of Printers: Each printer (also referred to as print queue) represents a physical print device or a number of homogeneous physical devices installed on the print server. Each printer object maintains the following data elements:

- A name that uniquely identifies the printer.
- A reference to a printer driver object for the printer.
- A reference to a print processor object.
- References to one or more port objects.<282>
- A list of queued or printing print jobs, where each print job has a SECURITY_DESCRIPTOR controlling access to the print job and a data type, which specifies the type of data provided by the printing application to the printer during the print job. By default, the SECURITY_DESCRIPTOR allows JOB_ALL_ACCESS to members of the Administrators group and to the client who created the print job, as well as a separate JOB_READ access to the client who created the job. Each print job can also have zero or more **Job Named Properties**.<283><284>
- Global _DEVMODE settings.
- Per-user _DEVMODE settings.
- A SECURITY_DESCRIPTOR controlling access to the printer object. By default, the SECURITY_DESCRIPTOR allows PRINTER_ALL_ACCESS to members of the Administrators group and PRINTER_ACCESS_USE to all other users.
- The name of the default data type for the printer.
- A list of clients to be notified by the server about printer changes. Each entry in this list contains the change notifications requested by one client (the notification filter settings from the client), the mechanism used to send notifications to this client, and the information about the client destination where the server sends these notifications. Clients can request printer change notifications using Notification Methods (section 3.1.4.10).

A print server initializes the list of printers to the persisted list of printers. The server does not persist the list of printers between server restarts.

Persisted List of Printers: A print server maintains in persistent storage a **persisted list of printers** that is identical to the **list of printers**. A print server updates the **persisted list of printers** whenever it updates the **list of printers**.

List of Printer Drivers: Each printer driver represents the software component responsible for converting print content submitted by applications into device-specific commands. Each printer driver object maintains the following data elements:

- A name that uniquely identifies the printer driver.
- A rendering module, a configuration module, and a data module.

  In addition, each printer driver object can maintain a list of Boolean values indicating whether the manufacturer of the printer driver declares it to have any of the following attributes, where a Boolean value of TRUE indicates that the printer driver has that attribute:
- The printer driver is a class printer driver.
- The printer driver is a derived printer driver.
- Printers using the printer driver cannot be shared from the print server to print clients.
- The printer driver is intended for use with fax printers.
- The printer driver is intended for use with file printers.

- The printer driver is intended for use with service printers.
- The printer driver is intended for use with virtual printers.

In addition, each printer driver object maintains the following optional data elements:

- A list of dependent files.
- Information about the printer driver manufacturer, timestamp, and driver version.<285>

Drivers are added, removed, enumerated, and managed using Printer Driver Management Methods (section 3.1.4.4).

A print server initializes the **list of printer drivers** to an empty list. The server persists the list of drivers between server restarts.

List of Core Printer Drivers: The server maintains a second list of printer drivers containing the core printer drivers installed on the **server**.

List of Language Monitors: A language monitor is a module that is tightly coupled to a printer driver and is part of the printer driver installation. It filters RAW format data as it is being sent from the printer driver to a port monitor, which then sends the data to the port. Language monitors add control information to the data stream, such as commands defined by a page description language (PDL). Language monitors are optional and are only associated with a particular type of printer if specified in the **INF file** for the printer.

Language monitors are added, removed, and enumerated using Port Monitor Management Methods (section 3.1.4.7).

A print server initializes the **list of language monitors** to an empty list. The server persists the **list of language monitors** between server restarts.

List of Port Monitors: A port monitor is a component that can send buffers of data to devices using supported protocols. Port monitors can manage extended communication with the device, such as collection status information from the device. Port monitors can expose zero or more ports.

Port monitor modules are implementation-specific for a given port type. A printer port identifies a device connected to the machine via an implementation-specific protocol understood by its parent port monitor.

Port monitors are added, removed, and enumerated using Port Monitor Management Methods (section 3.1.4.7).

A print server initializes the **list of port monitors** to an empty list.<286> The server persists the **list of port monitors** between server restarts.

List of Ports: A port represents a connection to an actual print device. Ports are exposed and managed by port monitors.

Each port monitor can persist the **list of ports** it manages between server restarts.

List of Print Providers: A print provider performs transparent routing of print system calls to a local or remote spooler. When RpcEnumPrinters (section 3.1.4.2.1) is called, an implementation-defined print provider name can be returned.<287>

A print server initializes the **list of print providers** to contain an implementation-specific set of included print providers. A print server persists the **list of print providers** between server restarts.<288>

List of Print Processors: Print processors, provided by device manufacturers or generic suppliers, perform additional manipulation of print content before it is sent to the device.

A print server persists the **list of print processors** between server restarts.

List of Known Printers: The server can maintain a **list of known printers** that includes printers not installed on the print server but installed on other print servers reachable on the network. The printers in this list are only used when composing a response for RpcEnumPrinters with the appropriate flags set, as specified in RpcEnumPrinters. This list facilitates printer detection in networks without directory services.

A print server does not persist the **list of known printers** between server restarts.

List of Warned Printer Drivers: The server can maintain a **list of printer drivers** that cannot be added through calls to RpcAddPrinterDriverEx (section 3.1.4.4.8) unless the appropriate flag is set, as specified in RpcAddPrinterDriverEx.<289>

If a print server maintains a **list of warned printer drivers**, it persists the **list of warned printer drivers** between server restarts.<290>

List of Notification Clients: A print server maintains a list of print clients that are notified upon server changes. Each entry in this list contains the change notifications requested by one client (the notification filter settings from the client), the mechanism used to send notifications to that client, and the information about the client destination where the server sends these notifications. Clients can request server change notifications by using Notification Methods (section 3.1.4.10).

A print server does not persist the **list of notification clients** between server restarts.

Job Named Properties: Each print job in a print queue can have zero or more **Job Named Properties** (also referred to as named properties or properties). The **Job Named Properties** are created, written, read, and deleted by the client. Each **Job Named Property** contains a name and a value. Unless deleted by the client, the **Job Named Properties** of a print job persist as long as the respective print job exists in the print queue.<291>

List of Per-Machine Connections: The server maintains a list of **shared printers** on other print servers. When a print server is used as a print client, all users logging on to the machine can use printers in this list without explicitly specifying the print server name or printer name of the shared printer.

A print server persists the **list of per-machine connections** between server restarts.

Branch Office Print Remote Log Entries: Each shared printer on a print server can be configured to operate in branch office print mode.<292> This mode enables a print client to print documents directly to a print device as defined by the print server, instead of routing print data to the server and then to the print device.

When in branch office print mode, if branch office print remote logging is enabled, the print client creates certain event channel entries on the print server in response to Windows Events while processing a print job. To do this, the client creates a structure that contains a **Branch Office Print Remote Log Entry** with the information needed to create the event ID in the correct event channel on the server, and it sends it to the server by using RpcLogJobInfoForBranchOffice (section 3.1.4.13.1).

If the print server cannot be contacted when the print job is being processed, the print client writes remote log entries to a **Branch Office Print Remote Log Offline Archive** (section 3.2.1) for transmission at a later time.

The abstract data model associates each printer with a single printer driver, one or more printer ports, and exactly one print processor. Every object stored in the abstract data model defines an associated set of attributes, as specified in IDL Data Types (section 2.2.1) and Custom-Marshaled Data Types (section 2.2.2).

Where the preceding data model requires persistence, and unless specified otherwise, the print server stores one persistent copy of each object in the registry.<293>

The server is responsible for ensuring consistency among persistently stored objects and their transient copies.<294> The server is also responsible for managing any resources, including memory, disk space, locks, and physical ports, that are used for object representations, throughout the lifetimes of the objects.

Note: The previous conceptual data can be implemented using a variety of techniques. A print server can implement such data as needed.

3.1.2 Timers

No protocol timer events are required on the server beyond timers for the underlying RPC protocol.

3.1.3 Initialization

The server SHOULD listen on the well-known endpoints defined for this RPC interface in section 2.1.

The server MUST perform initialization according to the following rules, when calling an RPC notification method on the client:

- Create an RPC binding handle to the server RPC endpoint (the client implements a server endpoint for this protocol in order to process notifications), or use an RPC context handle, as specified in [C706].
- Use RPC handles of the following types:
  - Context handles that are used across multiple calls to the client, for methods taking a PRINTER_HANDLE.
  - Handles that are bound to a single call to the client, for name-based methods taking a STRING_HANDLE. A **STRING_HANDLE_BIND** method MUST be implemented by the server.
- When creating the **RPC binding handle** on the named pipe **\pipe\spoolss**, the server MUST specify an **ImpersonationLevel** of 2 (**Impersonation**), as specified in [MS-SMB2], section 2.2.13.

If Active Directory is available, the server SHOULD perform according to the rules described in section 2.3.3.4 when starting additional initialization.

Both the server and the client MUST ignore all LDAP operation failures that occur during their initialization. For example, if LDAP initialization for one print queue fails, the server MUST ignore this failure and SHOULD continue to execute the remaining LDAP initializations for the remaining print queues.<295>

3.1.4 Message Processing Events and Sequencing Rules

The Print System Remote Protocol MUST indicate the following to the RPC **runtime** specified in [MS-RPCE] section 3.

- It is to perform a strict NDR data consistency check at target level 6.0.
- It is to reject a NULL unique or full pointer with non-zero conformant value.
- Using the **strict_context_handle** attribute, it is to reject the use of context handles that are created by the methods of a different RPC interface.

The methods that are defined by this protocol are grouped into functional categories, and their syntax and behavior are specified in sections, as shown in the following table.

Functional category	Description	Section
Printer management and discovery	Methods used for discovering and obtaining access to supported printers.	3.1.4.2
Job management	Methods for discovering, defining, and scheduling print jobs.	3.1.4.3
Printer driver management	Methods for discovering and installing printer drivers.	3.1.4.4
Form management	Methods for discovering and configuring printer forms.	3.1.4.5
Printer port management	Methods for discovering and communicating with printer ports.	3.1.4.6
Port monitor management	Methods for discovering and installing port monitor modules.	3.1.4.7
Print processor management	Methods for discovering and manipulating print processor objects.	3.1.4.8
Document printing	Methods for printing documents, pages and data.	3.1.4.9
Notifications	Methods for obtaining notifications of printing events.	3.1.4.10
Monitor modules	Methods specified by executable language monitors.	3.1.4.11
Job named property management	Methods for creating, updating, deleting, and enumerating **Job Named Properties** (section 3.1.1).<296>	3.1.4.12
Branch office print remote logging	Methods for processing **Branch Office Print Remote Log Entries** (section 3.1.1).<297>	3.1.4.13

The following table lists all the methods of the Print System Remote Protocol in ascending **opnum** order.

Methods in RPC Opnum Order

Method	Description
RpcEnumPrinters	RpcEnumPrinters enumerates available printers, print servers, domains, or print providers. Opnum: 0
RpcOpenPrinter	RpcOpenPrinter retrieves a handle for a printer, port, port monitor, print job, or print server. Opnum: 1
RpcSetJob	RpcSetJob pauses, resumes, cancels, or restarts a print job. It also sets print job parameters, for example, the job priority and the document name. Opnum: 2
RpcGetJob	RpcGetJob retrieves information about a specified print job. Opnum: 3
RpcEnumJobs	RpcEnumJobs retrieves information about a specified set of print jobs for a specified printer. Opnum: 4
RpcAddPrinter	RpcAddPrinter adds a printer to the list of supported printers for

Method	Description
	a specified server. Opnum: 5
RpcDeletePrinter	RpcDeletePrinter deletes the specified printer object. Opnum: 6
RpcSetPrinter	RpcSetPrinter sets the data for a specified printer or sets the state of the specified printer by pausing or resuming printing, or clearing all print jobs. Opnum: 7
RpcGetPrinter	RpcGetPrinter retrieves information about a specified printer. Opnum: 8
RpcAddPrinterDriver	RpcAddPrinterDriver installs a printer driver on the print server and links the configuration, data, and printer driver files. Opnum: 9
RpcEnumPrinterDrivers	RpcEnumPrinterDrivers enumerates the printer drivers installed on a specified print server. Opnum: 10
RpcGetPrinterDriver	RpcGetPrinterDriver retrieves printer driver data for the specified printer. Opnum: 11
RpcGetPrinterDriverDirectory	RpcGetPrinterDriverDirectory retrieves the path of the printer driver directory. Opnum: 12
RpcDeletePrinterDriver	RpcDeletePrinterDriver removes the specified printer driver from the list of supported drivers for a print server. Opnum: 13
RpcAddPrintProcessor	RpcAddPrintProcessor installs a print processor on the specified server and adds its name to an internal list of supported print processors. Opnum: 14
RpcEnumPrintProcessors	RpcEnumPrintProcessors enumerates the print processors installed on a specified server. Opnum: 15
RpcGetPrintProcessorDirectory	RpcGetPrintProcessorDirectory retrieves the path for the print processor on the specified server. Opnum: 16
RpcStartDocPrinter	RpcStartDocPrinter notifies the print spooler that a document is being spooled for printing. Opnum: 17
RpcStartPagePrinter	RpcStartPagePrinter notifies the spooler that a page is about to be printed on the specified printer. Opnum: 18
RpcWritePrinter	RpcWritePrinter sends data to the print spooler. Opnum: 19

Method	Description
RpcEndPagePrinter	RpcEndPagePrinter notifies the print spooler that the application is at the end of a page in a print job. Opnum: 20
RpcAbortPrinter	RpcAbortPrinter aborts the currently spooling print document. Opnum: 21
RpcReadPrinter	RpcReadPrinter retrieves data from the specified printer. Opnum: 22
RpcEndDocPrinter	RpcEndDocPrinter notifies the print spooler that the application is at the end of the current print job. Opnum: 23
RpcAddJob	RpcAddJob returns ERROR_INVALID_PARAMETER. Opnum: 24
RpcScheduleJob	RpcScheduleJob returns ERROR_SPL_NO_ADDJOB. Opnum: 25
RpcGetPrinterData	RpcGetPrinterData retrieves configuration data for a printer or print server. Opnum: 26
RpcSetPrinterData	RpcSetPrinterData sets the configuration data for a printer or print server. Opnum: 27
RpcWaitForPrinterChange	RpcWaitForPrinterChange retrieves information about the most recent change notification associated with a printer or print server. Opnum: 28
RpcClosePrinter	RpcClosePrinter closes a handle to a printer object, server object, job object, or port object. Opnum: 29
RpcAddForm	RpcAddForm adds a form name to the list of supported forms. Opnum: 30
RpcDeleteForm	RpcDeleteForm removes a form name from the list of supported forms. Opnum: 31
RpcGetForm	RpcGetForm retrieves information about a specified form. Opnum: 32
RpcSetForm	RpcSetForm replaces the form information for the specified form. Opnum: 33
RpcEnumForms	RpcEnumForms enumerates the forms that the specified printer supports. Opnum: 34
RpcEnumPorts	RpcEnumPorts enumerates the ports that are available for printing on a specified server. Opnum: 35

Method	Description
RpcEnumMonitors	RpcEnumMonitors retrieves information about the port monitors installed on the specified server. Opnum: 36
Opnum37NotUsedOnWire	Reserved for local use. Opnum: 37
Opnum38NotUsedOnWire	Reserved for local use. Opnum: 38
RpcDeletePort	RpcDeletePort removes a port added by the RpcAddPortEx method. Opnum: 39
RpcCreatePrinterIC	RpcCreatePrinterIC is called by the **Graphics Device Interface (GDI)** to create an information context for a printer. Opnum: 40
RpcPlayGdiScriptOnPrinterIC	RpcPlayGdiScriptOnPrinterIC returns identifying information for fonts available for printing to a printer object. Opnum: 41
RpcDeletePrinterIC	RpcDeletePrinterIC deletes a printer information context (IC). Opnum: 42
Opnum43NotUsedOnWire	Reserved for local use. Opnum: 43
Opnum44NotUsedOnWire	Reserved for local use. Opnum: 44
Opnum45NotUsedOnWire	Reserved for local use. Opnum: 45
RpcAddMonitor	RpcAddMonitor installs a local port monitor and links the configuration, data, and monitor files. Opnum: 46
RpcDeleteMonitor	RpcDeleteMonitor removes a port monitor. Opnum: 47
RpcDeletePrintProcessor	RpcDeletePrintProcessor removes a print processor. Opnum: 48
Opnum49NotUsedOnWire	Reserved for local use. Opnum: 49
Opnum50NotUsedOnWire	Reserved for local use. Opnum: 50
RpcEnumPrintProcessorDatatypes	RpcEnumPrintProcessorDatatypes enumerates the data types that a specified print processor supports. Opnum: 51
RpcResetPrinter	RpcResetPrinter resets the data type and device mode values to use for printing documents submitted by RpcStartDocPrinter (section 3.1.4.9.1).

Method	Description
	Opnum: 52
RpcGetPrinterDriver2	RpcGetPrinterDriver2 retrieves printer driver data for the specified printer. Opnum: 53
Opnum54NotUsedOnWire	Reserved for local use. Opnum: 54
Opnum55NotUsedOnWire	Reserved for local use. Opnum: 55
RpcFindClosePrinterChangeNotification	RpcFindClosePrinterChangeNotification closes a change notification object created by calling RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3). The printer or print server associated with the change notification object will no longer be monitored by that object. Opnum: 56
Opnum57NotUsedOnWire	Reserved for local use. Opnum: 57
RpcReplyOpenPrinter	RpcReplyOpenPrinter establishes a context handle from the server to the client. The server uses the context handle to send notification data to the client machine. Opnum: 58
RpcRouterReplyPrinter	RpcRouterReplyPrinter handles the notification coming from a remote router, as opposed to one coming from a print provider. Opnum: 59
RpcReplyClosePrinter	RpcReplyClosePrinter closes the notification channel opened by the RpcReplyOpenPrinter (section 3.2.4.1.1) method between the server and client. Opnum: 60
RpcAddPortEx	RpcAddPortEx adds a port name to the list of supported ports. Opnum: 61
RpcRemoteFindFirstPrinterChangeNotification	RpcRemoteFindFirstPrinterChangeNotification creates a remote change notification object that monitors changes to printer objects and sends change notifications to the client using the method RpcRouterReplyPrinter (section 3.2.4.1.2). Opnum: 62
Opnum63NotUsedOnWire	Reserved for local use. Opnum: 63
Opnum64NotUsedOnWire	Reserved for local use. Opnum: 64
RpcRemoteFindFirstPrinterChangeNotificationEx	RpcRemoteFindFirstPrinterChangeNotificationEx creates a remote change notification object that monitors changes to printer objects and sends change notifications to the client using the method RpcRouterReplyPrinterEx (section 3.2.4.1.4). Opnum: 65
RpcRouterReplyPrinterEx	RpcRouterReplyPrinterEx handles the notification coming from a remote router, as opposed to one coming from a print provider.

Method	Description
	It is similar to RpcRouterReplyPrinter but an RPC_V2_UREPLY_PRINTER structure to specify the set of notifications that the client requested. Opnum: 66
RpcRouterRefreshPrinterChangeNotification	RpcRouterRefreshPrinterChangeNotification returns change notification information. Opnum: 67
Opnum68NotUsedOnWire	Reserved for local use. Opnum: 68
RpcOpenPrinterEx	RpcOpenPrinterEx retrieves handle for a printer, port, port monitor, print job, or print server. This method is similar to RpcOpenPrinter but takes a pointer to an SPLCLIENT_CONTAINER (section 2.2.1.2.14) structure, which contains information about the connecting client. Opnum: 69
RpcAddPrinterEx	RpcAddPrinterEx installs a printer on the print server. This method is similar to RpcAddPrinter but takes a pointer to an SPLCLIENT_CONTAINER structure, which contains information about the connecting client. Opnum: 70
RpcSetPort	RpcSetPort sets the status associated with a printer port. Opnum: 71
RpcEnumPrinterData	RpcEnumPrinterData enumerates configuration data for a specified printer. Opnum: 72
RpcDeletePrinterData	RpcDeletePrinterData deletes specified configuration data for a printer. Opnum: 73
Opnum74NotUsedOnWire	Reserved for local use. Opnum: 74
Opnum75NotUsedOnWire	Reserved for local use. Opnum: 75
Opnum76NotUsedOnWire	Reserved for local use. Opnum: 76
RpcSetPrinterDataEx	RpcSetPrinterDataEx sets the configuration data for a printer or print server. This method is similar to RpcSetPrinterData but also allows the caller to specify the registry key under which to store the data. Opnum: 77
RpcGetPrinterDataEx	RpcGetPrinterDataEx retrieves configuration data for the specified printer or print server. This method extends RpcGetPrinterData and can retrieve values sorted under a specified key by RpcSetPrinterDataEx. Opnum: 78
RpcEnumPrinterDataEx	RpcEnumPrinterDataEx enumerates all value names and data for a specified printer and key. This method extends

Method	Description
	RpcEnumPrinterData by retrieving several values in a single call. Opnum: 79
RpcEnumPrinterKey	RpcEnumPrinterKey enumerates the subkeys of a specified key for a specified printer. Printer data is stored in the registry. Opnum: 80
RpcDeletePrinterDataEx	RpcDeletePrinterDataEx deletes specified configuration data for a printer. This method is similar to RpcDeletePrinterData but accesses the configuration data using a set of named and typed values that are stored in a hierarchy of registry keys. Opnum: 81
RpcDeletePrinterKey	RpcDeletePrinterKey deletes a specified key and all of its subkeys for a specified printer. Opnum: 82
Opnum83NotUsedOnWire	Reserved for local use. Opnum: 83
RpcDeletePrinterDriverEx	RpcDeletePrinterDriverEx removes the specified printer driver from the list of supported drivers for a print server and deletes the files associated with the driver. This method is similar to RpcDeletePrinterDriver but can also delete specific versions of the driver. Opnum: 84
RpcAddPerMachineConnection	RpcAddPerMachineConnection stores the print server name and the name of the binary executable used as a provider for a particular connection. Opnum: 85
RpcDeletePerMachineConnection	RpcDeletePerMachineConnection deletes information about a server and connection provider. Opnum: 86
RpcEnumPerMachineConnections	RpcEnumPerMachineConnections enumerates each of the connections and copies PRINTER_INFO_4 (section 2.2.1.10.5) structures for all the per-machine connections to the buffer **pPrinterEnum**. Opnum: 87
RpcXcvData	RpcXcvData provides an extensible mechanism by which a client can control ports on the server and exchange port-specific commands and data with the server. See section 3.1.4.11 for details on language monitor methods. Opnum: 88
RpcAddPrinterDriverEx	RpcAddPrinterDriverEx installs a printer driver on the print server. This method performs a function similar to RpcAddPrinterDriver and additionally allows options to be specified which permit **printer driver upgrade**, **printer driver downgrade**, copying of newer files only, and copying of all files regardless of their time stamps. Opnum: 89
Opnum90NotUsedOnWire	Reserved for local use. Opnum: 90

Method	Description
Opnum91NotUsedOnWire	Reserved for local use. Opnum: 91
Opnum92NotUsedOnWire	Reserved for local use. Opnum: 92
Opnum93NotUsedOnWire	Reserved for local use. Opnum: 93
Opnum94NotUsedOnWire	Reserved for local use. Opnum: 94
Opnum95NotUsedOnWire	Reserved for local use. Opnum: 95
RpcFlushPrinter	RpcFlushPrinter is used by the printer driver to send a buffer of bytes to the port to cleanly abort a print job. It also allows delaying the I/O line to the printer. Opnum: 96
RpcSendRecvBidiData	RpcSendRecvBidiData sends and receives bidirectional data. This method is used to communicate with port monitors that support such data. Opnum: 97
Opnum98NotUsedOnWire	Reserved for local use. Opnum: 98
Opnum99NotUsedOnWire	Reserved for local use. Opnum: 99
Opnum100NotUsedOnWire	Reserved for local use. Opnum: 100
Opnum101NotUsedOnWire	Reserved for local use. Opnum: 101
Opnum102NotUsedOnWire	Reserved for local use. Opnum: 102
RpcGetCorePrinterDrivers	RpcGetCorePrinterDrivers gets the GUIDs, versions, and publish dates of the specified core printer drivers, and the paths to their packages. Opnum: 102
Opnum103NotUsedOnWire	Reserved for local use. Opnum: 103
Opnum104NotUsedOnWire	Reserved for local use. Opnum: 104
RpcGetPrinterDriverPackagePath	RpcGetPrinterDriverPackagePath gets the path to the specified printer driver package. Opnum: 104
Opnum105NotUsedOnWire	Reserved for local use. Opnum: 105

Method	Description
Opnum106NotUsedOnWire	Reserved for local use. Opnum: 106
Opnum107NotUsedOnWire	Reserved for local use. Opnum: 107
Opnum108NotUsedOnWire	Reserved for local use. Opnum: 108
Opnum109NotUsedOnWire	Reserved for local use. Opnum: 109
RpcGetJobNamedPropertyValue	**RpcGetJobNamedPropertyValue** retrieves the value of the specified **Job Named Property** (section 3.1.1) for the specified print job. Opnum: 110
RpcSetJobNamedProperty	**RpcSetJobNamedProperty** creates a new **Job Named Property** or changes the value of an existent **Job Named Property** for the specified print job. Opnum: 111
RpcDeleteJobNamedProperty	**RpcDeleteJobNamedProperty** deletes a **Job Named Property** for the specified print job. Opnum: 112
RpcEnumJobNamedProperties	**RpcEnumJobNamedProperties** enumerates the **Job Named Properties** for the specified print job. Opnum: 113
Opnum114NotUsedOnWire	Reserved for local use. Opnum: 114
Opnum115NotUsedOnWire	Reserved for local use. Opnum: 115
RpcLogJobInfoForBranchOffice	**RpcLogJobInfoForBranchOffice** processes one or more **Branch Office Print Remote Log Entries** (section 3.1.1). Opnum: 116

In the preceding table, the term "Reserved for local use" means that the client MUST NOT send the opnum, and server behavior is undefined since it does not affect interoperability.<298>

All these methods are request/response RPC methods. They MUST return zero to indicate successful completion and nonzero values to indicate failure, except where specifically described.

Unless stated otherwise, if a method fails for any reason, returning a nonzero failure value, the server state as visible to the client through this or any other protocol MUST NOT change.

Two nonzero return codes have specific meanings in this protocol, ERROR_MORE_DATA and ERROR_INSUFFICIENT_BUFFER, as specified in [MS-ERREF]. When a method declaration in this specification has an output parameter that supplies a needed buffer size, one of the values in the following table can be returned from a call to that method to enable the caller to discover that size. Thus, there are circumstances in which a nonzero return value SHOULD NOT be treated as an error, but, instead, the client SHOULD allocate a buffer with a larger size and retry the request. These cases are noted in the method definitions in this section.

Error name/code	Meaning
ERROR_INSUFFICIENT_BUFFER 0x0000007A	The data area passed to a system call is too small.
ERROR_MORE_DATA 0x000000EA	More data is available.

3.1.4.1 Commonly Used Parameters

This section describes parameters commonly used in method calls with consistent definitions. The type of each parameter is given by the method declaration in the relevant method sections.

Individual method sections specify only parameters whose definitions are different from, or that are not listed in, this section; however, they may impose additional restrictions on the values of parameters defined in this section.

3.1.4.1.1 Datatype Name Parameters

pDatatype: This parameter MUST be one of the following:

- NULL, to indicate that the default data type for the printer MUST be used.
- A pointer to a string that specifies the data type to be associated with the printer handle.

For rules governing data type names, see section 2.2.4.2.

The individual method sections include the following parameter validation steps by reference:

- If *pDatatype* is not NULL, verify that the string that is referenced by the *pDatatype* parameter identifies one of the data types that the printer or print server supports, and if that verification fails, return ERROR_INVALID_DATATYPE.

Note: A client SHOULD use RpcEnumPrintProcessorDatatypes (section 3.1.4.8.5), specifying a print processor, to obtain a list of supported data types.

3.1.4.1.2 Dynamically Typed Query Parameters

Unless notified otherwise, methods returning one or more dynamically-typed values use a common API pattern, in which the caller passes in the following parameters:

- *BUFFER*: A buffer into which the server copies the requested dynamically-typed values. The term "BUFFER" is used here as a placeholder. Each method section defines which of its parameters is the pointer to the buffer.
- *pType*: An optional pointer to a variable that receives a code that indicates the type of data that is stored in the specified value. For a list of possible type values, see section 2.2.3.9.
- *nSize* or *cbData*: The size, in bytes, of the buffer. This value can be larger than the required size for the requested dynamically-typed values.
- *pcbNeeded* or *pcbData*: A pointer to a variable into which the server copies the number of bytes between the start of *BUFFER* and the last byte written by the server into *BUFFER*, both inclusive; or the required size of the buffer, in bytes, if the value of the buffer size parameter is smaller than the size of the data to return to the caller.

For methods capable of returning more than one dynamically-typed value, the caller also passes in:

- *pcReturned*: A pointer to a variable into which the server copies the number of dynamically-typed values that were written to the buffer, if the buffer was large enough to hold them. If the buffer is not large enough to hold these dynamically typed values, the client SHOULD ignore the value that the server can return through pcReturned.

The individual method sections include the following parameter validation steps by reference:

- The server MUST verify that the value of the buffer size parameter is not smaller than the number of bytes required to hold the dynamically-typed values to be written to the buffer. If that verification fails, the server MUST write the number of bytes required into the variable that is pointed to by *pcbNeeded* and return ERROR_MORE_DATA.
- If the value of the buffer size parameter is not zero, the server MUST verify that a non-NULL pointer to the buffer was passed in. If that verification fails, the server MUST return ERROR_INVALID_USER_BUFFER.

The individual method sections include the following processing and response steps by reference:

- The server MUST populate *BUFFER* with dynamically-typed values enumerated according to method-specific enumeration steps.
- If *pType* is not a NULL pointer, the server MUST write the type of the data returned in *BUFFER* to the variable pointed to by *pType*.
- For methods capable of returning more than one dynamically-typed value, the server MUST store the number of values that were written to *BUFFER* into the variable pointed to by *pcReturned*.
- The server MUST return zero for success or a nonzero error code if the method was not successful.

Except for diagnostic purposes, the server state as visible to the client through this protocol MUST NOT change as a result of processing the method call.

3.1.4.1.3 Environment Name Parameters

pEnvironment: This parameter MUST either be NULL or a pointer to a string that MUST specify the environment name. For rules governing environment names and Windows behaviors, see section 2.2.4.4.

The individual method sections include the following parameter validation steps by reference:

- If *pEnvironment* is NULL, use the local environment of the print server.
- If *pEnvironment* is a non-NULL pointer, verify that the string that is referenced by the *pEnvironment* parameter identifies an environment name that is supported on this server, and if that verification fails, return ERROR_INVALID_ENVIRONMENT.

3.1.4.1.4 Print Server Name Parameters

pName: This parameter is a pointer to a string that specifies the name of the print server that the method operates on. This MUST be a Domain Name System (DNS), **NetBIOS**, Internet Protocol version 4 (IPv4), Internet Protocol version 6 (IPv6), or Universal Naming Convention (UNC) name that remote procedure call (RPC) binds to, and it MUST uniquely identify a print server on the network.

For all methods taking a STRING_HANDLE custom binding handle parameter, the Print System Remote Protocol assumes that the bind routine provided by the client uses the name provided through the *pName* parameter to create the **RPC binding**, although that is not strictly necessary from an RPC perspective. Although it is possible to create an **RPC binding** to a different server than that identified by the *pName* parameter, the Print System Remote Protocol has not been designed and tested for that usage pattern. However, server implementations MAY choose to implement support for server names

not identical to the server name used to create the **RPC binding**, and as a result effectively route the call to another server.<299>

Note: Regardless of the preceding statement, server implementations MUST NOT assume that the server name passed via the *pName* parameter matches the name the server determined upon its own initialization; the server name passed in MAY differ from that name as a result of server name aliasing, for example, by use of **DNS** names or directory services. The server MUST use the passed-in name to compose names for responses because the client is not aware that aliasing occurred.

RPC binding handles are specified in [C706]. For rules governing server names, see section 2.2.4.16.

pServer: Synonymous with *pName*.

pszServer: Synonymous with *pName*.

Name: Synonymous with *pName*.

The individual method sections include the following parameter validation steps by reference:

- Verify that the string pointed to by the *Name* parameter is well-formed according to the rules governing server names (section 2.2.4.16). If that verification fails, return the error code ERROR_INVALID_NAME.
- Verify that the string pointed to by the *Name* parameter corresponds to a server name. If that verification fails, return any of the following error codes: ERROR_INVALID_NAME, ERROR_INVALID_PARAMETER, or ERROR_INVALID_PRINTER_NAME.

  The server SHOULD perform this validation step to ensure correctness with clients that do not derive the RPC binding directly from the STRING_HANDLE parameter.<300>

3.1.4.1.5 Printer Name Parameters

pPrinterName: This parameter is a pointer to a string that specifies the name of the printer connection, printer object, server object, job object, or port object. The string that is referenced by this parameter MUST NOT be empty. For rules governing printer names, see section 2.2.4.14.

The individual method sections include the following parameter validation steps by reference:

- Verify that the string pointed to by *pPrinterName* is well-formed according to the rules governing printer names. If that verification fails, return the error code ERROR_INVALID_PRINTER_NAME. When the string pointed to by *pPrinterName* contains a printer name postfix string appended at the end, the server SHOULD:
  - Ignore both the comma character preceding the postfix string and the postfix string.
  - Use only the printer name specified by the prefix string.
  - Validate the prefix string according to the rules governing printer names.

  The server can execute additional implementation-specific validation of the postfix string, including rejecting unsupported postfix string values by returning an implementation-specific error code.
- For server names, verify that the server name portion of the string corresponds to a server name. If that verification fails, return any of the following error codes: ERROR_INVALID_NAME, ERROR_INVALID_PARAMETER, or ERROR_INVALID_PRINTER_NAME.

  The server SHOULD perform this validation step to ensure correctness with clients that do not derive the RPC binding directly from the STRING_HANDLE parameter.<301>

- For printer, job, or port names, verify that the remainder of the string corresponds to a printer, job, or port name. If that verification fails, return the error code ERROR_INVALID_PRINTER_NAME.
- For port and port monitor, open requests by using RpcOpenPrinter or RpcOpenPrinterEx, and verify that the port monitor supports all the optional methods: **XcvOpenPort**, **XcvDataPort**, and **XcvClosePort**. And if that verification fails, return ERROR_INVALID_PRINT_MONITOR.

3.1.4.1.6 Standard Parameter Validation

The implementation MUST apply the following validation rules to all parameters unless more specific statements appear in the individual method sections.

Term used to describe parameter	Required validation
X MUST be a non-NULL pointer to a string.	Verify that X is not NULL. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST be A.	Verify that X is A. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST be a value from A through B, inclusive.	Verify that X is a value that is greater than or equal to A and less than or equal to B. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST NOT be A.	Verify that X is a value that is not A. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST be one of <list>.	Verify that X is a value that is a member of <list>. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST be the result of a bitwise OR of zero or more of the flags in <list>.	If <list> contains the statement "All other bits MUST be zero", verify that the only bits that are set are those that are specified in <list>. If that verification fails, return ERROR_INVALID_PARAMETER.
X MUST be the result of a bitwise OR of one or more of the flags in <list>.	Verify that at least one of the bit flags from <list> is set and if that verification fails, return ERROR_INVALID_PARAMETER. If <list> contains the statement "All other bits MUST be zero", verify that the only bits that are set are the bits that are specified in <list>. If that verification fails, return ERROR_INVALID_PARAMETER.

3.1.4.1.7 String Query Parameters

Unless noted otherwise, methods that return one or more string values use a common API pattern, in which the caller passes in the following parameters:

- *Level*: The value 0x00000001, if this parameter is present in the method signature.
- *BUFFER*: A buffer into which the server copies the requested string values. The term BUFFER is used here as a placeholder. Each method section defines which of its parameters is the pointer to the buffer. Methods capable of returning more than one string value MUST write the values to the buffer as a multisz.
- *cbBuf*: The size, in bytes, of the buffer. This value can be larger than the required size for the requested string values.
- *pcbNeeded*: A pointer to a variable into which the server copies the number of bytes between the start of the buffer and the last byte written by the server into the buffer, both inclusive, or the

required size of the buffer, in bytes, if the value of the *cbBuf* parameter is smaller than the actual size of the data to return to the caller.

For methods that are capable of returning more than one string value, the following parameter interpretation applies:

- If a *pcReturned* parameter is present in the method signature, it is a pointer to a variable into which the server copies the actual number of string values that are written to *BUFFER*, if the buffer is large enough to hold them.
- If a *pcReturned* parameter is not present in the method signature, the caller MUST interpret the data returned in *BUFFER* as a multisz.

The individual method sections include the following parameter validation steps by reference:

- The server MUST verify that the value of *cbBuf* is not smaller than the number of bytes required to hold the string values to be written to the buffer. If that verification fails, the server MUST write the number of bytes required to the variable pointed to by the *pcbNeeded* parameter and return ERROR_INSUFFICIENT_BUFFER.
- If the value of the *cbBuf* parameter is not zero, the server MUST verify that a non-NULL pointer to the buffer was passed in. If that verification fails, the server MUST return ERROR_INVALID_USER_BUFFER.

The individual method sections include the following processing and response steps by reference:

- The server MUST populate *BUFFER* with string values enumerated according to method-specific enumeration steps. Multiple string values MUST be represented as a multisz.
- For methods that are capable of returning more than one string value, if a *pcReturned* parameter is present in the method signature, the server MUST store the number of string values written to *BUFFER* into the variable pointed to by *pcReturned*.
- The server MUST return zero for success, or a nonzero error code if the method was not successful.

Except for diagnostic purposes, the server state as visible to the client through this protocol MUST NOT change as a result of processing the method call.

3.1.4.1.8 CONTAINER Parameters

This section specifies common CONTAINER parameters and related validation and processing requirements.

3.1.4.1.8.1 DEVMODE_CONTAINER Parameters

pDevMode: This parameter is a pointer to a DEVMODE_CONTAINER (section 2.2.1.2.1) structure.

pDevModeContainer: This parameter is synonymous with *pDevMode*.

The individual method sections include the following parameter validation steps by reference:

- If the *pDevModeContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, the server SHOULD skip the validation steps and assume validation success.
- If the *pDevModeContainer* parameter is not declared with the "unique" IDL attribute, the server MAY verify that its value is not NULL.<302>
- The server SHOULD verify that the *pDevMode* member of the DEVMODE_CONTAINER that is pointed to by *pDevModeContainer* is NULL, or that the _DEVMODE (section 2.2.2.1) structure that is pointed to by the *pDevMode* member is valid, which means that the total size specified in

_DEVMODE MUST be less than or equal to the number of bytes specified by the value of the **cbBuf** member of the DEVMODE_CONTAINER.

- The server SHOULD verify that the **dmSize** and **dmDriverExtra** members of the _DEVMODE structure comply with the constraints defined in section 2.2.2.1, and that the sum of their values is not larger than the *cbBuf* member of the DEVMODE_CONTAINER.

  Note: The server uses the printer driver associated with a print queue to validate all other _DEVMODE members. Incorrect settings for these other members are silently corrected by the printer driver. Therefore, the print client SHOULD obtain a valid _DEVMODE from the printer driver or print queue to use as template for the DEVMODE_CONTAINER parameter. The client SHOULD modify selected fields only as necessary and as indicated by the printer driver capabilities that are returned by a local call to the printer driver.

Unless noted otherwise, if any of the preceding validation steps fail, the server SHOULD return ERROR_INVALID_PARAMETER.

3.1.4.1.8.2 DOC_INFO_CONTAINER Parameters

pDocInfoContainer: This parameter is a pointer to a DOC_INFO_CONTAINER (section 2.2.1.2.2) structure that specifies the document to print.

The individual method sections include the following parameter validation steps by reference:

- If the *pDocInfoContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- If the *pDocInfoContainer* parameter is not declared with the "unique" IDL attribute, the server MAY verify that its value is not NULL.<303>
- Verify that the value of the **Level** member in the DOC_INFO_CONTAINER is 0x00000001.
- Verify that the **pDocInfo1** pointer in the DOC_INFO_CONTAINER is not NULL.
- Verify that all members of the structure that is pointed to by the **pDocInfo1** pointer in the DOC_INFO_CONTAINER comply with the constraints defined in section 2.2.1.2.2.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.3 DRIVER_CONTAINER Parameters

pDriverContainer: This parameter is a pointer to a DRIVER_CONTAINER (section 2.2.1.2.3) structure that specifies printer driver information. The value of the **Level** member of the DRIVER_CONTAINER structure MUST be 0x00000002, 0x00000003, 0x00000004, 0x00000006, or 0x00000008.

The individual method sections include the following parameter validation steps by reference:

- If the *pDriverContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that *pDriverContainer* points to a DRIVER_CONTAINER structure that specifies an appropriate level, as defined in the referring method definition. If that verification fails, return ERROR_INVALID_LEVEL.
- Verify that, within the DRIVER_INFO (section 2.2.1.5) structure that is contained in DRIVER_CONTAINER, the **pEnvironment** member specifies an environment name that is supported on the server (see section 2.2.4.4). If that verification fails, return ERROR_INVALID_ENVIRONMENT.

- Verify that all members of DRIVER_CONTAINER comply with the constraints defined in section 2.2.1.2.3.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.4 FORM_CONTAINER Parameters

pFormInfoContainer: This parameter is a pointer to a FORM_CONTAINER (section 2.2.1.2.4) structure that specifies form information.

The individual method sections include the following parameter validation steps by reference:

- If the *pFormInfoContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that *pFormInfoContainer* points to a FORM_CONTAINER that specifies an appropriate level as defined in the referring method definition. If that verification fails, return ERROR_INVALID_LEVEL.
- Verify that all members of the FORM_CONTAINER structure comply with the constraints defined in section 2.2.1.2.4.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.5 PORT_CONTAINER Parameters

pPortContainer: This parameter is a pointer to PORT_CONTAINER (section 2.2.1.2.7) structure that specifies port information.

The individual method sections include the following parameter validation steps by reference:

- If the *pPortContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that *pPortContainer* points to a PORT_CONTAINER that specifies an appropriate level as defined in the referring method definition. If that verification fails, return ERROR_INVALID_LEVEL.
- Verify that all members of the PORT_CONTAINER structure comply with the constraints specified in section 2.2.1.2.7.

Unless noted otherwise, if any of the above validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.6 PRINTER_CONTAINER Parameters

pPrinterContainer: This parameter is a pointer to a PRINTER_CONTAINER (section 2.2.1.2.9) structure, which specifies printer information. The **Level** member of the PRINTER_CONTAINER MUST be between 0x00000000 and 0x00000008, inclusive. When the **Level** member is 0x00000002, the **Status**, **cJobs**, and **AveragePPM** members of the PRINTER_INFO_2 structure (sections 2.2.1.3.6 and 2.2.1.10.3) MUST be set to zero by the caller and MUST be ignored on receipt. For details concerning PRINTER_INFO structures, see section 2.2.1.10.

The individual method sections include the following parameter validation steps by reference:

- If the *pPrinterContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.

- Verify that *pPrinterContainer* points to a PRINTER_CONTAINER that specifies an appropriate level as defined in the referring method definition. If that verification fails, return ERROR_INVALID_LEVEL.
- If *pDatatype* is not NULL, verify that it points to a string that identifies a data type supported by the associated print processor. If that verification fails, return ERROR_INVALID_DATATYPE.
- If *pPrintProcessor* is not NULL, verify that it points to a string that identifies a print processor. If that verification fails, return ERROR_UNKNOWN_PRINTPROCESSOR.
- If *pSepFile* is not NULL, verify that it points to a string that names an existing file on the server. If that verification fails, return ERROR_INVALID_SEPARATOR_FILE.
- Verify that *pPortName* is not NULL and points to a string that identifies an existing port on the server. If that verification fails, return ERROR_UNKNOWN_PORT.
- Verify that *pDriverName* is not NULL and points to a string that identifies an existing printer driver on the server. If that verification fails, return ERROR_UNKNOWN_PRINTER_DRIVER.
- Verify that, if the *Attributes* parameter has the flag PRINTER_ATTRIBUTE_SHARED (section 2.2.3.12) set, the printer driver identified by *pDriverName* does not have the attribute that printers using that printer driver cannot be shared (section 3.1.1). If that verification fails, return ERROR_PRINTER_NOT_SHAREABLE.
- Verify that the **Priority** member of PRINTER_INFO_2 is within the range specified in section 2.2.1.10.3. If that verification fails, return ERROR_INVALID_PRIORITY.
- Verify that all members of the PRINTER_CONTAINER structure comply with the constraints defined in section 2.2.1.10, with the exception of **pServerName**, which SHOULD be ignored.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

The individual method sections further include the following parameter processing steps by reference:

- If the value of the **Level** member specifies a PRINTER_INFO structure that contains a **pDevMode** member (section 2.2.1.3.6), replace the value of that member with the **pDevMode** value of the DEVMODE_CONTAINER structure (section 2.2.1.2.1) that is pointed to by the *pDevModeContainer* parameter of the method.
- If the value of the **Level** member specifies a PRINTER_INFO structure that contains a **pSecurityDescriptor** member (section 2.2.1.3.6), replace the value of that member with the **pSecurity** value of the SECURITY_CONTAINER structure (2.2.1.2.13) that is pointed to by the *pSecurityContainer* parameter of the method.

3.1.4.1.8.7 SECURITY_CONTAINER Parameters

pSecurityContainer: This parameter is a pointer to a SECURITY_CONTAINER (section 2.2.1.2.13) structure that specifies security information and components. The created printer MUST allow security access based on this information.<304>

The individual method sections include the following parameter validation steps by reference:

- If the *pSecurityContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that the **pSecurity** member of the SECURITY_CONTAINER structure is NULL, or that it points to a well-formed SECURITY_DESCRIPTOR in self-relative form ([MS-DTYP] section 2.4.6).

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.8 SPLCLIENT_CONTAINER Parameters

pClientInfo: This parameter is a pointer to an SPLCLIENT_CONTAINER (section 2.2.1.2.14) structure that specifies client information. The **Level** member of the SPLCLIENT_CONTAINER structure MUST be 0x00000001.<305>

The individual method sections include the following parameter validation steps by reference:

- If the *pClientInfo* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that *pClientInfo* points to an SPLCLIENT_CONTAINER that contains a pointer to a structure, and that all members of that structure comply with the constraints defined in section 2.2.1.2.14.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.8.9 MONITOR_CONTAINER Parameters

pMonitorContainer: This parameter is a pointer to a MONITOR_CONTAINER (section 2.2.1.2.6) structure that specifies monitor information. The individual method sections include the following parameter validation steps by reference:

- If the *pMonitorContainer* parameter is declared with the "unique" IDL attribute, and its value is NULL, skip the validation steps and assume validation success.
- Verify that *pMonitorContainer* points to a MONITOR_CONTAINER that specifies an appropriate level as defined in the referring method definition. If that verification fails, return ERROR_INVALID_LEVEL.
- If the value of the **Level** member specifies a MONITOR_INFO structure that contains a **pEnvironment** member, verify that the string that is referenced by **pEnvironment** identifies an environment name that is supported on this server; and if that verification fails, return ERROR_INVALID_ENVIRONMENT.
- If the value of the **Level** member specifies a **MONITOR_INFO** structure that contains a **pEnvironment** member, the environment name referenced by **pEnvironment** is "Windows ARM", and this server doesn't support that environment name (see section 2.2.4.4), return ERROR_NOT_SUPPORTED. Print servers can optionally be configured to not perform this validation step.<306>
- Verify that all members of the MONITOR_CONTAINER structure comply with the constraints defined in 2.2.1.8.

Unless noted otherwise, if any of the preceding validation steps fail, return ERROR_INVALID_PARAMETER.

3.1.4.1.9 INFO Structures Query Parameters

Unless noted otherwise, methods that return one or more of the INFO structures (see sections 2.2.1.3 to 2.2.1.11) use a common API pattern, in which the caller passes in the following parameters:

- *Level*: The desired information level of the INFO structures, if this parameter is present in the method signature.
- *BUFFER*: A buffer into which the server copies the requested INFO structures. The term "BUFFER" is used here as a placeholder. Each method section defines which of its parameters contains the pointer to the buffer.

- *cbBuf*: The size, in bytes, of the buffer. The value of *cbBuf* can be larger than the required size for the requested INFO structures.
- *pcbNeeded*: A pointer to a variable into which the server copies the number of bytes between the start of *BUFFER* and the last byte written by the server into *BUFFER*, both inclusive; or the required size of the buffer, in bytes, if the value of *cbBuf* is smaller than the size of the data to return to the caller.

For methods capable of returning more than one INFO structure, the caller also passes in:

- *pcReturned*: This parameter is a pointer to a variable into which the server copies the number of INFO structures written to the buffer, if the buffer was large enough to hold them.

The individual method sections use the following documentation conventions:

- **BUFFER TYPE**: The type of INFO structures returned, which is one of the following:
  - DATATYPES_INFO_1
  - _DRIVER_INFO
  - _FORM_INFO
  - _JOB_INFO
  - _MONITOR_INFO
  - _PORT_INFO
  - _PRINTER_INFO
  - PRINTPROCESSOR_INFO_1
- *Level*: The valid levels of INFO structures.

The individual method sections include the following parameter validation steps by reference:

- The server MUST verify that *Level* is valid, and if that verification fails, the server MUST return ERROR_INVALID_LEVEL.
- The server MUST verify that the value of *cbBuf* is not smaller than the number of bytes required to hold the INFO structures to be written to the buffer, and if that verification fails, the server MUST write the number of bytes required into the variable pointed to by *pcbNeeded* and return ERROR_INSUFFICIENT_BUFFER.
- If the value of *cbBuf* is zero, the server MUST ignore the pointer to the buffer that was passed in. If the value of *cbBuf* is not zero, the server MUST verify that the pointer to the buffer that was passed in is a non-NULL pointer, and if that verification fails, the server MUST return ERROR_INVALID_USER_BUFFER.

The individual method sections include the following processing and response steps by reference:

- The server MUST populate *BUFFER* with INFO structures of a type specified by **TYPE** that describe the objects enumerated according to method-specific enumeration steps.
- For methods capable of returning more than one INFO structure, the server MUST store the number of INFO structures that it writes to *BUFFER* in the variable pointed to by *pcReturned*.
- The server MUST return zero for success or a nonzero error code if the method was not successful.

Except for diagnostic purposes, the server state as visible to the client through this protocol MUST NOT change as a result of processing the method call.

3.1.4.1.10 PRINTER_ENUM_VALUES Structures Query Parameters

Unless noted otherwise, methods returning one or more PRINTER_ENUM_VALUES structures (section 2.2.2.11) use a common API pattern, in which the caller passes in the following parameters:

- *BUFFER*: A buffer into which the server copies the requested PRINTER_ENUM_VALUES structures. The term "BUFFER" is used here as a placeholder; each method section defines which of its parameters contains a pointer to the buffer.
- *cbEnumValues*: The size, in bytes, of *BUFFER*. The value of *cbEnumValues* can be larger than the required size for the requested PRINTER_ENUM_VALUES structures.
- *pcbEnumValues*: A pointer to a variable into which the server copies the number of bytes between the start of *BUFFER* and the last byte written by the server into *BUFFER*, both inclusive; or the required size of the buffer, in bytes, if the value of *cbEnumValues* is smaller than the size of the data to return to the caller.

For methods capable of returning more than one PRINTER_ENUM_VALUES structure, the caller also passes in:

- *pnEnumValues*: A pointer to a variable into which the server copies the number of PRINTER_ENUM_VALUES structures written to the buffer, if the buffer was large enough to hold them.

The individual method sections include the following parameter validation steps by reference:

- The server MUST verify that the value of *cbEnumValues* is not smaller than the number of bytes required by the PRINTER_ENUM_VALUES structures to be written to the buffer. If that verification fails, the server MUST write the number of bytes required to the variable that is pointed to by *pcbEnumValues* and return ERROR_MORE_DATA.
- If the value of *cbEnumValues* is not zero, the server MUST verify that a non-NULL pointer to the buffer was passed in. If that verification fails, the server MUST return ERROR_INVALID_USER_BUFFER.
- For a printer object with a printer driver version (see *cVersion* in section 2.2.1.3.1) of 0x00000004, the server SHOULD verify that the size in bytes of the data to be returned to the client (the value to be returned via the *pcbEnumValues* parameter) does not exceed 0xFFFFFFFF bytes. If this verification fails, the server SHOULD return ERROR_NOT_ENOUGH_MEMORY. <307>

The individual method sections include the following processing and response steps by reference:

- The server MUST populate *BUFFER* with PRINTER_ENUM_VALUES structures that describe the enumerated objects according to method-specific enumeration steps.
- For methods capable of returning more than one PRINTER_ENUM_VALUES structure, the server MUST write the number of PRINTER_ENUM_VALUES structures that were written to *BUFFER* into the variable pointed to by the *pnEnumValues* parameter.
- The server MUST return zero for success or a nonzero error code if the method was not successful.

Except for diagnostic purposes, the server state as visible to the client through this protocol MUST NOT change as a result of processing the method call.

3.1.4.1.11 PRINTER_HANDLE Parameters

hPrinter: An RPC context handle ([C706]) to an object managed by the server. The individual method sections describe which methods are used to obtain the handle and which types of object (printer object, server object, port object, or job object) are acceptable.

The individual method sections include the following parameter validation steps by reference:

- Verify that *hPrinter* is an RPC context handle to an object managed by the server.
- Verify that *hPrinter* is an RPC context handle to an appropriate object as defined in the referring method definition.
- Unless noted otherwise, if the preceding validation steps fail, return ERROR_INVALID_PARAMETER as specified in [MS-ERREF].

3.1.4.2 Printer Management and Discovery Methods

This section specifies methods for discovering and obtaining access to supported printers.

Method	Description
RpcEnumPrinters	RpcEnumPrinters enumerates available printers, print servers, domains, or print providers. Opnum 0
RpcOpenPrinter	RpcOpenPrinter retrieves a handle for a printer, port, port monitor, print job, or print server. Opnum 1
RpcAddPrinter	RpcAddPrinter adds a printer to the list of supported printers for a specified server. Opnum 5
RpcDeletePrinter	RpcDeletePrinter deletes the specified printer object. Opnum 6
RpcSetPrinter	RpcSetPrinter sets the data for a specified printer or sets the state of the specified printer by pausing or resuming printing or clearing all print jobs. Opnum 7
RpcGetPrinter	RpcGetPrinter retrieves information about a specified printer. Opnum 8
RpcGetPrinterData	RpcGetPrinterData retrieves printer configuration data for a printer or print server. Opnum 26
RpcSetPrinterData	RpcSetPrinterData sets the configuration data for a printer or print server. Opnum 27
RpcClosePrinter	RpcClosePrinter closes a handle to a printer object, server object, job object, or port object. Opnum 29
RpcCreatePrinterIC	RpcCreatePrinterIC called by the Graphics Device Interface (GDI) to create an information context for a printer. Opnum 40
RpcPlayGdiScriptOnPrinterIC	RpcPlayGdiScriptOnPrinterIC returns identifying information for fonts available for printing to a printer object. Opnum 41
RpcDeletePrinterIC	RpcDeletePrinterIC deletes a printer information context. Opnum 42

Method	Description
RpcResetPrinter	RpcResetPrinter resets the data type and device mode values to use for printing documents submitted by the RpcStartDocPrinter method (see section 3.1.4.9.1). Opnum 52
RpcOpenPrinterEx	RpcOpenPrinterEx retrieves a handle for a printer, port, port monitor, print job, or print server. Opnum 69
RpcAddPrinterEx	RpcAddPrinterEx installs a printer on the print server. Opnum 70
RpcEnumPrinterData	RpcEnumPrinterData enumerates configuration data for a specified printer. Opnum 72
RpcDeletePrinterData	RpcDeletePrinterData deletes specified configuration data for a printer. Opnum 73
RpcSetPrinterDataEx	RpcSetPrinterDataEx sets the configuration data for a printer or print server. This method extends RpcSetPrinterData (see section 3.1.4.2.8), but by additionally allowing the caller to specify the registry key under which to store the data. Opnum 77
RpcGetPrinterDataEx	RpcGetPrinterDataEx retrieves configuration data for the specified printer or print server. This method extends RpcGetPrinterData (see section 3.1.4.2.7) and can retrieve values sorted under a specified key by RpcSetPrinterDataEx (see section 3.1.4.2.18). Opnum 78
RpcEnumPrinterDataEx	RpcEnumPrinterDataEx enumerates all value names and data for a specified printer and key. This method extends RpcEnumPrinterData (see section 3.1.4.2.16) by retrieving several values in a single call. Opnum 79
RpcEnumPrinterKey	RpcEnumPrinterKey enumerates the subkeys of a specified key for a specified printer. Printer data is stored in the registry. Opnum 80
RpcDeletePrinterDataEx	RpcDeletePrinterDataEx deletes a specified value from a printer's configuration data, which consists of a set of named and typed values stored in a hierarchy of registry keys. Opnum 81
RpcDeletePrinterKey	RpcDeletePrinterKey deletes a specified key and all of its subkeys for a specified printer. Opnum 82
RpcAddPerMachineConnection	RpcAddPerMachineConnection stores the print server name and the name of the binary executable used as a provider for a particular connection. Opnum 85
RpcDeletePerMachineConnection	RpcDeletePerMachineConnection deletes information about a server and connection provider. Opnum 86
RpcEnumPerMachineConnections	RpcEnumPerMachineConnections enumerates each connection and copies PRINTER_INFO_4 structures (see section 2.2.1.10.5) for all the per-machine

Method	Description
	connections into the buffer *pPrinterEnum*. Opnum 87
RpcSendRecvBidiData	RpcSendRecvBidiData sends and receives bidirectional data. This method is used to communicate with port monitors that support such data. Opnum 97

3.1.4.2.1 RpcEnumPrinters (Opnum 0)

RpcEnumPrinters enumerates available printers, print servers, domains, or print providers.

```
DWORD RpcEnumPrinters(
  [in] DWORD Flags,
  [in, string, unique] STRING_HANDLE Name,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPrinterEnum,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

Flags: The types of print objects that this method enumerates. The value of this parameter MUST be the result of a bitwise OR of one or more of the Printer Enumeration Flags (section 2.2.3.7), with the following additional specifications:

- **PRINTER_ENUM_NAME**: If the *Name* parameter is NULL or points to an empty string, and the *Level* parameter value is 0x00000001, available print providers SHOULD be enumerated. If this flag is not set, the server SHOULD ignore the *Name* parameter.
- **PRINTER_ENUM_REMOTE**: The *Level* parameter value MUST be 0x00000001.
- **PRINTER_ENUM_NETWORK**: The *Level* parameter value MUST be 0x00000001.

Name: NULL or a server name parameter as specified in Printer Server Name Parameters (section 3.1.4.1.4). If the *Flags* parameter contains the **PRINTER_ENUM_NAME** flag, the *Name* parameter value controls where the server SHOULD enumerate. The server SHOULD enumerate locally if the *Name* parameter is either NULL or an empty string; otherwise, remotely on the server whose name is specified by the *Name* string.

Level: The level of printer information structure, as follows.

Value	Meaning
0x00000000	_PRINTER_INFO_STRESS (section 2.2.2.9.1)
0x00000001	_PRINTER_INFO_1 (section 2.2.2.9.2)
0x00000002	_PRINTER_INFO_2 (section 2.2.2.9.3)
0x00000004	_PRINTER_INFO_4 (section 2.2.2.9.5)
0x00000005	_PRINTER_INFO_5 (section 2.2.2.9.6)

pPrinterEnum: A pointer to a BUFFER defined in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _PRINTER_INFO.

This parameter can be NULL if the value of *cbBuf* is zero.

cbBuf: An input parameter that MUST adhere to the specification in INFO Structures Query Parameters.

pcbNeeded: An output parameter that MUST adhere to the specification in INFO Structures Query Parameters.

pcReturned: An output parameter that MUST adhere to the specification in INFO Structures Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server SHOULD<308> validate parameters as follows:

- Perform validation steps as specified in Print Server Name Parameters (section 3.1.4.1.4).
- Perform validation steps as specified in INFO Structures Query Parameters (section 3.1.4.1.9).
- If the **PRINTER_ENUM_NETWORK** or **PRINTER_ENUM_REMOTE** flag is set, verify that the value of the *Level* parameter is 0x00000001. Otherwise, the server SHOULD<309> return ERROR_INVALID_LEVEL ([MS-ERREF] section 2.2).
- Additional validation MAY<310> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the value of the *Level* parameter is 0x00000001 and the **PRINTER_ENUM_NETWORK** bit is set in the *Flags* parameter, the server SHOULD enumerate all printers from the "List of Known Printers" (section 3.1.1). <311>

  If the server does not maintain a list of known printers, or if the list has not contained at least one entry for an implementation-specific period of time, the server MAY<312> return ERROR_CAN_NOT_COMPLETE ([MS-ERREF] section 2.2).<313>
- For any other validated values for the *Level* and *Flags* parameters, the server SHOULD enumerate all printers in the "List of Printers" (section 3.1.1) on the print server or print provider that comply with the value of the *Flags* parameter. This information SHOULD be restricted for security reasons.<314>
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters (section 3.1.4.1.9).
- Return the status of the operation.

3.1.4.2.2 RpcOpenPrinter (Opnum 1)

RpcOpenPrinter retrieves a handle for a printer, port, port monitor, print job, or print server.

```
DWORD RpcOpenPrinter(
  [in, string, unique] STRING_HANDLE pPrinterName,
  [out] PRINTER_HANDLE* pHandle,
  [in, string, unique] wchar_t* pDatatype,
  [in] DEVMODE_CONTAINER* pDevModeContainer,

  [in] DWORD AccessRequired
```

```
);
```

pPrinterName: A STRING_HANDLE (section 2.2.1.1.7) for a printer connection, printer object, server object, job object, port object, or port monitor object. For opening a server object, this parameter MUST adhere to the specification in Print Server Name Parameters (section 3.1.4.1.4); for opening all other objects, it MUST adhere to the specification in Printer Name Parameters (section 3.1.4.1.5).

pHandle: A pointer to a PRINTER_HANDLE (section 2.2.1.1.4) that MUST receive the RPC context handle [C706] to the object identified by the *pPrinterName* parameter.

pDatatype: A pointer to a string that specifies the data type to be associated with the printer handle. This parameter MUST adhere to the specification in Datatype Name Parameters (section 3.1.4.1.1).

pDevModeContainer: A pointer to a DEVMODE_CONTAINER structure. This parameter MUST adhere to the specification in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

AccessRequired: The access level that the client requires for interacting with the object to which a handle is being opened. The value of this parameter MUST be one of those specified in Access Values (section 2.2.3.1) or 0. For rules governing access values, see section 2.2.4.1.

If AccessRequired is set to 0 (if no specific access level is requested), the server MUST assume a GENERIC_READ (see section 2.2.3.1) access level.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- For opening a server object, perform the validation steps specified in Print Server Name Parameters; for opening all other objects, perform the validation steps specified in Printer Name Parameters.
- Perform the validation steps that are specified in Datatype Name Parameters (section 3.1.4.1.1).
- Perform the validation steps that are specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).
- Verify that the client issuing the call has authorization equivalent to the value of the *AccessRequired* parameter.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Locate one of the following objects that corresponds to the request:
  - The printer in the "List of Printers", or a job queued for that printer
  - The server in the "List of Print Server Names"
  - The port monitor in the "List of Port Monitors"
  - The port in the "List of Ports".

  These lists are specified in section 3.1.1.
- Create an implementation-specific representation of the printer, server, job, port monitor, or port ("the object") that MUST include:

- A remote procedure call (RPC) handle, which is a snapshot of the printer, server, job, port monitor, or port data that is specific to this instance of the invocation.
- The data from *pDataType* and *pDevModeContainer*, if they were not NULL.
- All other relevant, implementation-specific data required to process all other protocol methods passing in a PRINTER_HANDLE.

- Store the RPC handle for the object in the variable pointed to by *pHandle*.
- Increment the reference count of the object to prevent deletion.
- Return the status of the operation.

3.1.4.2.3 RpcAddPrinter (Opnum 5)

RpcAddPrinter adds a printer to the list of supported printers for a specified server.

```
DWORD RpcAddPrinter(
  [in, string, unique] STRING_HANDLE pName,
  [in] PRINTER_CONTAINER* pPrinterContainer,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] SECURITY_CONTAINER* pSecurityContainer,
  [out] PRINTER_HANDLE* pHandle
);
```

pName: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pPrinterContainer: A parameter specified in PRINTER_CONTAINER Parameters (section 3.1.4.1.8.6). The **Level** member of the PRINTER_CONTAINER MUST be 0x00000001 or 0x00000002.

pDevModeContainer: A parameter specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

pSecurityContainer: A parameter specified in SECURITY_CONTAINER Parameters (section 3.1.4.1.8.7).

pHandle: A pointer to a variable that receives the printer RPC context handle to the printer object added. RPC context handles are specified in [C706].

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform validation steps as specified in Print Server Name Parameters (section 3.1.4.1.4).
- Perform validation steps as specified in PRINTER_CONTAINER Parameters (section 3.1.4.1.8.6).
- Perform validation steps as specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).
- Perform validation steps as specified in SECURITY_CONTAINER Parameters (section 3.1.4.1.8.7).
- If the **Level** member of the PRINTER_CONTAINER is 0x00000002:
  - Verify that the printer driver specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by *pPrinterContainer* already exists in the system, and if that verification fails, return ERROR_UNKNOWN_PRINTER_DRIVER, as specified in [MS-ERREF].

- Verify that the port specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by *pPrinterContainer* already exists in the system, and if that verification fails, return ERROR_UNKNOWN_PORT, as specified in [MS-ERREF].
- Verify that the print processor specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by *pPrinterContainer* already exists in the system, and if that verification fails, return ERROR_UNKNOWN_PRINTPROCESSOR, as specified in [MS-ERREF].
- Verify that the printer with the name specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by *pPrinterContainer* does not already exist in the system, and if that verification fails, return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF].

- Additional validation MAY<315> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Perform PRINTER_CONTAINER parameter processing steps as specified in PRINTER_CONTAINER Parameters (section 3.1.4.1.8.6).
- If the value of the **Level** member of the PRINTER_CONTAINER is 0x00000001, and if the server does not maintain a "List of Known Printers", the server MUST return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF]. Otherwise, the server MUST continue to process the message and compose a response to the client as follows:
  - If the PRINTER_ATTRIBUTE_SHARED bit is set in the **Flags** member of the PRINTER_INFO_1 structure, add the printer<316> to the "List of Known Printers" as specified in section 3.1.1.
  - If PRINTER_ATTRIBUTE_SHARED bit is not set in the **Flags** member of the PRINTER_INFO_1 structure, remove the printer from the "List of Known Printers".
  - Store NULL in the output parameter pointed to by *pHandle*.
  - Increment the number of network printers added to this server.
  - Return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF].

    Note: An error return code is required by remote procedure call (RPC) because NULL was stored to the output parameter pointed to by *pHandle*.
- If the **Level** member of the PRINTER_CONTAINER is 0x00000002:
  - Create the printer object and assign to it the security descriptor from the SECURITY_CONTAINER that is pointed to by *pSecurityContainer* parameter.
  - Add the printer to the "List of Printers" (section 3.1.1).
  - Create a session that includes:
    - An RPC handle
    - A snapshot of the printer data specific to this instance of the printer invocation.
    - The data from the DEVMODE that is contained in the DEVMODE_CONTAINER pointed to by the *pDevModeContainer* parameter if it is not NULL.
  - Store the RPC handle for the session in the output parameter pointed to by *pHandle*. The handle returned from this method MUST be granted PRINTER_ALL_ACCESS permission.

- Increment the printer's reference count to prevent deletion.
- If there are any clients that are registered for notifications on the server object change, a notification MUST be sent to those clients.

- Return the status of the operation.

3.1.4.2.4 RpcDeletePrinter (Opnum 6)

RpcDeletePrinter is a method that deletes the specified printer object.

```
DWORD RpcDeletePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: This parameter MUST specify a handle to a printer object that MUST have been opened by using the RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, or RpcOpenPrinterEx methods.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps as specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- If any jobs are pending on the printer, use the implementation-specific policy<317> to determine if a delete operation can be made pending or if an error should be returned.

Additional validation MAY<318> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Mark the printer object as "Delete Pending".
- Modify the list of printers in the system to exclude the deleted printer for any subsequent calls to RpcEnumPrinters, RpcOpenPrinter, RpcOpenPrinterEx, and RpcStartDocPrinter. Clients that already have a valid handle to the same printer object MAY continue using the printer object for all operations except RpcStartDocPrinter.
- If the deleted printer has been published to the directory service, delete the print queue object from the directory as described in section 2.3.3.2.<319> If the directory service operation fails, the print server MUST continue processing the printer deletion operation and MUST NOT return the status of the directory service operation to the client.
- If any clients have registered for notifications of the server object change, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.2.5 RpcSetPrinter (Opnum 7)

RpcSetPrinter sets the data or state of a specified printer by pausing or resuming printing or by clearing all print jobs.

```
DWORD RpcSetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] PRINTER_CONTAINER* pPrinterContainer,
```

```
    [in] DEVMODE_CONTAINER* pDevModeContainer,
    [in] SECURITY_CONTAINER* pSecurityContainer,
    [in] DWORD Command
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pPrinterContainer: A parameter specified in PRINTER_CONTAINER Parameters (section 3.1.4.1.8.6). If the *Command* is 0, the **Level** member of the PRINTER_CONTAINER MUST be 0x00000000 or a number from 0x00000002 to 0x00000007. If the *Command* parameter is 1, 2, or 3, the **Level** member of the PRINTER_CONTAINER MUST be 0x00000000.

pDevModeContainer: A parameter specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

pSecurityContainer: A parameter specified in SECURITY_CONTAINER Parameters (section 3.1.4.1.8.7).

Command: A value that specifies an action to perform. If this value is non-zero, it MUST be one of the values in the following table, and the **Level** member of the PRINTER_CONTAINER (section 2.2.1.2.9) structure that is pointed to by the *pPrinterContainer* parameter MUST be 0x00000000.

Printer Control Name/Value	Meaning
PRINTER_CONTROL_PAUSE 0x00000001	Pauses the printer object.
PRINTER_CONTROL_RESUME 0x00000002	Resumes a paused printer object.
PRINTER_CONTROL_PURGE 0x00000003	Deletes all print jobs queued for the printer object.

If this value is zero, the **PrinterInfo** member of the PRINTER_CONTAINER structure that is pointed to by the *pPrinterContainer* parameter MUST contain a pointer to a PRINTER_INFO (section 2.2.2.9) structure that this method can use. See section 2.2.1.10.1 for details.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Perform the validation steps specified in PRINTER_CONTAINER Parameters.
- Verify that the information provided in the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter is consistent with the value in *Command*, according to the following table:

Command	Level in PRINTER_CONTAINER
0x00000000	MUST be 0x00000000 or a number from 0x00000002 to 0x00000007, inclusive.

Command	Level in PRINTER_CONTAINER
0x00000001	MUST be 0x00000000.
0x00000002	MUST be 0x00000000.
0x00000003	MUST be 0x00000000.

If this verification fails, return ERROR_INVALID_LEVEL.

- Perform the validation steps specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).
- Perform the validation steps specified in SECURITY_CONTAINER Parameters (section 3.1.4.1.8.7).
- If the **Level** member of the PRINTER_CONTAINER is 0x00000007 and the **dwAction** field in the **PRINTER_INFO_7** structure that is pointed to by the **pPrinterInfo7** member of the PRINTER_CONTAINER pointed to by **pPrinterContainer** is DSPRINT_UPDATE (0x00000002):
  - Verify that *hPrinter* specifies a printer object that is already published in the directory service. If *hPrinter* represents a printer that is not already published in the directory service, the server MUST fail the call by returning an HRESULT (see [MS-ERREF] section 2.1) code of 0x80070002 with the same meaning as the ERROR_FILE_NOT_FOUND error specified in [MS-ERREF].
- Additional validation SHOULD<320> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If *hPrinter* specifies a server object, the server MUST only apply the SECURITY_CONTAINER parameter to set the print server's security descriptor, and MUST not perform the remaining processing steps that follow.
- Perform PRINTER_CONTAINER parameter processing steps that are specified in PRINTER_CONTAINER Parameters.
- Perform the operation from the following table that corresponds to the value of the *Command* parameter.

Command parameter values	Operation that MUST be performed
0x0000000 No command	Update the printer configuration using the settings in *pPrinterContainer*.<321>
0x00000001 Pause the printer	Temporarily suspend sending data to the printer without changing the state of any jobs associated with the printer. Clients are allowed to continue adding data to the job.
0x00000002 Resume the printer	Resume sending data to the printer without changing the state of any jobs associated with the printer.
0x00000003 Purge the printer	Remove all jobs that are currently associated with the printer and mark them as having failed to print.

- If *hPrinter* specifies a printer object that the print server has published in the Active Directory, and as a result of this call the print server has changed the printer driver associated with the printer or changed printer configuration information that the print server has published in the Active Directory, update the print queue object in the directory as described in section 2.3.3.2 with attributes obtained from the print queue configuration and the printer driver as described in section 2.3.3.1.<322> If the directory service operation fails, the print server MUST continue

processing the current call and MUST NOT return the status of the directory service operation to the client.

- If any clients registered for notifications of the printer object change, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.2.6 RpcGetPrinter (Opnum 8)

RpcGetPrinter retrieves information about a specified printer.

```
DWORD RpcGetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPrinter,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14). This value MAY be a handle to a print server object.<323>

Level: The level of printer information structure, as follows.

Value	Meaning
0x00000000	Corresponds to _PRINTER_INFO_STRESS (section 2.2.2.9.1).
0x00000001	Corresponds to _PRINTER_INFO_1 (section 2.2.2.9.2).
0x00000002	Corresponds to _PRINTER_INFO_2 (section 2.2.2.9.3).
0x00000003	Corresponds to _PRINTER_INFO_3 (section 2.2.2.9.4).
0x00000004	Corresponds to _PRINTER_INFO_4 (section 2.2.2.9.5).
0x00000005	Corresponds to _PRINTER_INFO_5 (section 2.2.2.9.6).
0x00000006	Corresponds to _PRINTER_INFO_6 (section 2.2.2.9.7).
0x00000007	Corresponds to _PRINTER_INFO_7 (section 2.2.2.9.8).
0x00000008	Corresponds to _PRINTER_INFO_8 (section 2.2.2.9.9).
0x00000009	Not valid remotely; the server MUST respond by returning ERROR_NOT_SUPPORTED.

pPrinter: A pointer to a BUFFER (INFO Structures Query Parameters (section 3.1.4.1.9)).

BUFFER TYPE: _PRINTER_INFO.

cbBuf: A parameter specified in INFO Structures Query Parameters.

pcbNeeded: A parameter specified in INFO Structures Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Perform the validation steps specified in INFO Structures Query Parameters.
- Additional validation MAY<324> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Using information about the printer, perform the processing and response steps specified in INFO Structures Query Parameters.
- Return the status of the operation.

3.1.4.2.7 RpcGetPrinterData (Opnum 26)

RpcGetPrinterData retrieves printer configuration data for a printer or print server.

```
DWORD RpcGetPrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName,
  [out] DWORD* pType,
  [out, size_is(nSize)] BYTE* pData,
  [in] DWORD nSize,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pValueName: A pointer to a string that identifies the configuration data to get. For rules governing value names, see section 2.2.4.18.

For print servers, the value name MUST be one of the predefined strings listed in Server Handle Key Values (section 2.2.3.10).

For printer objects, the value name MAY be one of the predefined strings listed in Printer Data Values (section 2.2.3.11). Also, the value name "ChangeID"<325> is reserved by the protocol and has a special meaning. It identifies a read-only value that specifies that a change identifier MUST be returned in the buffer pointed to by *pData*. This identifier is a **DWORD** that MUST be set by the print server to a new, unique value each time printer information changes. The client SHOULD use the change identifier to decide if it has stale information about a printer object, in which case it SHOULD call this method or RpcGetPrinter (section 3.1.4.2.6) to update its view of the printer object. Only the fact that the *pData* buffer value changes is significant; the change identifier value itself is arbitrary. If the value name is not one of these predefined strings, it is an arbitrary string defined by the printer driver associated with the printer object or by client applications.

pType: A parameter specified in Dynamically Typed Query Parameters (section 3.1.4.1.2).

pData: A pointer to BUFFER as specified in Dynamically Typed Query Parameters.

This parameter can be NULL if *nSize* equals zero

nSize: A parameter specified in Dynamically Typed Query Parameters.

pcbNeeded: A parameter specified in Dynamically Typed Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- The server MUST perform the validation steps that are specified in Dynamically Typed Query Parameters.
- For server objects, the server MUST verify that the *pValueName* parameter points to a string that is one of the predefined value names listed in Server Handle Key Values (section 2.2.3.10). If this verification fails, return ERROR_INVALID_PARAMETER.
- For printer objects, the server MUST verify that, if the *pValueName* parameter points to a string that is one of the predefined value names listed in Printer Data Values (section 2.2.3.11), the print server supports retrieving the value for this printer according to the rules in section 2.2.3.11. If this verification fails, return ERROR_NOT_SUPPORTED.
- For printer objects, the server MAY verify that the *pValueName* parameter points to a string that complies with the rules specified in section 2.2.4.18.
- The server MUST NOT perform access checks on the *hPrinter* object.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- With the data identified by *pValueName*, perform the processing and response steps<326> specified in Dynamically Typed Query Parameters.
- Return the status of the operation.

3.1.4.2.8 RpcSetPrinterData (Opnum 27)

RpcSetPrinterData sets the configuration data for a printer or print server.

```
DWORD RpcSetPrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName,
  [in] DWORD Type,
  [in, size_is(cbData)] BYTE* pData,
  [in] DWORD cbData
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pValueName: A pointer to a string that identifies the configuration data to set. For rules governing value names, see section 2.2.4.18.

For print servers, the value name MUST be one of the predefined strings listed in Server Handle Key Values (section 2.2.3.10).

For printer objects, the value name is an arbitrary string defined by the printer driver associated with the printer object. The value name "ChangeID"<327> is reserved by the protocol and MAY NOT be used in a call to RpcSetPrinterData.

Type: The type value for data pointed to by the *pData* parameter. This value SHOULD be one of the type codes defined in section 2.2.3.9. For rules governing registry type values, see section 2.2.4.15.

pData: A pointer to an array of bytes that contain the printer configuration data. The type of the data in the buffer is specified by the *Type* parameter.

cbData: The size, in bytes, of the *pData* array. This value SHOULD NOT be zero.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11),.
- For server objects, the server MUST verify that the *pValueName* parameter points to a string that is one of the predefined value names listed in Server Handle Key Values with the "read-write" column selected. If this verification fails, return ERROR_INVALID_PARAMETER.
- For printer objects, the server MAY verify that the *pValueName* parameter points to a string that complies with the rules specified in section 2.2.4.18.
- Additional validation SHOULD<328> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Set the printer data<329> associated with *pValueName* to the data pointed to by *pData*.
- Return the status of the operation.

3.1.4.2.9 RpcClosePrinter (Opnum 29)

RpcClosePrinter closes a handle to a printer object, server object, job object, or port object.

```
DWORD RpcClosePrinter(
  [in, out] PRINTER_HANDLE* phPrinter
);
```

phPrinter: A pointer to a handle to a printer object, server object, job object, or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object, server object, job object, or port object can be used without further access checks.

  If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client.

Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the object is a printer object, and RpcStartDocPrinter has been called without a matching RpcEndDocPrinter, the same processing as for RpcEndDocPrinter MUST occur.

  If there is an active notification context associated with the object, as a result of the client not calling RpcFindClosePrinterChangeNotification, the server MUST close the notification context now by calling the client's RpcReplyClosePrinter method.

- Free any internal state that is associated with the handle that is pointed to by *phPrinter*.
- Store NULL in the variable that is pointed to by *phPrinter*.
- Decrement the reference count on that object.
- If the object is a printer object marked as "Delete Pending", and the usage count is zero, the following steps MUST be performed:
  - Handle any pending jobs in an implementation-specific manner.
  - Clear references to this printer from any other data structures.
  - Delete the printer object.
- Return the status of the operation.

3.1.4.2.10 RpcCreatePrinterIC (Opnum 40)

RpcCreatePrinterIC is called by the Graphics Device Interface (GDI) to create an information context for a specified printer.

```
DWORD RpcCreatePrinterIC(
  [in] PRINTER_HANDLE hPrinter,
  [out] GDI_HANDLE* pHandle,
  [in] DEVMODE_CONTAINER* pDevModeContainer
);
```

hPrinter: A handle to a printer object (section 2.2.1.1.4) that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pHandle: A pointer to a printer information context handle (section 2.2.1.1.2).

pDevModeContainer: A parameter specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Perform the validation steps specified in DEVMODE_CONTAINER Parameters.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Perform implementation-specific steps to create the appropriate printer information context.

- Store an RPC context handle associated with the information context in *pHandle*.
- Return the status of the operation.

Except for diagnostic purposes, the server state, as visible to the client through this or any other protocol, MUST NOT change as a result of processing this call.

3.1.4.2.11 RpcPlayGdiScriptOnPrinterIC (Opnum 41)

RpcPlayGdiScriptOnPrinterIC returns font information for a printer connection. UNIVERSAL_FONT_ID (section 2.2.2.12) structures are used to identify the fonts.

```
DWORD RpcPlayGdiScriptOnPrinterIC(
  [in] GDI_HANDLE hPrinterIC,
  [in, size_is(cIn)] BYTE* pIn,
  [in] DWORD cIn,
  [out, size_is(cOut)] BYTE* pOut,
  [in] DWORD cOut,
  [in] DWORD ul
);
```

hPrinterIC: A printer information context handle (section 2.2.1.1.2) that was returned by RpcCreatePrinterIC (section 3.1.4.2.10).

pIn: A pointer that SHOULD be set to NULL when sent and MUST be ignored on receipt.

cIn: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

pOut: A pointer to a buffer, the size and contents of which are determined by the value of the *cOut* parameter.

cOut: The size, in bytes, of the buffer pointed to by *pOut*.

ul: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Verify that *hPrinterIC* is a valid printer information context handle. This method SHOULD assume that this handle can be used without further access checks.<330>
- Verify that the value of the *pOut* parameter is not NULL.<331>
- Verify the value of the *cOut* parameter as follows:
  - If *cOut* is less than 0x00000004, ERROR_NOT_ENOUGH_MEMORY SHOULD be returned, as specified in [MS-ERREF].
  - If *cOut* is equal to 0x00000004, proceed.
  - If *cOut* is greater than 0x00000004, it specifies the size of the buffer pointed to by the *pOut* parameter. In this case, the minimum value of *cOut* is computed as follows:

```
((*pOut) * (size of(UNIVERSAL_FONT_ID))) + 0x00000004
```

The buffer pointed to by *pOut* MUST be large enough to contain all the font information plus a DWORD for the number of fonts. If the value of *cOut* is less than this minimum, ERROR_NOT_ENOUGH_MEMORY SHOULD be returned.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If *cOut* is equal to 0x00000004, the value of *pOut* is a pointer to a DWORD that specifies the number of UNIVERSAL_FONT_ID structures to be returned by the next call to this method.
- If *cOut* is greater than 0x00000004, font information MUST be returned as follows:

  1. Query the fonts that are available on the server.
  2. Write the DWORD number of fonts to the buffer location that is pointed to by the value of the *pOut* parameter.
  3. Write UNIVERSAL_FONT_ID structures for the fonts to the buffer location that is pointed to by the value of *pOut* plus 0x00000004.

  A print client MAY assume that the fonts identified by the UNIVERSAL_FONT_ID structures are available on the print server for use in spooled print jobs.<332>
- Return the status of the operation.

Except for diagnostic purposes, the server state, as visible to the client through this or any other protocol, MUST NOT change as a result of processing this call.

3.1.4.2.12 RpcDeletePrinterIC (Opnum 42)

RpcDeletePrinterIC deletes a printer information context.

```
DWORD RpcDeletePrinterIC(
  [in, out] GDI_HANDLE* phPrinterIC
);
```

phPrinterIC: A pointer to a printer information context handle (section 2.2.1.1.2) that was returned by RpcCreatePrinterIC (section 3.1.4.2.10).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST verify that the handle pointed to by the *phPrinterIC* parameter is associated with an information context. This method SHOULD assume that this handle can be used without further access checks.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the printer information context.
- Store NULL in the variable pointed to by *phPrinterIC*.
- Return the status of the operation.

Except for diagnostic purposes, the server state, as visible to the client through this or any other protocol, MUST NOT change as a result of processing this call.

3.1.4.2.13 RpcResetPrinter (Opnum 52)

RpcResetPrinter resets the data type and device mode (For more information, see [DEVMODE]) values to use for printing documents submitted by the RpcStartDocPrinter (section 3.1.4.9.1) method.

```
DWORD RpcResetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] wchar t* pDatatype,
  [in] DEVMODE_CONTAINER* pDevModeContainer
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pDatatype: A parameter specified in Datatype Name Parameters (section 3.1.4.1.1).

pDevModeContainer: A parameter specified in DEVMODE_CONTAINER Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Perform the validation steps specified in Datatype Name Parameters.
- Perform the validation steps specified in DEVMODE_CONTAINER Parameters.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Update the default data type that is associated with the context for *hPrinter*.
- Update the default DEVMODE structure that is associated with the context for *hPrinter*.
- Return the status of the operation.

3.1.4.2.14 RpcOpenPrinterEx (Opnum 69)

RpcOpenPrinterEx retrieves a handle for a printer, port, port monitor, print job, or print server.<333>

```
DWORD RpcOpenPrinterEx(
  [in, string, unique] STRING_HANDLE pPrinterName,
  [out] PRINTER HANDLE* pHandle,
  [in, string, unique] wchar t* pDatatype,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] DWORD AccessRequired,
  [in] SPLCLIENT CONTAINER* pClientInfo
);
```

pPrinterName: A STRING_HANDLE (section 2.2.1.1.7) for a printer connection, printer object, server object, job object, port object, or port monitor object. For opening a server object, this parameter MUST adhere to the specification in Print Server Name Parameters (section 3.1.4.1.4); for opening

all other objects, it MUST adhere to the specification in Printer Name Parameters (section 3.1.4.1.5).

pHandle: A pointer to a PRINTER_HANDLE (section 2.2.1.1.4) that MUST receive the RPC context handle [C706] to the object identified by the *pPrinterName* parameter.

pDatatype: A pointer to a string that specifies the data type to be associated with the printer handle. This parameter MUST adhere to the specification in Datatype Name Parameters (section 3.1.4.1.1).

pDevModeContainer: A pointer to a DEVMODE_CONTAINER structure. This parameter MUST adhere to the specification in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

AccessRequired: The access level that the client requires for interacting with the object to which a handle is being opened. The value of this parameter MUST be one of those specified in Access Values (section 2.2.3.1). For rules governing access values, see section 2.2.4.1.

If no specific access level is requested, the server MUST assume a generic read access level.

pClientInfo: A pointer to a SPLCLIENT_CONTAINER (section 2.2.1.2.14) structure. This parameter MUST adhere to the specification in SPLCLIENT_CONTAINER Parameters (section 3.1.4.1.8.8).

The value of the **Level** member of the container that is pointed to by *pClientInfo* MUST be 0x00000001.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- For opening a server object, perform the validation steps specified in Print Server Name Parameters; for opening all other objects, perform the validation steps specified in Printer Name Parameters.
- Perform the validation steps that are specified in Datatype Name Parameters.
- Perform the validation steps that are specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).
- Perform the validation steps that are specified in SPLCLIENT_CONTAINER Parameters (section 3.1.4.1.8.8).
- Verify that the client issuing the call has authorization equivalent to the value of the *AccessRequired* parameter. If verification fails, return ERROR_ACCESS_DENIED.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Locate one of the following objects that correspond to the request:
  - The printer in the "List of Printers", or a job queued for that printer;
  - The server in the "List of Print Server Names"; or
  - The port monitor in the "List of Port Monitors"; or
  - The port in the "List of Ports".

  These lists are specified in section 3.1.1.

- Create an implementation-specific representation of the printer, server, job, port monitor, or port ("the object") that MUST include:
  - An **RPC handle**, which is a snapshot of the printer, server, job, port monitor, or port data that is specific to this instance of the invocation.
  - The data from *pDatatype* and *pDevModeContainer*, if they are not NULL.
  - All other relevant, implementation-specific data that is required to process all other protocol methods that pass in a PRINTER_HANDLE.
  - The data from *pClientInfo*, if it is not NULL.
- Increment the object's reference count to prevent deletion.
- Store the **RPC handle** for the session in the output parameter *pHandle*.
- Return the status of the operation.

3.1.4.2.15 RpcAddPrinterEx (Opnum 70)

RpcAddPrinterEx installs a printer on the print server.<334>

```
DWORD RpcAddPrinterEx(
  [in, string, unique] STRING_HANDLE pName,
  [in] PRINTER_CONTAINER* pPrinterContainer,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] SECURITY_CONTAINER* pSecurityContainer,
  [in] SPLCLIENT_CONTAINER* pClientInfo,
  [out] PRINTER_HANDLE* pHandle
);
```

pName: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pPrinterContainer: A parameter specified in PRINTER_CONTAINER Parameters (section 3.1.4.1.8.6). The **Level** member of the PRINTER_CONTAINER MUST be 0x00000001 or 0x00000002.

pDevModeContainer: A parameter specified in DEVMODE_CONTAINER Parameters (section 3.1.4.1.8.1).

pSecurityContainer: A parameter specified in SECURITY_CONTAINER Parameters (section 3.1.4.1.8.7).

pClientInfo: A parameter specified in SPLCLIENT_CONTAINER Parameters (section 3.1.4.1.8.8).

pHandle: A pointer to a variable that MUST receive the printer remote procedure call (RPC) context handle to the printer object added. RPC context handles are specified in [C706].

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in Print Server Name Parameters.
- Perform the validation steps specified in PRINTER_CONTAINER Parameters.
- Perform the validation steps specified in DEVMODE_CONTAINER Parameters.
- Perform the validation steps specified in SECURITY_CONTAINER Parameters.

- Perform the validation steps specified in SPLCLIENT_CONTAINER Parameters.
- If the value of the **Level** member of the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter is 0x00000002:
  - Verify that the printer driver specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by the *pPrinterContainer* parameter already exists in the system; if that verification fails, return ERROR_UNKNOWN_PRINTER_DRIVER, as specified in [MS-ERREF].
  - Verify that the port specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by the *pPrinterContainer* parameter already exists in the system; if that verification fails, return ERROR_UNKNOWN_PORT, as specified in [MS-ERREF].
  - Verify that the print processor specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by the *pPrinterContainer* parameter already exists in the system; if that verification fails, return ERROR_UNKNOWN_PRINTPROCESSOR, as specified in [MS-ERREF].
  - Verify that the printer with the name specified in the PRINTER_INFO that is pointed to by the **pointer** member of the PRINTER_CONTAINER pointed to by the *pPrinterContainer* parameter does not already exist in the system; if that verification fails, return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF].
- Additional validation MAY<335> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Perform PRINTER_CONTAINER parameter processing steps as specified in PRINTER_CONTAINER Parameters, section 3.1.4.1.8.6.
- If the value of the **Level** member of the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter is 0x00000001, and if the server does not maintain a "List of Known Printers", the server MUST return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF]. Otherwise, the server MUST continue to process the message and compose a response to the client as follows:
  - If the PRINTER_ATTRIBUTE_SHARED bit is set in the **Flags** member of the PRINTER_INFO structure pointed to by the **pPrinterInfo1** member of the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter, add the printer to the "List of Known Printers" as specified in Abstract Data Model (section 3.1.1).<336>
  - If PRINTER_ATTRIBUTE_SHARED is not set in the **Flags** member of the PRINTER_INFO structure pointed to by the **pPrinterInfo1** member of the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter, remove the printer from the "List of Known Printers".
  - Store NULL in the output parameter that is pointed to by *pHandle*.
  - Return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF].

  Note: An error return code is required by remote procedure call (RPC) because NULL was stored to the output parameter pointed to by *pHandle*.
- If the value of the **Level** member of the PRINTER_CONTAINER that is pointed to by the *pPrinterContainer* parameter is 0x00000002:

- Instead of failing the validation steps for missing printer driver, port, and print processor, the server MAY create the required printer driver, port, and print processor if they do not exist in the system.<337>
- Create the printer object and assign to it the security descriptor from the SECURITY_CONTAINER that is pointed to by the *pSecurityContainer* parameter.
- Add the printer to the "List of Printers" specified in section 3.1.1).
- Create a session that includes:
  - An **RPC handle**.
  - A snapshot of the printer data specific to this instance of the printer invocation.
  - The data from DEVMODE that is contained in the DEVMODE_CONTAINER pointed to by the *pDevModeContainer* parameter if it is not NULL.
  - The data from the SPLCLIENT_CONTAINER that is pointed to by the *pClientInfo* parameter if it is not NULL.
- Store the **RPC handle** for the session in the output parameter *pHandle*. The handle returned from this method MUST be granted PRINTER_ALL_ACCESS permission.
- Increment the printer's reference count to prevent deletion.
- If any clients are registered for notifications of the server object change, a notification MUST be broadcast to them.

- Return the status of the operation.

3.1.4.2.16 RpcEnumPrinterData (Opnum 72)

RpcEnumPrinterData enumerates configuration data for a specified printer.<338>

```
DWORD RpcEnumPrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD dwIndex,
  [out, size_is(cbValueName/sizeof(wchar_t))]
    wchar t* pValueName,
  [in] DWORD cbValueName,
  [out] DWORD* pcbValueName,
  [out] DWORD* pType,
  [out, size_is(cbData)] BYTE* pData,
  [in] DWORD cbData,
  [out] DWORD* pcbData
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

dwIndex: The index of the configuration data value to retrieve. The value MUST be greater than or equal to zero and less than the total number of configuration data values for the printer. The client SHOULD use **RpcEnumPrinterKeys** to determine the total number of configuration data values for the printer.

pValueName: A pointer to a buffer that receives a string specifying the name of the configuration data value. For rules governing value names, see section 2.2.4.18.

This parameter can be NULL if *cbValueName* equals zero.

cbValueName: The size, in bytes, of the buffer that is pointed to by the *pValueName* parameter.

pcbValueName: A pointer to a variable that receives the number of bytes stored in the buffer that is pointed to by the *pValueName* parameter.

pType: A parameter specified in Dynamically Typed Query Parameters (section 3.1.4.1.2).

pData: A pointer to BUFFER as specified in Dynamically Typed Query Parameters. This parameter can be NULL if *cbData* equals zero.

cbData: A parameter specified in Dynamically Typed Query Parameters.

pcbData: A parameter specified in Dynamically Typed Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Verify that the value of the *cbValueName* parameter is not smaller than the number of bytes required to hold the string that specifies the name of the value. If that verification fails, the server MUST update the variable that is pointed to by the *pcbValueName* parameter with the number of bytes required and return ERROR_MORE_DATA, as specified in [MS-ERREF].
- Verify that the value of the *dwIndex* parameter is greater than or equal to zero, and smaller than the total number of values for the printer. If *dwIndex* is greater than or equal to the number of values available for the printer, the server MUST return ERROR_NO_MORE_ITEMS, as specified in [MS-ERREF]
- Perform the validation steps specified in Dynamically Typed Query Parameters.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Store the name of the printer property in the string buffer that is pointed to by the *pValueName* parameter and store the length of the name stored in the variable that is pointed to by the *pcbValueName* parameter.
- Using the data identified by *pValueName*, <339> perform the processing and response steps specified in Dynamically Typed Query Parameters.
- Return the status of the operation.

3.1.4.2.17 RpcDeletePrinterData (Opnum 73)

RpcDeletePrinterData deletes specified configuration data for a printer.<340>

```
DWORD RpcDeletePrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pValueName: A pointer to a string that identifies the configuration data to delete. For rules governing value names, see section 2.2.4.18.

The value name is an arbitrary string defined by the printer driver associated with the printer object. The value name "ChangeID"<341> is reserved by the protocol and MAY NOT be used in a call to RpcDeletePrinterData.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- The server MAY verify that the value name is not "ChangeID" and that it complies with the rules specified in section 2.2.4.18.
- Additional validation SHOULD<342> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the printer data specified by *pValueName*.<343>
- Return the status of the operation.

3.1.4.2.18 RpcSetPrinterDataEx (Opnum 77)

RpcSetPrinterDataEx sets the configuration data for a printer or print server.<344> This method is similar to RpcSetPrinterData (section 3.1.4.2.8) but additionally allows the caller to specify the registry key under which to store the data.

```
DWORD RpcSetPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName,
  [in] DWORD Type,
  [in, size_is(cbData)] BYTE* pData,
  [in] DWORD cbData
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pKeyName: A pointer to a string that specifies the key under which the value is to be set. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

If *hPrinter* is a handle to a server object, the key name can be NULL, and the server MUST ignore this parameter.

pValueName: A pointer to a string that identifies the data to set. For rules governing value names, see section 2.2.4.18.

For print servers, a value name MUST be one of the predefined strings listed in Server Handle Key Values (section 2.2.3.10).

For printer objects, a value name is an arbitrary string defined by the printer driver associated with the printer object. The value name "ChangeID"<345> is reserved by the protocol and MAY NOT be used in a call to RpcSetPrinterDataEx.

Type: A code that indicates the type of data that is pointed to by the *pData* parameter. The value SHOULD be one of the possible type codes defined by type values in section 2.2.3.9. For rules governing registry type values, see section 2.2.4.15.

pData: A pointer to an array of bytes that contain the printer configuration data. The type of the data in the buffer is specified by the *Type* parameter.

cbData: The size, in bytes, of the *pData* array. This value SHOULD NOT be zero.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- If the *hPrinter* parameter is a handle to a printer object, the server MUST verify that the *pKeyName* parameter points to a string that complies with the rules for key names specified in section 2.2.4.7.
- For server objects, the server MUST verify that the *pValueName* parameter points to a string that is one of the predefined value names listed in Server Handle Key Values (section 2.2.3.10) with the "read-write" column selected. If this verification fails, return ERROR_INVALID_PARAMETER.
- For printer objects, the server MAY verify that the *pValueName* parameter points to a string that complies with the rules specified in section 2.2.4.18.
- Additional validation SHOULD<346> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the *hPrinter* parameter is a handle to a printer object, store the data that is provided by *pData* with the type that is supplied by *Type* in the printer data value that is identified by *pKeyName* and *pValueName*.
- If *hPrinter* is a handle to a server object, store the data that is provided by *pData* with the type that is supplied by *Type* in the server data value that is identified by the *pValueName* parameter.
- Return the status of the operation.

3.1.4.2.19 RpcGetPrinterDataEx (Opnum 78)

RpcGetPrinterDataEx retrieves configuration data for the specified printer or print server.<347> This method is similar to RpcGetPrinterData (section 3.1.4.2.7), but it also allows the caller to specify the registry key from which to retrieve the data.

```
DWORD RpcGetPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName,
  [out] DWORD* pType,
  [out, size_is(nSize)] BYTE* pData,
  [in] DWORD nSize,
  [out] DWORD* pcbNeeded
```

```
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pKeyName: A pointer to a string that specifies the key under which the value is to be queried. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

If *hPrinter* is a handle to a server object, the key name can be NULL.

pValueName: A pointer to a string that identifies the data to get. For rules governing value names, see section 2.2.4.18.

For print servers, the value name MUST be one of the predefined strings listed in Server Handle Key Values (section 2.2.3.10).

For printer objects, the value name MAY be one of the predefined strings listed in Printer Data Values (section 2.2.3.11). If the value name is not one of the predefined strings, it is an arbitrary string defined by the printer driver associated with the printer object. See RpcGetPrinterData for further details on the interpretation of this value.

pType: A parameter specified in Dynamically Typed Query Parameters (section 3.1.4.1.2).

pData: A pointer to BUFFER, as specified in Dynamically Typed Query Parameters. This parameter can be NULL if *nSize* equals zero

nSize: A parameter specified in Dynamically Typed Query Parameters.

pcbNeeded: A parameter specified in Dynamically Typed Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- If *hPrinter* is a handle to a printer object, the server MUST verify that the *pKeyName* parameter points to a string that complies with the rules for key names specified in section 2.2.4.7.
- For server objects, the server MUST verify that the *pValueName* parameter points to a string that is one of the predefined value names listed in Server Handle Key Values (section 2.2.3.10). If this verification fails, return ERROR_INVALID_PARAMETER.
- For printer objects, the server MUST verify that, if the *pValueName* parameter points to a string that is one of the predefined value names listed in Printer Data Values (section 2.2.3.11), the print server supports retrieving the value for this printer according to the rules in section 2.2.3.11. If this verification fails, return ERROR_NOT_SUPPORTED.
- For printer objects, the server MAY verify that the *pValueName* parameter points to a string that complies with the rules specified in section 2.2.4.18.
- The server MUST perform the validation steps that are specified in section 3.1.4.1.2.
- The server MUST NOT perform access checks on the *hPrinter* object.

- If *hPrinter* is a handle to a printer object with a printer driver version (see *cVersion* in section 2.2.1.3.1) of 0x00000004, the server SHOULD verify that the size in bytes of the data to be returned to the client (the value to be returned via the *pcbNeeded* parameter) does not exceed 0xFFFFFFFF bytes. If this verification fails, the server SHOULD return ERROR_NOT_ENOUGH_MEMORY.<348>
- If *hPrinter* is a handle to a printer object with a printer driver version of 0x00000004, the server SHOULD verify that that the *pValueName* parameter points to a string value that is supported for the printer object. If this verification fails, the server SHOULD return ERROR_NOT_SUPPORTED. <349>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the *hPrinter* parameter is a handle to a printer object, with the data identified by *pKeyName* and *pValueName*, perform the processing and response steps that are specified in section 3.1.4.1.2.
- If *hPrinter* is a handle to a server object, with the data that is identified by *pValueName*, perform the processing and response steps that are specified in section 3.1.4.1.2.
- Return the status of the operation.

3.1.4.2.20 RpcEnumPrinterDataEx (Opnum 79)

RpcEnumPrinterDataEx enumerates all value names and data for a specified printer and key.<350> This method is similar to RpcEnumPrinterData (see section 3.1.4.2.16) but also allows the caller to specify the registry key from which to enumerate the data, and allows retrieving several values in a single call.

```
DWORD RpcEnumPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [out, size_is(cbEnumValues)] BYTE* pEnumValues,
  [in] DWORD cbEnumValues,
  [out] DWORD* pcbEnumValues,
  [out] DWORD* pnEnumValues
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pKeyName: A pointer to a string that specifies the key containing the values to enumerate. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

pEnumValues: A pointer to BUFFER as specified in PRINTER_ENUM_VALUES Structures Query Parameters (section 3.1.4.1.10).

This parameter can be NULL if *cbEnumValues* equals zero.

cbEnumValues: A parameter specified in PRINTER_ENUM_VALUES Structures Query Parameters.

pcbEnumValues: A parameter specified in PRINTER_ENUM_VALUES Structures Query Parameters.

pnEnumValues: A parameter specified in PRINTER_ENUM_VALUES Structures Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Verify that the *pKeyName* parameter points to a string that complies with the rules for key names specified in section 2.2.4.7. If this verification fails, return ERROR_INVALID_PARAMETER.
- Perform the validation steps specified in PRINTER_ENUM_VALUES Structures Query Parameters.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all the values referenced by the specified printer data key.
- Using the enumerated objects, perform the processing and response steps specified in PRINTER_ENUM_VALUES Structures Query Parameters.
- Return the status of the operation.

3.1.4.2.21 RpcEnumPrinterKey (Opnum 80)

RpcEnumPrinterKey enumerates the subkeys of a specified key for a specified printer.<351>

```
DWORD RpcEnumPrinterKey(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [out, size_is(cbSubkey/sizeof(wchar_t))]
    wchar t* pSubkey,
  [in] DWORD cbSubkey,
  [out] DWORD* pcbSubkey
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pKeyName: A pointer to a string that specifies the key containing the subkeys to enumerate. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

pSubkey: A pointer to BUFFER as specified in String Query Parameters (section 3.1.4.1.7).

This parameter can be NULL if *cbSubkey* equals zero.

cbSubkey: A value that is synonymous with the *cbBuf* parameter specified in String Query Parameters.

pcbSubkey: A value that is synonymous with the *pcbNeeded* parameter specified in String Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Perform the validation steps specified in String Query Parameters, substituting ERROR_MORE_DATA (as specified in [MS-ERREF]) for ERROR_INSUFFICIENT_BUFFER.
- Verify that the key specified in the string that is pointed to by the *pKeyName* parameter exists on the server. If it does not exist, the server MUST return ERROR_FILE_NOT_FOUND, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate the key names that have the key specified in the string that is pointed to by the *pKeyName* parameter as an immediate parent. This method returns zero or more key names by storing them as multisz values in the *BUFFER* pointed to by *pSubkey*.
- Using the enumerated objects, perform the processing and response steps that are specified in String Query Parameters.
- Return the status of the operation.

3.1.4.2.22 RpcDeletePrinterDataEx (Opnum 81)

RpcDeletePrinterDataEx deletes a specified value from a printer's configuration data, which consists of a set of named and typed values stored in a hierarchy of registry keys.<352>

```
DWORD RpcDeletePrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pKeyName: A pointer to a string that specifies the key containing the value to delete. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

pValueName: A pointer to a string that identifies the configuration data to delete. For rules governing value names, see section 2.2.4.18.

The value name is an arbitrary string defined by the printer driver associated with the printer object. The value name "ChangeID"<353> is reserved by the protocol and SHOULD NOT be used in a call to RpcDeletePrinterDataEx.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server validates parameters as follows:

- The server MUST perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11),.

- The server MUST verify that the *pKeyName* parameter points to a string that complies with the rules for key names specified in section 2.2.4.7. If this verification fails, return ERROR_INVALID_PARAMETER.
- The server MAY verify that the *pValueName* parameter points to a string that complies with the rules specified in section 2.2.4.18.
- Additional validation SHOULD<354> be performed.

If parameter validation fails, the server MUST fail the operation immediately, returning a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the printer data value indicated by the *pKeyName* and *pValueName* parameters.
- Return the status of the operation.

3.1.4.2.23 RpcDeletePrinterKey (Opnum 82)

RpcDeletePrinterKey deletes a specified key and all of its subkeys for a specified printer.<355>

```
DWORD RpcDeletePrinterKey(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14) methods.

pKeyName: A pointer to a string that specifies the key to delete. A key name is an arbitrary string defined by the printer driver associated with the printer object. For rules governing key names, see section 2.2.4.7.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the *pKeyName* parameter points to a string that complies with the rules for key names specified in section 2.2.4.7. If this verification fails, return ERROR_INVALID_PARAMETER.
- Additional validation MAY<356> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the printer data key indicated by the *pKeyName* parameter and all the subkeys of that key.
- Return the status of the operation.

3.1.4.2.24 RpcAddPerMachineConnection (Opnum 85)

RpcAddPerMachineConnection adds a remote printer name to the list of supported printer connections for every user who locally logs onto the computer running the print server.<357>

This method is used for remote administration of client computers running the print system.

```
DWORD RpcAddPerMachineConnection(
  [in, string, unique] STRING_HANDLE pServer,
  [in, string] const wchar_t* pPrinterName,
  [in, string] const wchar_t* pPrintServer,
  [in, string] const wchar_t* pProvider
);
```

pServer: A value that MUST adhere to the specification in Print Server Name Parameters (section 3.1.4.1.4).

pPrinterName: A value that MUST adhere to the specification in Printer Name Parameters (section 3.1.4.1.5). This parameter MUST specify a printer connection, and it MUST be of the form:

```
SERVER_NAME LOCAL_PRINTER_NAME  [with a non-empty SERVER_NAME.]
```

pPrintServer: A pointer to a string that specifies the name of the print server that is hosting the printer to which the connection is established. For rules governing server names, see section 2.2.4.16.

pProvider: A pointer to a string that specifies the name of a print provider. If the string is the empty string "", an implementation-specific default print provider name MUST be used.<358> For rules governing print provider names, see section 2.2.4.12.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Printer Name Parameters. Perform validations only of the correctness of the syntax of the name; do not perform existence checks for the actual printer object. The existence checks are deferred until the actual creation of the printer object upon user login, and no status feedback is possible. The administration client, therefore, is expected to pass only existing names.
- Perform the validation steps that are specified in Print Server Name Parameters. Perform validations only of the correctness of the syntax of the name; do not perform existence checks for the actual server object. The existence checks are deferred until the actual creation of the printer object upon user login, and no status feedback is possible. The administration client, therefore, is expected to pass only existing names.
- Verify that a per-machine printer connection with the same name does not already exist; and if that verification fails, return ERROR_PRINTER_ALREADY_EXISTS, as specified in [MS-ERREF].
- Additional validation MAY<359> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Add the printer to the **list of per-machine connections** on the print server.
- Defer creation of the actual printer object for each user until the user logs on, and then create a printer object for the printer connection to the printer specified in *pPrinterName*. The created printer object cannot be shared, and can be used only when locally logged onto the computer.

- Return the status of the operation.

3.1.4.2.25 RpcDeletePerMachineConnection (Opnum 86)

RpcDeletePerMachineConnection deletes information about a printer connection.<360>

This method is used for remote administration of client computers running the print system.

```
DWORD RpcDeletePerMachineConnection(
  [in, string, unique] STRING_HANDLE pServer,
  [in, string] const wchar_t* pPrinterName
);
```

pServer: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pPrinterName: A parameter specified in Printer Name Parameters (section 3.1.4.1.5). This parameter MUST specify a printer connection.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in Print Server Name Parameters.
- Perform the validation steps specified in Printer Name Parameters.
- Verify that the per-machine connection exists, and if that fails, return ERROR_INVALID_PRINTER_NAME.
- Additional validation MAY<361> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the per-machine printer connection from the **list of per-machine connections** that is identified by the string that is pointed to by the *pPrinterName* parameter. Defer deletion of the actual printer object for each user until the user logs on, and then delete the printer object for the printer connection to the printer specified in *pPrinterName*.
- Return the status of the operation.

3.1.4.2.26 RpcEnumPerMachineConnections (Opnum 87)

Enumerates each of the connections and copies PRINTER_INFO_4 (section 2.2.1.10.5) structures for all the per-machine connections into the buffer *pPrinterEnum*.<362>

This method is used for remote administration of client computers running the print system.

```
DWORD RpcEnumPerMachineConnections(
  [in, string, unique] STRING_HANDLE pServer,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPrinterEnum,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pServer: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pPrinterEnum: A pointer to the BUFFER, as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _PRINTER_INFO_4

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: A parameter specified in section 3.1.4.1.9.

pcbNeeded: A parameter specified in section 3.1.4.1.9.

pcReturned: A parameter specified in section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in section 3.1.4.1.4.
- Perform the validation steps specified in section 3.1.4.1.9.
- This method SHOULD NOT perform any access checks.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all printers in the **list of per-machine connections** on the server that is identified by the *pServer* parameter (regardless of whether the user has logged on and the printer object has actually been created).
- Using the enumerated objects, perform the processing and response steps specified in section 3.1.4.1.9.
- Return the status of the operation.

The server MUST NOT change the list of printer objects representing pushed printers as part of processing this method.

3.1.4.2.27 RpcSendRecvBidiData (Opnum 97)

RpcSendRecvBidiData sends and receives bidirectional data. This method is used to communicate with port monitors that support such data.<363>

```
DWORD RpcSendRecvBidiData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] const wchar_t* pAction,
  [in] RPC_BIDI_REQUEST_CONTAINER* pReqData,
  [out] RPC_BIDI_RESPONSE_CONTAINER** ppRespData
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pAction: A pointer to a string that specifies an action to take. The following actions SHOULD be supported.<364><365> Port monitors MAY support additional, implementation-specific action strings.<366>

Name/value	Meaning
BIDI_ACTION_ENUM_SCHEMA "EnumSchema"	The method MUST enumerate the supported schemas. The *pReqData* parameter MUST be ignored. The method MUST store one or more values that correspond to supported schema entries in the buffer that is pointed to by the *ppRespData* parameter.
BIDI_ACTION_GET "Get"	The method MUST return the specific value item requested. The *pReqData* parameter MUST specify a single value entry in the schema. The method MUST store the value of that entry in the buffer that is pointed to by the *ppRespData* parameter.
BIDI_ACTION_SET "Set"	The method MUST store the supplied data in a single value item in the schema. The *pReqData* parameter MUST specify a single value entry for the schema and the new value to be stored there. This action MUST NOT change the contents of the buffer that is pointed to by the *ppRespData* parameter.
BIDI_ACTION_GET_ALL "GetAll"	The method MUST return one or more value items that are reachable from the requested schema item. The *pReqData* parameter MUST specify an entry in the schema, which is either a value item or an inner schema entry. The action MUST store one or more value entries, and their associated values, in the buffer that is pointed to by the *ppRespData* parameter.
BIDI_ACTION_GET_WITH_ARGUMENT "GetWithArgument"	The method MUST return one or more value items that are reachable from the requested schema item. The *pReqData* parameter MUST specify an entry in the schema, which is either a value item or an inner schema entry, and a data value to be used when processing the request. The action MUST store one or more value entries, and their associated values, in the buffer that is pointed to by the *ppRespData* parameter.

pReqData: A pointer to an RPC_BIDI_REQUEST_CONTAINER (section 2.2.1.2.10) structure that contains the requested binary data.

ppRespData: A pointer to a variable that receives a pointer to a RPC_BIDI_RESPONSE_CONTAINER (section 2.2.1.2.11) structure that contains the response binary data.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that the port monitor supports this method, and if that verification fails, return ERROR_NOT_SUPPPORTED.
- Verify that the string that is pointed to by the *pAction* parameter specifies a valid command and is supported by the port monitor, and if that verification fails, return any of the following error codes to indicate the request cannot be supported: ERROR_NOT_SUPPORTED, ERROR_INVALID_PARAMETER.

- Verify that the *pReqData* is a non-NULL pointer, and if that verification fails, return ERROR_INVALID_PARAMETER.
- Additional validation MAY be performed<367>.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the *hPrinter* parameter is a handle to a printer object, load the executable object of the monitor supporting the port associated with the printer identified by *hPrinter*.
- If the *hPrinter* parameter is a handle to a port object, load the executable object of the monitor supporting the port identified by *hPrinter*.
- Invoke the method in that library that is identified by the value of the *pAction* parameter and pass *pReqData* to that method.
- Copy the data that is sent from the action method in the buffer that is pointed to by the *ppRespData* parameter; the number of response items MUST match the number of request items.
- Return the status of the operation.

3.1.4.3 Job Management Methods

This section specifies methods for discovering, defining, and scheduling print jobs.

Method	Description
RpcSetJob	RpcSetJob pauses, resumes, cancels, or restarts a print job. It also sets print job parameters, such as the job priority and the document name. Opnum 2
RpcGetJob	RpcGetJob retrieves information about a specified print job. Opnum 3
RpcEnumJobs	RpcEnumJobs retrieves information about a specified set of print jobs for a specified printer. Opnum 4
RpcAddJob	RpcAddJob returns ERROR_INVALID_PARAMETER. Opnum 24
RpcScheduleJob	RpcScheduleJob returns ERROR_SPL_NO_ADDJOB. Opnum 25

3.1.4.3.1 RpcSetJob (Opnum 2)

RpcSetJob pauses, resumes, cancels, or restarts a print job. It also sets print job parameters, such as the job priority and the document name.

```
DWORD RpcSetJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in, unique] JOB_CONTAINER* pJobContainer,
  [in] DWORD Command
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of the print job. This value MUST NOT be zero.

pJobContainer: An optional pointer to a JOB_CONTAINER (section 2.2.1.2.5) that specifies the parameters to set on the job object.

If the value of the *Command* parameter is zero, this pointer MUST be specified.

Command: A Job Control Value (section 2.2.4.6) that specifies an action. This value MUST be one of the following:

Job control action name/value	Meaning
0x00000000	Perform no additional action.
JOB_CONTROL_PAUSE 0x00000001	Pause the print job.
JOB_CONTROL_RESUME 0x00000002	Resume a paused print job.
JOB_CONTROL_CANCEL 0x00000003	Delete a print job.<368>
JOB_CONTROL_RESTART 0x00000004	Restart a print job.
JOB_CONTROL_DELETE 0x00000005	Delete a print job.<369>
JOB_CONTROL_SENT_TO_PRINTER 0x00000006	Used by port monitors to signal that a print job has been sent to the printer. This value SHOULD NOT be used remotely.
JOB_CONTROL_LAST_PAGE_EJECTED 0x00000007	Used by language monitors to signal that the last page of a print job has been ejected from the printer. This value SHOULD NOT be used remotely.
JOB_CONTROL_RETAIN 0x00000008	Keep the print job in the print queue after it prints.
JOB_CONTROL_RELEASE 0x00000009	Release the print job, undoing the effect of a **JOB_CONTROL_RETAIN** action.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps for PRINTER_HANDLE parameters (section 3.1.4.1.11).
- Verify that the value of the *JobId* parameter corresponds to a print job in the list of jobs. If this method is called with a job object handle, *JobId* MUST match the job identifier specified in the call to RpcOpenPrinter or RpcOpenPrinterEx.
- If the *pJobContainer* parameter is specified, verify that it points to a valid JOB_CONTAINER as follows:

- The **Level** member MUST be a value between 0x00000001 and 0x00000004, inclusive;
- The **JobInfo** member MUST point to a JOB_INFO structure (section 2.2.1.7);
- If the **Level** value specified in JOB_CONTAINER is 0x00000003, the **JobId** member of the contained JOB_INFO_3 structure (section 2.2.1.7.3) MUST match the *JobId* input parameter; otherwise, the *JobId* parameter MUST be ignored.
- The members of the JOB_INFO structure MUST comply with the constraints specified for JOB_INFO members (section 2.2.1.3.3) with the exception of **JobId** for all levels except 0x00000003, **PrinterName**, **ServerName**, **PrinterDriverName**, **Size**, **Submitted**, **Time**, **TotalPages**, **pDevMode**, and **pSecurityDescriptor**, which MUST be ignored, and **pMachineName**, which SHOULD be ignored.
- If the **Level** value specified in JOB_CONTAINER is 0x00000001, 0x00000002, or 0x00000004, perform the validation steps that are specified in Datatype Name Parameters (section 3.1.4.1.1) on the *pDataType* member of the JOB_INFO structure.
- If the **Level** value specified in JOB_CONTAINER is 0x00000002 or 0x00000004, and the print processor specified in the *pPrintProcessor* member of the JOB_INFO structure does not already exist in the system, the server SHOULD return ERROR_UNKNOWN_PRINTPROCESSOR to the client.

- Verify that the *Command* parameter is a supported command.
- Additional validation SHOULD<370> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return to the client ERROR_INVALID_PARAMETER or another nonzero error described in the preceding JOB_CONTAINER validation steps; otherwise, the server MUST process the message and respond to the client as follows:

- Modify the job with a **JobId** that matches the *JobId* input parameter to reflect the required changes based on the value of *Command*:
  - (0x00000000): No additional action will be performed.
  - Pausing the print job (0x00000001): Pause the current job specified by *JobId* and allow any succeeding job to print.
  - Resuming the print job (0x00000002): Resume the job specified by *JobId*.
  - Canceling the print job (0x00000003): Cancel the job specified by *JobId*.
  - Restarting the print job (0x00000004): Reinitialize the internal state of the job specified by *JobId* and re-schedule the job for printing.
  - Deleting the print job (0x00000005): Delete the job specified by *JobId* and any internal structures representing that job.
  - Sent the print job to the printer (0x00000006): MAY be set by port monitors associated with a port to signal the job has been sent completely to the device, but it is not sent over the wire.
  - Last Page Ejected (0x00000007): MAY be set by language monitors associated with a port to signal that the physical printer ejected the last page of the job, but it is not sent over the wire.
  - Retain Job (0x00000008): Keep the print job in the print queue after printing is finished. It MAY then be restarted.
  - Release Job (0x00000009): Release a job previously retained. Remove the print job from the queue if it has finished printing and has not been restarted.

- If any clients have registered for notification of a job object change, those clients SHOULD be sent notifications about the changes that the server performs.
- Modify the print job with a **JobId** that matches the *JobId* input parameter by applying the information in the JOB_INFO structure that is contained in the JOB_CONTAINER specified by the *pJobContainer* parameter. The following modifications SHOULD be performed:
  - If the **Level** value specified in JOB_CONTAINER is 0x00000003, the contained JOB_INFO_3 structure specifies the order of print jobs in the job queue. The server SHOULD change the order of jobs so that the job with the identifier specified by the **NextJobId** member of JOB_INFO_3 follows immediately after the job with the identifier specified by the **JobId** member. In addition, the server SHOULD link the two jobs together, so they form an atomic entity and are scheduled such that no other job can intervene between them.
  - If the **Level** value specified in JOB_CONTAINER is 0x00000001, 0x00000002, or 0x00000004, the contained JOB_INFO_1 (section 2.2.1.7.1), JOB_INFO_2 (section 2.2.1.7.2), or JOB_INFO_4 (section 2.2.1.7.4) structure, respectively, specifies the new position of the print job in the job queue. The server SHOULD set the position of the job to the value specified by the **Position** member of the JOB_INFO structure, and the server SHOULD reorder the list of jobs to reflect the new positions.
- Return the status of the operation.

3.1.4.3.2 RpcGetJob (Opnum 3)

RpcGetJob retrieves information about a specified print job.

```
DWORD RpcGetJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of the print job. This value MUST NOT be zero.

Level: The job information level. This value MUST be 0x00000001, 0x00000002, 0x00000003, or 0x00000004.

pJob: A pointer to BUFFER as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _JOB_INFO.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: A parameter specified in section 3.1.4.1.9.

pcbNeeded: A parameter specified in section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that the value of the *JobId* parameter corresponds to a job in the list of jobs. If this verification fails, return ERROR_INVALID_PARAMETER.
- Perform the validation steps that are specified in section 3.1.4.1.9.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Using information about the job, perform the processing and response steps specified in section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.3.3 RpcEnumJobs (Opnum 4)

RpcEnumJobs retrieves information about a specified set of print jobs for a specified printer or port.

```
DWORD RpcEnumJobs(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD FirstJob,
  [in] DWORD NoJobs,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

FirstJob: The zero-based position within the print queue of the first print job to enumerate.

NoJobs: The total number of print jobs to enumerate.

Level: The job information level.

This value MUST be 0x00000001, 0x00000002, 0x00000003, or 0x00000004.

pJob: A pointer to the BUFFER structure specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _JOB_INFO.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: Specified in INFO Structures Query Parameters (section 3.1.4.1.9) .

pcbNeeded: Specified in INFO Structures Query Parameters (section 3.1.4.1.9).

pcReturned: Specified in INFO Structures Query Parameters (section 3.1.4.1.9).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Perform the validation steps specified in INFO Structures Query Parameters (section 3.1.4.1.9).

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If *hPrinter* specifies a printer object, enumerate jobs on the job queue of the printer, up to the number specified by the *NoJobs* parameter, starting with the job whose index is specified by the *FirstJob* parameter.
- If *hPrinter* specifies a port object, enumerate jobs on the job queue of an arbitrary printer associated with that port, up to the number specified by the *NoJobs* parameter, starting with the job whose index is as specified by the *FirstJob* parameter. The method for selecting an arbitrary printer SHOULD match the method used when RpcStartDocPrinter (section 3.1.4.9.1) is called with a port object.
- Using the enumerated objects, perform the processing and response steps specified in section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.3.4 RpcAddJob (Opnum 24)

RpcAddJob does not perform any function but returns a nonzero Windows error code to indicate failure.

```
DWORD RpcAddJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pAddJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object that was opened using RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Level: A value that MUST be 0x00000001, 0x00000002 or 0x00000003.

pAddJob: A pointer to a buffer of undefined values. This value can be NULL if *cbBuf* is zero and *Level* is 0x00000001.

cbBuf: The size, in bytes, of the buffer pointed to by *pAddJob*. If *Level* is 0x00000002 or 0x00000003, this value SHOULD be greater than or equal to 10 bytes.

pcbNeeded: A pointer to a variable that SHOULD receive zero.

Return Values: This method MUST return a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server SHOULD validate parameters as follows:

- Verify that *Level* is valid, and if this verification fails, return ERROR_INVALID_LEVEL.
- If *Level* is 0x0000002 or 0x0000003, verify that *cbBuf* is greater than or equal to 10 bytes on 32-bit and 18 bytes on 64-bit, and if this verification fails, return ERROR_INVALID_DATATYPE.
- If *Level* is 0x0000002 or 0x0000003, verify that at offset 0 from the beginning of the *pAddJob* buffer, there is a 32-bit value on 32-bit implementations and a 64-bit value on 64-bit implementations, between 0 and *cbBuf*, inclusive. If this verification fails, return ERROR_INVALID_LEVEL.

If parameter validation fails, the server SHOULD fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST return ERROR_INVALID_PARAMETER.

This method MUST be implemented to ensure compatibility with protocol clients.

3.1.4.3.5 RpcScheduleJob (Opnum 25)

RpcScheduleJob does not perform any function, but returns a nonzero Windows error code to indicate failure.

```
DWORD RpcScheduleJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId
);
```

hPrinter: A handle to a printer object that was opened using RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, or RpcOpenPrinterEx.

JobId: The identifier of the print job.

Return Values: This method MUST return a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server SHOULD validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).

If parameter validation fails, the server SHOULD fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST return ERROR_SPL_NO_ADDJOB.

This method MUST be implemented to ensure compatibility with protocol clients.

3.1.4.4 Printer Driver Management Methods

This section specifies methods for discovering and installing printer drivers.

Method	Description
RpcAddPrinterDriver	RpcAddPrinterDriver installs a printer driver on the print server and links the configuration, data, and printer driver files. Opnum 9
RpcEnumPrinterDrivers	RpcEnumPrinterDrivers enumerates the printer drivers installed on a specified print server. Opnum 10
RpcGetPrinterDriver	RpcGetPrinterDriver retrieves printer driver data for the specified printer. Opnum 11

Method	Description
RpcGetPrinterDriverDirectory	RpcGetPrinterDriverDirectory retrieves the path of the printer driver directory. Opnum 12
RpcDeletePrinterDriver	RpcDeletePrinterDriver removes the specified printer driver from the list of supported drivers for a server. Opnum 13
RpcGetPrinterDriver2	RpcGetPrinterDriver2 retrieves printer driver data for the specified printer. Opnum 53
RpcDeletePrinterDriverEx	RpcDeletePrinterDriverEx removes the specified printer driver from the list of supported drivers for a server and deletes the files associated with the printer driver. This method also can delete specific versions of the printer driver. Opnum 84
RpcAddPrinterDriverEx	RpcAddPrinterDriverEx installs a printer driver on the print server. This method performs a similar function as RpcAddPrinterDriver (see section 3.1.4.4.1) and is also used to specify options that permit printer driver upgrade, printer driver downgrade, copying of newer files only, and copying of all files (regardless of their time stamps). Opnum 89
RpcGetCorePrinterDrivers	RpcGetCorePrinterDrivers gets the GUIDs, versions, and publish dates of the specified core printer drivers, and the paths to their packages. Opnum 102
RpcGetPrinterDriverPackagePath	RpcGetPrinterDriverPackagePath gets the path to the specified printer driver package. Opnum 104

3.1.4.4.1 RpcAddPrinterDriver (Opnum 9)

RpcAddPrinterDriver installs a printer driver on the print server and links the configuration, data, and printer driver files.

```
DWORD RpcAddPrinterDriver(
  [in, string, unique] STRING_HANDLE pName,
  [in] DRIVER_CONTAINER* pDriverContainer
);
```

pName: Specified in Print Server Name Parameters (section 3.1.4.1.4).

pDriverContainer: Specified in DRIVER_CONTAINER Parameters (section 3.1.4.1.8.3). The **Level** member of the DRIVER_CONTAINER MUST be 0x00000002, 0x00000003, or 0x00000004.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST perform the validation steps specified in:

- Print Server Name Parameters (section 3.1.4.1.4).
- DRIVER_CONTAINER Parameters.

Additional validation MAY<371> be performed.

In addition, print servers SHOULD<372> validate parameters as follows:

- Validate that the **cVersion** member of the **DRIVER_INFO** structure contained in the DRIVER_CONTAINER pointed to by the *pDriverContainer* is strictly less than 0x00000004. If this validation fails, return ERROR_PRINTER_DRIVER_BLOCKED.
- Validate that the **pEnvironment** member of the **DRIVER_INFO** structure contained in the DRIVER_CONTAINER pointed to by the *pDriverContainer* parameter is not "Windows ARM". If this validation fails, return ERROR_NOT_SUPPORTED.

If the installation requested by the print client is a printer driver upgrade, print servers SHOULD perform the following additional validation steps:

- Validate that the currently installed printer driver is not a class printer driver.
- Validate that if the currently installed printer driver has a driver version of 0x00000004, the currently installed printer driver does not have a newer driver date, or if the driver dates are the same, that the currently installed printer driver does not have a newer manufacturer-provided driver version number.
- Validate that if the currently installed printer driver has a driver version of 0x00000004, there are no printers on the print server that are shared and also use the currently installed printer driver.

If this validation fails, the print server MUST return ERROR_PRINTER_DRIVER_BLOCKED. <373>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Copy the printer driver files to their destination. If the copy operation fails, the server MUST fail the call immediately and return a nonzero error response to the client.
- Create the printer driver object, using an implementation-specific mechanism to determine the Boolean values of each of the attributes of the printer driver object. <374>
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.4.2 RpcEnumPrinterDrivers (Opnum 10)

RpcEnumPrinterDrivers enumerates the printer drivers installed on a specified print server.

```
DWORD RpcEnumPrinterDrivers(
  [in, string, unique] STRING_HANDLE pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pDrivers,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pName: Specified in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: Specified in Environment Name Parameters (section 3.1.4.1.3).<375>

Level: The driver information level.

Value	Meaning
0x00000001	Corresponds to _DRIVER_INFO_1 (section 2.2.2.4.1).
0x00000002	Corresponds to _DRIVER_INFO_2 (section 2.2.2.4.2).
0x00000003	Corresponds to _DRIVER_INFO_3 (section 2.2.2.4.3).
0x00000004	Corresponds to _DRIVER_INFO_4 (section 2.2.2.4.4).
0x00000005	Corresponds to _DRIVER_INFO_5 (section 2.2.2.4.5).
0x00000006	Corresponds to _DRIVER_INFO_6 (section 2.2.2.4.6).
0x00000008	Corresponds to _DRIVER_INFO_8 (section 2.2.2.4.8).

pDrivers: A pointer to the BUFFER, as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _DRIVER_INFO.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: Specified in INFO Structures Query Parameters (section 3.1.4.1.9).

pcbNeeded: Specified in INFO Structures Query Parameters (section 3.1.4.1.9).

pcReturned: Specified in INFO Structures Query Parameters (section 3.1.4.1.9).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST perform the validation steps specified in the following sections:

- Print Server Name Parameters (section 3.1.4.1.4).
- Environment Name Parameters (section 3.1.4.1.3).
- INFO Structures Query Parameters (section 3.1.4.1.9).

Additional validation MAY<376> be performed.

In addition, the print server SHOULD validate that, if the *Level* parameter is 0x00000065, every printer driver installed on the print server has a driver version strictly less than 0x00000004. If that validation fails, this method MUST return ERROR_CAN_NOT_COMPLETE.<377>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all drivers on the specified print server that match the requested environment.
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters (section 3.1.4.1.9).
- If at any point during the operation the print server failed to calculate the size of the **INFO** structure for any printer driver, the print server SHOULD fail the operation immediately and return ERROR_CAN_NOT_COMPLETE.<378>
- Return the status of the operation.

3.1.4.4.3 RpcGetPrinterDriver (Opnum 11)

RpcGetPrinterDriver retrieves printer driver data for the specified printer.

```
DWORD RpcGetPrinterDriver(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] wchar t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pDriver,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pEnvironment: A parameter specified in Environment Name Parameters (section 3.1.4.1.3).

Level: The driver information level.

Value	Meaning
0x00000001	Corresponds to _DRIVER_INFO_1 (section 2.2.2.4.1).
0x00000002	Corresponds to _DRIVER_INFO_2 (section 2.2.2.4.2).
0x00000003	Corresponds to _DRIVER_INFO_3 (section 2.2.2.4.3).
0x00000004	Corresponds to _DRIVER_INFO_4 (section 2.2.2.4.4).
0x00000006	Corresponds to _DRIVER_INFO_6 (section 2.2.2.4.6).
0x00000008	Corresponds to _DRIVER_INFO_8 (section 2.2.2.4.8).

pDriver: An optional pointer to *BUFFER*, as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _DRIVER_INFO.

This parameter SHOULD be NULL if *cbBuf* is zero.

cbBuf: Specified in INFO Structures Query Parameters.

pcbNeeded: Specified in INFO Structures Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters by performing the validation steps specified in:

- PRINTER_HANDLE Parameters (section 3.1.4.1.11)). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Environment Name Parameters (section 3.1.4.1.3).
- INFO Structures Query Parameters (section 3.1.4.1.9).

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- The server SHOULD select a printer driver that meets the following specifications:
  - Compatible with the environment specified by the *pEnvironment* parameter.
  - Compatible with one of the printer drivers listed in the **pszzPreviousNames** member of the DRIVER_INFO of the printer driver that is associated with the printer.

  If such a printer driver cannot be located, the server SHOULD return ERROR_UNKNOWN_PRINTER_DRIVER.<379>
- Using the information about the printer driver, perform the processing and response steps specified in INFO Structures Query Parameters.
- Return the status of the operation.

3.1.4.4.4 RpcGetPrinterDriverDirectory (Opnum 12)

RpcGetPrinterDriverDirectory retrieves the path of the printer driver directory.

```
DWORD RpcGetPrinterDriverDirectory(
  [in, string, unique] STRING_HANDLE pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pDriverDirectory,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

pName: Specified in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: Specified in Environment Name Parameters (section 3.1.4.1.3).

Level: A value that MUST be 0x00000001.

pDriverDirectory: An optional pointer to *BUFFER*, as specified in String Query Parameters (section 3.1.4.1.7). If *cbBuf* is zero, this parameter SHOULD be NULL.

cbBuf: Specified in String Query Parameters (section 3.1.4.1.7).

pcbNeeded: Specified in String Query Parameters (section 3.1.4.1.7).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters by performing the validation steps specified in:

- Print Server Name Parameters (section 3.1.4.1.4).
- Environment Name Parameters (section 3.1.4.1.3).
- String Query Parameters (section 3.1.4.1.7).

Additional validation MAY<380> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- With the path of the printer driver directory on the print server, perform the processing and response steps specified in String Query Parameters.
- Return the status of the operation.

3.1.4.4.5 RpcDeletePrinterDriver (Opnum 13)

RpcDeletePrinterDriver removes the specified printer driver from the list of supported drivers for a server.

```
DWORD RpcDeletePrinterDriver(
  [in, string, unique] STRING_HANDLE pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pDriverName
);
```

pName: Specified in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: Specified in Environment Name Parameters (section 3.1.4.1.3).

pDriverName: This parameter MUST be a non-NULL pointer to a string that MUST specify the name of the printer driver to delete. For rules governing printer driver names, see section 2.2.4.3.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters.
- Perform the validation steps that are specified in Environment Name Parameters.
- Verify that the string pointed to by the *pDriverName* parameter contains the name of a driver that is part of the list of drivers that are installed on the server for the environment specified by the string pointed to by the *pEnvironment* parameter; if that verification fails, return ERROR_UNKNOWN_PRINTER_DRIVER.
- Verify that the printer driver is not used by any printer in the system, and if that verification fails, return ERROR_PRINTER_DRIVER_IN_USE.
- Additional validation MAY<381> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear all references to this printer driver from any other data structures.
- Delete the printer driver object.
- If any clients have registered for notifications of the server object change, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.4.6 RpcGetPrinterDriver2 (Opnum 53)

RpcGetPrinterDriver2 retrieves printer driver data for the specified printer.<382>

```
DWORD RpcGetPrinterDriver2(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pDriver,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [in] DWORD dwClientMajorVersion,
  [in] DWORD dwClientMinorVersion,
  [out] DWORD* pdwServerMaxVersion,
  [out] DWORD* pdwServerMinVersion
);
```

hPrinter: A handle to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pEnvironment: A parameter specified in Environment Name Parameters (section 3.1.4.1.3).

Level: The driver information level.

Value	Meaning
0x00000001	Corresponds to _DRIVER_INFO_1 (section 2.2.2.4.1).
0x00000002	Corresponds to _DRIVER_INFO_2 (section 2.2.2.4.2).
0x00000003	Corresponds to _DRIVER_INFO_3 (section 2.2.2.4.3).
0x00000004	Corresponds to _DRIVER_INFO_4 (section 2.2.2.4.4).
0x00000005	Corresponds to _DRIVER_INFO_5 (section 2.2.2.4.5).
0x00000006	Corresponds to _DRIVER_INFO_6 (section 2.2.2.4.6).
0x00000008	Corresponds to _DRIVER_INFO_8 (section 2.2.2.4.8).
0x00000065	Corresponds to _DRIVER_INFO_101 (section 2.2.2.4.9).

pDriver: A pointer to the BUFFER, as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _DRIVER_INFO.

This parameter can be NULL if *cbBuf* is zero.

cbBuf: A parameter specified in INFO Structures Query Parameters.

pcbNeeded: A parameter specified in INFO Structures Query Parameters.

dwClientMajorVersion: The implementation-specific major printer driver version of the client operating system.<383>

dwClientMinorVersion: The implementation-specific minor printer driver version of the client operating system.<384>

pdwServerMaxVersion: A pointer to a DWORD that receives the implementation-specific major version that the operating system supports for that printer driver.

pdwServerMinVersion: A pointer to a DWORD that receives the implementation-specific minimum version that the operating system supports for that printer driver.<385>

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Perform the validation steps specified in Environment Name Parameters.
- Perform the validation steps specified in INFO Structures Query Parameters.

In addition, the print server SHOULD validate that, if the *Level* parameter is 0x00000065, the printer driver associated with the printer object has a driver version strictly less than 0x00000004. If that validation fails, this method MUST return ERROR_CAN_NOT_COMPLETE.<386>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Find a printer driver that is compatible with the OS version on the specified print server that is specified by the value of the *dwClientMajorVersion* and *dwClientMinorVersion* parameters.
- Using the identified printer driver, perform the processing and response steps specified in INFO Structures Query Parameters.

If parameter validation was successful, the server SHOULD also perform the following steps:

- Store the actual compatible operating system version for the printer driver in the variables pointed to by the *pdwServerMaxVersion* and *pdwServerMinVersion* parameters.
- Return the status of the operation.

3.1.4.4.7 RpcDeletePrinterDriverEx (Opnum 84)

RpcDeletePrinterDriverEx removes the specified printer driver from the list of supported drivers for a server and deletes the files associated with it.<387> This method can also be used to delete specific versions of a driver.

```
DWORD RpcDeletePrinterDriverEx(
  [in, string, unique] STRING_HANDLE pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pDriverName,
  [in] DWORD dwDeleteFlag,
  [in] DWORD dwVersionNum
);
```

pName: A STRING_HANDLE structure (section 2.2.1.1.7) that conforms to the parameter specification in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: A string that conforms to the parameter specification in Environment Name Parameters (section 3.1.4.1.3).

pDriverName: A pointer to a string that specifies the name of the printer driver to delete. For rules governing printer driver names, see section 2.2.4.3.

dwDeleteFlag: A bit field that specifies options for deleting files and versions of the printer driver. If the value of this parameter is zero, the driver MUST be removed from the list of supported drivers, and the driver files MUST remain on the print server.

These flags can be combined to specify multiple options.

Value	Meaning
DPD_DELETE_UNUSED_FILES 0x00000001	Remove unused printer driver files. In this case, an error MUST NOT be returned if some of the files are being used by another installed driver.
DPD_DELETE_SPECIFIC_VERSION 0x00000002	Delete the version specified by the value of the *dwVersionNum* parameter. Because more than one version of a printer driver can be installed on a print server, setting this flag does not guarantee that the driver will be removed from the list of supported drivers on the server.
DPD_DELETE_ALL_FILES 0x00000004	Delete the printer driver only if all its associated files can be removed. In this case, an error MUST be returned if some of the files are being used by another installed driver.

dwVersionNum: The version of the printer driver to delete.

The value of this parameter is implementation-specific and identifies the driver version and the operating system for which the driver was written (the driver version contained by each printer driver object in the list of printer drivers described in section 3.1.1). It has the same format and meaning as the **cVersion** members in RPC_DRIVER_INFO structures (section 2.2.1.5).<388>

This parameter MUST be ignored if the **DPD_DELETE_SPECIFIC_VERSION** flag in the *dwDeleteFlag* parameter is not set.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps for Print Server Name Parameters.
- Perform the validation steps for Environment Name Parameters.
- Verify that the string pointed to by the *pDriverName* parameter contains the name of a driver that is part of the list of drivers that are installed on the server for the environment specified by the string pointed to by the *pEnvironment* parameter. If that verification fails, ERROR_UNKNOWN_PRINTER_DRIVER MUST be returned.
- Verify that the printer driver identified by *pDriverName* is not being used by any printer in the system. If that verification fails, ERROR_PRINTER_DRIVER_IN_USE MUST be returned.
- Verify that the value of the *dwDeleteFlag* parameter contains the result of the bitwise OR of zero or more of the **DPD_DELETE** defined constants and that all other bits are zero. If that verification fails, ERROR_INVALID_PARAMETER MUST be returned.
- Additional validation MAY<389> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear references to this version of the printer driver in any other data structures.
- Delete the printer driver object and any associated driver files in compliance with the settings in the *dwDeleteFlag* parameter.

- If the **DPD_DELETE_SPECIFIC_VERSION** bit is set in *dwDeleteFlag*, delete only printer drivers with a version number that matches the value of *dwVersionNum*.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.4.8 RpcAddPrinterDriverEx (Opnum 89)

RpcAddPrinterDriverEx installs a printer driver on the print server.<390> This method performs a function similar to RpcAddPrinterDriver (section 3.1.4.4.1) and is also used to specify options that permit printer driver upgrade, printer driver downgrade, copying of newer files only, and copying of all files regardless of their time stamps.

```
DWORD RpcAddPrinterDriverEx(
  [in, string, unique] STRING_HANDLE pName,
  [in] DRIVER_CONTAINER* pDriverContainer,
  [in] DWORD dwFileCopyFlags
);
```

pName: A string that conforms to the parameter specification in Print Server Name Parameters (section 3.1.4.1.4).

pDriverContainer: A pointer to a DRIVER_CONTAINER structure (section 2.2.1.2.3) that MUST conform to the specification in DRIVER_CONTAINER parameters (section 3.1.4.1.8.3).

The **Level** member of the DRIVER_CONTAINER refers to the level of driver information structure, as follows:

Value	Meaning
0x00000002	DRIVER_INFO_2, specified in section 2.2.1.5.2.
0x00000003	RPC_DRIVER_INFO_3, specified in section 2.2.1.5.3.
0x00000004	RPC_DRIVER_INFO_4, specified in section 2.2.1.5.4.
0x00000006	RPC_DRIVER_INFO_6, specified in section 2.2.1.5.5.
0x00000008	RPC_DRIVER_INFO_8, specified in section 2.2.1.5.6.

dwFileCopyFlags: A bit field that specifies options for copying replacement printer driver files. The value of this parameter is a combination of flags from the following tables.

Exactly one of the following flags MUST be specified:

Value	Meaning
APD_STRICT_UPGRADE 0x00000001	Add the replacement printer driver only if none of the files of the replacement driver are older than any corresponding files of the currently installed driver.
APD_STRICT_DOWNGRADE 0x00000002	Add the replacement printer driver only if none of the files of the currently installed driver are older than any corresponding files of the replacement driver.
APD_COPY_ALL_FILES 0x00000004	Add the printer driver and copy all the files in the driver directory. File time stamps MUST be ignored.

Value	Meaning
APD_COPY_NEW_FILES 0x00000008	Add the printer driver and copy the files in the driver directory that are newer than any of the corresponding files that are currently in use.

Zero or more of the following flags can be specified.

Value	Meaning
APD_COPY_FROM_DIRECTORY 0x00000010	Add the printer driver by using the fully qualified file names that are specified in the _DRIVER_INFO_6 structure. If this flag is specified, one of the other copy flags in this bit field MUST be specified.
APD_DONT_COPY_FILES_TO_CLUSTER 0x00001000	When adding a printer driver to a print server cluster, do not copy the driver files to the shared cluster disk.
APD_COPY_TO_ALL_SPOOLERS 0x00002000	Add the printer driver to cluster spooler servers.
APD_INSTALL_WARNED_DRIVER 0x00008000	Add the printer driver even if it is in the server's warned list.<391>
APD_RETURN_BLOCKING_STATUS_CODE 0x00010000	Specifies the implementation-specific error code to return if the printer driver is blocked from installation by server policy. <392>

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters.
- Perform the validation steps that are specified in DRIVER_CONTAINER Parameters.
- Verify that *dwFileCopyFlags* contains a valid flag value or set of values as described in the *dwFileCopyFlags* parameter definition above. If this verification fails, return ERROR_INVALID_PARAMETER.
- Additional validation MAY <393> be performed.

In addition, print servers SHOULD<394> validate parameters as follows:

- Validate that the **cVersion** member of the **DRIVER_INFO** structure contained in the DRIVER_CONTAINER pointed to by the pDriverContainer is strictly less than 0x00000004. If this validation fails, return ERROR_PRINTER_DRIVER_BLOCKED.
- Validate that the **pEnvironment** member of the **DRIVER_INFO** structure contained in the DRIVER_CONTAINER pointed to by the *pDriverContainer* parameter is not "Windows ARM". If this validation fails, return ERROR_NOT_SUPPORTED.

If the installation requested by the print client is a printer driver upgrade, print servers SHOULD perform the following additional validation steps:

- Validate that the currently installed printer driver is not a class printer driver.
- Validate that if the currently installed printer driver has a driver version of 0x00000004, the currently installed printer driver does not have a newer driver date, or if the driver dates are the

same, that the currently installed printer driver does not have a newer manufacturer-provided driver version number.

- Validate that if the currently installed printer driver has a driver version of 0x00000004, there are no printers on the print server that are shared and also use the currently installed printer driver.

If this validation fails, the print server MUST return ERROR_PRINTER_DRIVER_BLOCKED. <395>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Copy the printer driver files to their destinations, in compliance with the settings described by the value of the *dwFileCopyFlags* parameter.
- Create the printer driver object, using an implementation-specific mechanism to determine the Boolean values of each of the attributes of the printer driver.<396>
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.4.9 RpcGetCorePrinterDrivers (Opnum 102)

RpcGetCorePrinterDrivers gets the GUIDs, versions, and publish dates of the specified core printer drivers, and the paths to their packages.<397>

```
HRESULT RpcGetCorePrinterDrivers(
  [in, string, unique] STRING_HANDLE pszServer,
  [in, string] const wchar_t* pszEnvironment,
  [in] DWORD cchCoreDrivers,
  [in, size_is(cchCoreDrivers)] const wchar_t* pszzCoreDriverDependencies,
  [in] DWORD cCorePrinterDrivers,
  [out, size_is(cCorePrinterDrivers)]
    CORE_PRINTER_DRIVER* pCorePrinterDrivers
);
```

pszServer: A STRING_HANDLE (section 2.2.1.1.7) for a server object. This parameter MUST adhere to the specification in Print Server Name Parameters (section 3.1.4.1.4).

pszEnvironment: A pointer to a **string** that specifies the environment name for which the core printer driver information will be returned. For rules governing environment names and product behaviors, see 2.2.4.4.

cchCoreDrivers: The size, in bytes, of the buffer that is referenced by the *pszzCoreDriverDependencies* parameter.

pszzCoreDriverDependencies: A pointer to a multisz that contains a list of IDs<398> of the core printer drivers to be retrieved.

A print client SHOULD obtain this list of IDs as follows:

1. Call RpcGetPrinterDriver (section 3.1.4.4.3) with a *Level* parameter value of 0x00000008.
2. A _DRIVER_INFO_8 custom-marshaled structure is returned in the *pDriver* parameter.
3. In the _DRIVER_INFO_8 structure, the **szzCoreDependenciesOffset** member contains an offset to a multisz that contains the list of IDs.

cCorePrinterDrivers: The count of CORE_PRINTER_DRIVER (section 2.2.2.13) structures that are contained in the buffer that is pointed to by the *pCorePrinterDrivers* parameter. It MUST equal the number of IDs that are specified in the multisz that is pointed to by the *pszzCoreDriverDependencies* parameter.

pCorePrinterDrivers: A pointer to a buffer that receives an array of CORE_PRINTER_DRIVER structures.

Return Values: This method MUST return zero or an **HRESULT** success value ([MS-ERREF] section 2.1) to indicate successful completion or an ***HRESULT*** error value to indicate failure.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The string pointed to by the *pszEnvironment* parameter MUST specify one of the supported environment names on the server for that type of driver; otherwise the server MUST return ERROR_INVALID_ENVIRONMENT.
- *cCorePrinterDrivers* MUST be equal to or greater than 1; otherwise the server MUST return ERROR_INVALID_PARAMETER.
- *cCorePrinterDrivers* MUST be equal to the number of GUIDs present in *pszzCoreDriverDependencies*; otherwise the server MUST return ERROR_INVALID_PARAMETER.
- The *pCorePrinterDrivers* parameter MUST NOT be NULL.

If parameter validation fails, the server MUST fail the operation immediately, and return a nonzero error response to the client.

Processing and Response Requirements: If parameter validation succeeds, the server MUST process the method call by

- Enumerating the CORE_PRINTER_DRIVER structures in the system.
- Populating each CORE_PRINTER_DRIVER structure in the supplied buffer with information about the core printer driver.
- Returning a response that MUST contain the output parameters mentioned above and the status of the operation.

The server MUST NOT change the **List of Core Printer Drivers** as part of processing this method call.

3.1.4.4.10 RpcGetPrinterDriverPackagePath (Opnum 104)

RpcGetPrinterDriverPackagePath gets the path to the specified printer driver package.<399>

```
HRESULT RpcGetPrinterDriverPackagePath(
  [in, string, unique] STRING_HANDLE pszServer,
  [in, string] const wchar_t* pszEnvironment,
  [in, string, unique] const wchar_t* pszLanguage,
  [in, string] const wchar_t* pszPackageID,
  [in, out, unique, size_is(cchDriverPackageCab)]
    wchar_t* pszDriverPackageCab,
  [in] DWORD cchDriverPackageCab,
  [out] DWORD* pcchRequiredSize
);
```

pszServer: A STRING_HANDLE (section 2.2.1.1.7) for a server object. This parameter MUST adhere to the specification in Print Server Name Parameters (section 3.1.4.1.4).

pszEnvironment: This parameter MUST be a non-NULL pointer to a string that MUST specify the environment name for which the **driver package** path MUST be returned. For rules governing environment names, see section 2.2.4.4.

pszLanguage: This parameter MUST be NULL, or MUST be a non-NULL pointer to a string that MUST specify the language for which the driver package path MUST<400> be returned.

pszPackageID: This parameter MUST be a non-NULL pointer to a string that MUST specify the package name. The package name MUST be obtained by calling RpcGetCorePrinterDrivers.

pszDriverPackageCab: This parameter MUST be a pointer to a buffer that MUST<401> receive a string that specifies the path name of the driver package file. For rules governing path names, see section 2.2.4.9. *pszDriverPackageCab* MUST NOT be NULL unless *cchDriverPackageCab* is zero.

cchDriverPackageCab: This parameter MUST specify the size, in characters, of the buffer that is referenced by the *pszDriverPackageCab* parameter. The value of this parameter MAY<402> be zero.

pcchRequiredSize: This parameter MUST be a non-NULL pointer to a variable that MUST receive the required size of the buffer that is pointed to by the *pszDriverPackageCab* parameter.

Return Values: This method MUST return zero or an **HRESULT** success value ([MS-ERREF] section 2.1) to indicate successful completion or an **HRESULT** error value to indicate failure.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The string pointed to by the *pszEnvironment* parameter MUST specify one of the supported environment names on the server for that type of driver; otherwise the server MUST return ERROR_INVALID_ENVIRONMENT.
- The value of the *pszPackageID* parameter MUST NOT be NULL; otherwise the server MUST return ERROR_INVALID_PARAMETER.
- The value of the *pcchRequiredSize* parameter MUST NOT be NULL; otherwise the server MUST return ERROR_INVALID_PARAMETER.
- The size specified by *cchDriverPackageCab* MUST be sufficient to hold the path name of the driver package file; otherwise the server MUST calculate the required number of characters and write this number to the variable pointed to by the *pcchRequiredSize* output parameter, and return ERROR_INSUFFICIENT_BUFFER.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client.

Processing and Response Requirements: If parameter validation succeeds, the server MUST process the method call by:

- Searching for the driver package cab file for the specified *pszPackageID*.
- Returning the driver package cab path for package ID in the output parameter pszDriverPackageCab.
- Setting the contents of the parameter *pcchRequiredSize* to the size of the string buffer required to hold the driver package cab.

- Returning a response that MUST contain the output parameters mentioned above and the status of the operation.

The server MUST NOT change the list of driver package cabs as part of processing this method call.

3.1.4.5 Form Management Methods

This section specifies methods for discovering and configuring printer forms.

Method	Description
RpcAddForm	RpcAddForm adds a form name to the list of supported forms. Opnum 30
RpcDeleteForm	RpcDeleteForm removes a form name from the list of supported forms. Opnum 31
RpcGetForm	RpcGetForm retrieves information about a specified form. Opnum 32
RpcSetForm	RpcSetForm replaces the form information for the specified form. Opnum 33
RpcEnumForms	The RpcEnumForms method enumerates the forms that the specified printer supports. Opnum 34

3.1.4.5.1 RpcAddForm (Opnum 30)

RpcAddForm adds a form name to the list of supported forms.

```
DWORD RpcAddForm(
  [in] PRINTER_HANDLE hPrinter,
  [in] FORM_CONTAINER* pFormInfoContainer
);
```

hPrinter: This parameter MUST specify a handle to a printer object or server object that MUST have been opened using the RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, or RpcOpenPrinterEx methods.

pFormInfoContainer: This parameter MUST adhere to the parameter specification in FORM_CONTAINER Parameters (section 3.1.4.1.8.4).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Perform the validation steps that are specified in FORM_CONTAINER Parameters (section 3.1.4.1.8.4).
- Verify that the form does not already exist, and if that verification fails, return ERROR_FILE_EXISTS, as specified in [MS-ERREF].

- Additional validation MAY<403> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create the form object.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.5.2 RpcDeleteForm (Opnum 31)

RpcDeleteForm removes a form name from the list of supported forms.

```
DWORD RpcDeleteForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName
);
```

hPrinter: This parameter MUST specify a handle to a printer object or server object that MUST have been opened by using the RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, or RpcOpenPrinterEx methods.

pFormName: This parameter MUST be a non-NULL pointer to a string that MUST identify the form to delete. For rules governing form names, see section 2.2.4.5.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the *pFormName* parameter points to a string that identifies an existing form. If that verification fails, return ERROR_INVALID_FORM_NAME, as specified in [MS-ERREF].
- Additional validation MAY<404> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear the references to this form from any other data structures.
- Delete the form object.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.5.3 RpcGetForm (Opnum 32)

RpcGetForm retrieves information about a specified form.

```
DWORD RpcGetForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pForm,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pFormName: A pointer to a string that specifies the form name for which data is required. For rules governing form names, see section 2.2.4.5.

Level: The form information level.

Value	Meaning
0x00000001	Corresponds to _FORM_INFO_1 (section 2.2.2.5.1).
0x00000002	Corresponds to _FORM_INFO_2 (section 2.2.2.5.2).

pForm: A pointer to the BUFFER, as specified in INFO Structures Query Parameters (section 3.1.4.1.9).

BUFFER TYPE: _FORM_INFO.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: Specified in INFO Structures Query Parameters.

pcbNeeded: Specified in INFO Structures Query Parameters.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that the *pFormName* parameter points to a string that identifies an existing form. If that verification fails, return ERROR_INVALID_FORM_NAME, as specified in [MS-ERREF].
- Perform the validation steps specified in INFO Structures Query Parameters.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Using information about the form, perform the processing and response steps specified in INFO Structures Query Parameters.
- Return the status of the operation.

3.1.4.5.4 RpcSetForm (Opnum 33)

RpcSetForm replaces the form information for the specified form.

```
DWORD RpcSetForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName,
  [in] FORM_CONTAINER* pFormInfoContainer
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pFormName: A pointer to a string that specifies the form name on which the form information is set. For rules governing form names, see section 2.2.4.5.

pFormInfoContainer: A parameter specified in FORM_CONTAINER Parameters, section 3.1.4.1.8.4.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11.
- Verify that the *pFormName* parameter points to a string that identifies an existing form. If that verification fails, return ERROR_INVALID_FORM_NAME, as specified in [MS-ERREF].
- Perform the validation steps that are specified in FORM_CONTAINER Parameters, section 3.1.4.1.8.4.
- Additional validation MAY<405> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Modify the object that corresponds to *pFormName* in order to reflect the new settings supplied in *pFormInfoContainer*.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.5.5 RpcEnumForms (Opnum 34)

The RpcEnumForms method enumerates the forms that the specified printer supports.

```
DWORD RpcEnumForms(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pForm,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hPrinter: This parameter MUST specify a handle to a printer object or server object that MUST have been opened by using the RpcAddPrinter, RpcAddPrinterEx, RpcOpenPrinter, or RpcOpenPrinterEx methods.

Level: This value refers to the level of form information structure, as follows.

Value	Meaning
0x00000001	Corresponds to _FORM_INFO_1, specified in section 2.2.2.5.1.
0x00000002	Corresponds to _FORM_INFO_2, specified in section 2.2.2.5.2.

pForm: This parameter MAY be NULL if *cbBuf* equals zero; otherwise, it MUST be a non-NULL pointer to the BUFFER, as specified in INFO Structures Query Parameters, section 3.1.4.1.9.

BUFFER TYPE: _FORM_INFO.

cbBuf: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcbNeeded: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcReturned: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11.
- Perform the validation steps that are specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- The server MUST NOT perform access checks on the *hPrinter* object.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all forms on the printer or print server.
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.6 Port Management Methods

This section specifies methods for discovering and communicating with printer ports.

Method	Description
RpcEnumPorts	RpcEnumPorts enumerates the ports that are available for printing on a specified server.
RpcDeletePort	Removes a port added by the RpcAddPortEx method (see section 3.1.4.6.3).

Method	Description
RpcAddPortEx	RpcAddPortEx adds a port name to the list of supported ports.
RpcSetPort	RpcSetPort sets the status associated with a printer port.
RpcXcvData	RpcXcvData provides an extensible mechanism by which a client can control ports on the server and exchange port-specific commands and data with the server. For more information about language monitor methods, see section 3.1.4.11.

3.1.4.6.1 RpcEnumPorts (Opnum 35)

RpcEnumPorts enumerates the ports that are available for printing on a specified server.

```
DWORD RpcEnumPorts(
  [in, string, unique] STRING_HANDLE pName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPort,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pName: A parameter specified in Print Server Name Parameters, section 3.1.4.1.4.

Level: The port information level.

Value	Meaning
0x00000001	Corresponds to _PORT_INFO_1, specified in section 2.2.2.8.1.
0x00000002	Corresponds to _PORT_INFO_2, specified in section 2.2.1.9.2.

pPort: A pointer to the BUFFER, as specified in INFO Structures Query Parameters, section 3.1.4.1.9.

BUFFER TYPE: _PORT_INFO.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

pcbNeeded: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

pcReturned: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform the validation steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Additional validation MAY<406> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all ports on the print server.
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.6.2 RpcDeletePort (Opnum 39)

Removes a port.<407>

```
DWORD RpcDeletePort(
  [in, string, unique] STRING_HANDLE pName,
  [in] ULONG_PTR hWnd,
  [in, string] wchar_t* pPortName
);
```

pName: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

hWnd: The value of this parameter SHOULD be set to zero when sent and MUST be ignored on receipt.

pPortName: A pointer to a string that specifies the name of the port that will be deleted. For rules governing port names, see section 2.2.4.10.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps specified in Print Server Name Parameters (section 3.1.4.1.4).
- Verify that the string that is referenced by *pPortName* represents a port installed on the server, and if that validation fails, return ERROR_UNKNOWN_PORT.
- Verify that the port monitor for the port supports the **DeletePort** **monitor module** method, and if that validation fails, return ERROR_NOT_SUPPORTED.
- Verify that the port is not being used by any printer or print job in the system, and if that validation fails, return ERROR_BUSY.
- Additional validation MAY<408> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear the references to this port from any other data structures.
- Delete the port object.
- Modify the list of ports in the system to exclude the deleted port.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.

- Return the status of the operation.

3.1.4.6.3 RpcAddPortEx (Opnum 61)

RpcAddPortEx adds a port name to the list of supported ports.<409>

```
DWORD RpcAddPortEx(
  [in, string, unique] STRING_HANDLE pName,
  [in] PORT_CONTAINER* pPortContainer,
  [in] PORT_VAR_CONTAINER* pPortVarContainer,
  [in, string] wchar_t* pMonitorName
);
```

pName: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pPortContainer: A parameter specified in PORT_CONTAINER Parameters (section 3.1.4.1.8.5). The value of the **Level** member in the PORT_CONTAINER that is referenced by this parameter MUST be 0x00000001 or 0xFFFFFFFF.

pPortVarContainer: A pointer to a PORT_VAR_CONTAINER (section 2.2.1.2.8) information structure that contains information about the port.

pMonitorName: A pointer to a string that specifies the monitor associated with the port. For rules governing monitor names, see section 2.2.4.8.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in Print Server Name Parameters (section 3.1.4.1.4).
- Perform the validation steps specified in PORT_CONTAINER Parameters (section 3.1.4.1.8.5).
- If the value of the **Level** member of the PORT_CONTAINER that is referenced by the *pPortContainer* parameter is 0xFFFFFFFF, verify that the *pPortVarContainer* parameter is not NULL; if that verification fails, return ERROR_INVALID_PARAMETER, as specified in [MS-ERREF].
- Verify that the port does not already exist, and if that verification fails, return ERROR_ALREADY_EXISTS, as specified in [MS-ERREF].
- Verify that the port monitor identified by the *pMonitorName* parameter exists, and if that verification fails, the server MUST return ERROR_INVALID_NAME, as specified in [MS-ERREF].
- Verify that the port monitor for the port supports the **AddPort** monitor module method, and if that validation fails, return ERROR_INVALID_PARAMETER.
- Additional validation MAY<410> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create a new port that is compatible with the port monitor identified by the string that is referenced by the *pMonitorName* parameter; if the **Level** member of the PORT_CONTAINER is 0xFFFFFFFF, pass the data that is contained in the PORT_VAR_CONTAINER that is pointed to by the *pPortVarContainer* parameter to the port monitor.
- Associate the new port with the port monitor identified by the string that is referenced by the *pMonitorName* parameter.

- Modify the list of ports in the system to include the port created by this method.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.6.4 RpcSetPort (Opnum 71)

RpcSetPort sets the status associated with a printer port.<411>

```
DWORD RpcSetPort(
  [in, string, unique] STRING_HANDLE pName,
  [in, string, unique] wchar_t* pPortName,
  [in] PORT_CONTAINER* pPortContainer
);
```

pName: A parameter that adheres to the specification in Print Server Name Parameters (section 3.1.4.1.4).

pPortName: A pointer to a string that specifies the name of the printer port. For rules governing port names, see section 2.2.4.10.

pPortContainer: A parameter that adheres to the specification in PORT_CONTAINER Parameters (section 3.1.4.1.8.5). The level as specified in the **Level** member of the PORT_CONTAINER structure MUST be 0x00000003.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Verify that the string that is referenced by *pPortName* represents a port installed on the server, and if validation fails, the server MUST fail the operation immediately and return ERROR_UNKNOWN_PORT.
- Perform the validation steps that are specified in Print Server Name Parameters.
- Perform the validation steps that are specified in PORT_CONTAINER Parameters.
- Additional validation MAY<412> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- The *pPortContainer* parameter points to a PORT_CONTAINER structure. This structure references a PORT_INFO structure that contains members **dwStatus**, **pszStatus**, and **dwSeverity**. Copy these members to the port object that is referenced by the *pPortName* parameter.
- Return the status of the operation.

3.1.4.6.5 RpcXcvData (Opnum 88)

RpcXcvData provides an extensible mechanism by which a client can control ports on the server and exchange port specific commands and data with the server.<413>

```
DWORD RpcXcvData(
  [in] PRINTER_HANDLE hXcv,
  [in, string] const wchar_t* pszDataName,
```

```
    [in, size_is(cbInputData)] BYTE* pInputData,
    [in] DWORD cbInputData,
    [out, size_is(cbOutputData)] BYTE* pOutputData,
    [in] DWORD cbOutputData,
    [out] DWORD* pcbOutputNeeded,
    [in, out] DWORD* pdwStatus
);
```

hXcv: A handle to a port or port monitor object that was opened by RpcOpenPrinter (section 3.1.4.2.2) or RpcOpenPrinterEx (section 3.1.4.2.14).

pszDataName: A pointer to a string representing the name of the requested data or action. The following table shows the actions that SHOULD be supported. Other port monitor–specific action strings MAY be supported.<414>

Value	Meaning
"AddPort"	Add an instance of a specific port type controlled by the port monitor.
"DeletePort"	Delete an instance of a specific port type controlled by the port monitor.
"MonitorUI"	The action returns the name of the associated port monitor client-side executable configuration module in the buffer that is referenced by the *pOutputData* parameter.

pInputData: A pointer to a buffer that contains input data. This parameter can be NULL if *cbInputData* equals zero.

cbInputData: The size, in bytes, of the buffer pointed to by the *pInputData* parameter.

pOutputData: A pointer to a buffer to receive output data. This parameter can be NULL if *cbOutputData* equals zero.

cbOutputData: The size, in bytes, of the buffer pointed to by the *pOutputData* parameter.

pcbOutputNeeded: A pointer to a location that receives the size, in bytes, required for the buffer pointed to by the *pOutputData* parameter.

pdwStatus: A pointer to a variable that receives the status value that is returned by the port monitor's XcvData method. The value MUST be zero to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate that the print server successfully called the port monitor's XcvData method, or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11, substituting *hXcv* for *hPrinter*. This method assumes that the handle to the port object can be used without further access checks.

The print server SHOULD<415> further validate parameters as follows:

- Verify that the string referenced by the *pszDataName* parameter is a valid command name, and if that verification fails, return ERROR_INVALID_PARAMETER.
- Verify that *pdwStatus* is not NULL, and if that verification fails, return ERROR_INVALID_PARAMETER.

- If the *pszDataName* parameter is set to "AddPort", verify that the *pInputData* parameter is not set to NULL, verify that the value of the *cbInputData* parameter is not zero and that the *pInputData* parameter contains a null terminated character string, and if those verifications fail, return ERROR_INVALID_DATA.
- If the action returns information in *pOutputData* and *cbOutputData* is not zero, verify that the value of the *pOutputData* parameter is not NULL, and if that verification fails, return ERROR_INVALID_PARAMETER.
- For actions that return data in the buffer that is pointed to by the *pOutputData* parameter, verify that the size of the buffer, as specified by the value of the *cbOutputData* parameter, is sufficient. If that verification fails, store the required buffer size in the variable that is pointed to by the *pcbOutputNeeded* parameter and return ERROR_INSUFFICIENT_BUFFER, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Load the executable object of the monitor that supports the port that is associated with the port object that is referenced by the *hXcv* parameter.
- Invoke the **XcvData** method in that library, passing in the *pszDataName*, *pInputData*, *cbInputData*, *pOutputData*, *cbOutputData*, *pcbOutputNeeded*, and *pdwStatus* parameters.
- Return the status from the **XcvData** method.

The port monitor's **XcvData** method processes the message by performing the operation indicated by the string pointed to by the *pszDataName* parameter, and then composes a response as follows:

- If the *pOutputData* parameter is not NULL, copy the output data from the method into *pOutputData*, up to the limit that is specified by the value of the *cbOutputData* parameter.
- Write the size of the data that was copied into *pcbOutputNeeded*.
- If the *pdwStatus* parameter is not NULL, store the status of the port in the variable that is referenced by the *pdwStatus* parameter.
- Return the status of the operation.

3.1.4.7 Port Monitor Management Methods

This section specifies methods for discovering and installing port monitors.

Method	Description
RpcEnumMonitors	The RpcEnumMonitors method retrieves information about the port monitors installed on the specified server. Opnum 36
RpcAddMonitor	RpcAddMonitor installs a local port monitor and links the configuration, data, and monitor files. Opnum 46
RpcDeleteMonitor	RpcDeleteMonitor removes a port monitor. Opnum 47

3.1.4.7.1 RpcEnumMonitors (Opnum 36)

The RpcEnumMonitors method retrieves information about the port monitors installed on the specified server.

```
DWORD RpcEnumMonitors(
  [in, string, unique] STRING_HANDLE pName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pMonitor,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pName: This parameter MUST adhere to the parameter specification in Print Server Name Parameters, section 3.1.4.1.4.

Level: This value refers to the level of port monitor information structure, as follows.

Value	Meaning
0x00000001	Corresponds to _MONITOR_INFO_1, specified in section 2.2.2.7.1.
0x00000002	Corresponds to _MONITOR_INFO_2, specified in section 2.2.2.7.2.

pMonitor: This parameter SHOULD be ignored if *cbBuf* equals zero; otherwise, it is a pointer to the BUFFER, as specified in INFO Structures Query Parameters, section 3.1.4.1.9.

BUFFER TYPE: _MONITOR_INFO.

cbBuf: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcbNeeded: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcReturned: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform the validation steps that are specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Additional validation MAY<416> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all port monitors supporting the methods listed as required or optional in section Monitor Module Methods (section 3.1.4.11) <417> on the print server.

- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.7.2 RpcAddMonitor (Opnum 46)

RpcAddMonitor installs a local port monitor and links the configuration, data, and monitor files.

```
DWORD RpcAddMonitor(
  [in, string, unique] STRING_HANDLE Name,
  [in] MONITOR_CONTAINER* pMonitorContainer
);
```

Name: A parameter that adheres to the specification in Print Server Name Parameters (section 3.1.4.1.4).

pMonitorContainer: A parameter that adheres to the specification in MONITOR_CONTAINER Parameters (section 3.1.4.1.8.9). The **Level** member of the MONITOR_CONTAINER MUST be 0x00000002.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform validation steps as specified in MONITOR_CONTAINER Parameters (section 3.1.4.1.8.9).
- Verify that the port monitor does not already exist in the system, and if that verification fails, return *ERROR_PRINT_MONITOR_ALREADY_INSTALLED*, as specified in [MS-ERREF].
- Additional validation MAY<418> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create the monitor object.
- Return the status of the operation.

3.1.4.7.3 RpcDeleteMonitor (Opnum 47)

RpcDeleteMonitor removes a port monitor.

```
DWORD RpcDeleteMonitor(
  [in, string, unique] STRING HANDLE Name,
  [in, string, unique] wchar t* pEnvironment,
  [in, string] wchar_t* pMonitorName
);
```

Name: This parameter MUST adhere to the parameter specification in Print Server Name Parameters, section 3.1.4.1.4.

pEnvironment: This parameter MUST adhere to the parameter specification in Environment Name Parameters, section 3.1.4.1.3.

pMonitorName: This parameter MUST be a non-NULL pointer to a string that MUST specify the name of the monitor to remove. For rules governing monitor names, see section 2.2.4.8.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform the validation steps that are specified in Environment Name Parameters, section 3.1.4.1.3.
- Verify that the *pMonitorName* parameter points to a string identifying a port monitor installed on the system and, if that verification fails, return ERROR_UNKNOWN_PRINT_MONITOR as specified in MS-ERREF section 2.2.
- Verify that there are no ports (in use by any printer or print job) on the system that are controlled by this monitor at this time and, if that verification fails, return ERROR_PRINT_MONITOR_IN_USE as specified in MS-ERREF section 2.2.
- Additional validation MAY be performed.<419>

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear references to the port monitor from any other data structures.
- Delete the port monitor object.
- Return the status of the operation.

3.1.4.8 Print Processor Management Methods

This section specifies methods for discovering and manipulating print processor objects.

Method	Description
RpcAddPrintProcessor	RpcAddPrintProcessor installs a print processor on the specified server and adds its name to an internal list of supported print processors. Opnum 14
RpcEnumPrintProcessors	RpcEnumPrintProcessors enumerates the print processors installed on a specified server. Opnum 15
RpcGetPrintProcessorDirectory	RpcGetPrintProcessorDirectory retrieves the path for the print processor on the specified server. Opnum 16
RpcDeletePrintProcessor	RpcDeletePrintProcessor removes a print processor. Opnum 48
RpcEnumPrintProcessorDatatypes	RpcEnumPrintProcessorDatatypes enumerates the data types that a specified print processor supports. Opnum 51

3.1.4.8.1 RpcAddPrintProcessor (Opnum 14)

RpcAddPrintProcessor installs a print processor on the specified server and adds its name to an internal list of supported print processors.

```
DWORD RpcAddPrintProcessor(
  [in, string, unique] STRING_HANDLE pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pPathName,
  [in, string] wchar_t* pPrintProcessorName
);
```

pName: This parameter MUST adhere to the parameter specification in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: This parameter MUST adhere to the parameter specification in Environment Name Parameters (section 3.1.4.1.3).

pPathName: This parameter MUST be a non-NULL pointer to a string that MUST specify the file name of the print processor. For rules governing path names, see section 2.2.4.9.

pPrintProcessorName: This parameter MUST be a non-NULL pointer to a string that MUST specify the name of the print processor. For rules governing print processor names, see section 2.2.4.11.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters.
- Perform the validation steps that are specified in Environment Name Parameters.
- Verify that the path identified by the string that is referenced by the *pPathName* parameter contains the necessary file for installing the print processor.
- Verify that the print processor to be added does not have the name "winprint", and if that verification fails, return ERROR_PRINT_PROCESSOR_ALREADY_INSTALLED, as specified in [MS-ERREF].
- Verify that the environment name specified by the *pEnvironment* parameter is not "Windows ARM", and if that verification fails, return ERROR_NOT_SUPPORTED, as specified in [MS-ERREF].<420>
- Additional validation MAY<421> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Copy the print processor file as appropriate to its destination and overwrite an existing print processor with the same name, if necessary.
- Create the print processor object.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.8.2 RpcEnumPrintProcessors (Opnum 15)

RpcEnumPrintProcessors enumerates the print processors installed on a specified server.

```
DWORD RpcEnumPrintProcessors(
  [in, string, unique] STRING_HANDLE pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPrintProcessorInfo,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pName: A parameter specified in Print Server Name Parameters, section 3.1.4.1.4.

pEnvironment: A parameter specified in Environment Name Parameters, section 3.1.4.1.3.

Level: The information level. This value MUST be 0x00000001.

pPrintProcessorInfo: A pointer to BUFFER as specified in INFO Structures Query Parameters, section 3.1.4.1.9

BUFFER TYPE: PRINTPROCESSOR_INFO_1.

This parameter can be NULL if *cbBuf* equals zero.

cbBuf: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

pcbNeeded: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

pcReturned: A parameter specified in INFO Structures Query Parameters, section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform the validation steps specified in Environment Name Parameters, section 3.1.4.1.3.
- Perform the validation steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Additional validation MAY<422> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all print processors on the print server.
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.8.3 RpcGetPrintProcessorDirectory (Opnum 16)

RpcGetPrintProcessorDirectory retrieves the path for the print processor on the specified server.

```
DWORD RpcGetPrintProcessorDirectory(
  [in, string, unique] STRING_HANDLE pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pPrintProcessorDirectory,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

pName: This parameter MUST adhere to the parameter specification in Print Server Name Parameters, section 3.1.4.1.4.

pEnvironment: This parameter MUST adhere to the parameter specification in Environment Name Parameters, section 3.1.4.1.3.

Level: The value of this parameter MUST be 0x00000001.

pPrintProcessorDirectory: This parameter MAY be NULL if *cbBuf* equals zero; otherwise, it MUST be a non-NULL pointer to BUFFER as specified in String Query Parameters, section 3.1.4.1.7.

cbBuf: This parameter MUST adhere to the parameter specification in String Query Parameters, section 3.1.4.1.7.

pcbNeeded: This parameter MUST adhere to the parameter specification in String Query Parameters, section 3.1.4.1.7.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters, section 3.1.4.1.4.
- Perform the validation steps that are specified in Environment Name Parameters, section 3.1.4.1.3.
- Perform the validation steps that are specified in String Query Parameters, section 3.1.4.1.7.
- Additional validation MAY<423> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Using the path of the print processor directory on the print server, perform the processing and response steps specified in String Query Parameters, section 3.1.4.1.7.
- Return the status of the operation.

3.1.4.8.4 RpcDeletePrintProcessor (Opnum 48)

RpcDeletePrintProcessor removes a print processor.

```
DWORD RpcDeletePrintProcessor(
  [in, string, unique] STRING HANDLE Name,
  [in, string, unique] wchar_t* pEnvironment,
  [in, string] wchar_t* pPrintProcessorName
);
```

Name: A parameter specified in Print Server Name Parameters (section 3.1.4.1.4).

pEnvironment: A parameter specified in Environment Name Parameters (section 3.1.4.1.3).

pPrintProcessorName: A pointer to a string that specifies the name of the print processor that will be removed. For rules governing print processor names, see Print Processor Names (section 2.2.4.11).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps specified in Print Server Name Parameters.
- Perform the validation steps specified in Environment Name Parameters.
- The string that is referenced by the *pPrintProcessorName* parameter identifies a print processor installed on the server.
- Verify that there are no printers on the system that use the print processor at this time, and if that verification fails, return ERROR_CAN_NOT_COMPLETE.
- Additional validation MAY<424> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear all references to the specified print processor from any other data structures.
- Delete the print processor object.
- If any clients have registered for notification of server object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.8.5 RpcEnumPrintProcessorDatatypes (Opnum 51)

RpcEnumPrintProcessorDatatypes enumerates the data types that a specified print processor supports.

```
DWORD RpcEnumPrintProcessorDatatypes(
  [in, string, unique] STRING HANDLE pName,
  [in, string, unique] wchar_t* pPrintProcessorName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf), disable_consistency_check]
    BYTE* pDatatypes,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

pName: This parameter MUST adhere to the parameter specification in Print Server Name Parameters, section 3.1.4.1.4.

pPrintProcessorName: This parameter MUST be a non-NULL pointer to a string that MUST specify the name of the print processor whose data types MUST be enumerated. For rules governing print processor names, see section 2.2.4.11.

Level: The value of this parameter MUST be 0x00000001.

pDatatypes: This parameter MAY be NULL if *cbBuf* equals zero; otherwise, it MUST be a non-NULL pointer to BUFFER as specified in INFO Structures Query Parameters, section 3.1.4.1.9.

BUFFER TYPE: _DATATYPES_INFO_1

cbBuf: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcbNeeded: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

pcReturned: This parameter MUST adhere to the parameter specification in INFO Structures Query Parameters, section 3.1.4.1.9.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in Print Server Name Parameters, section 3.1.4.1.4.
- Verify that the print processor that is identified by the string that is referenced by the *pPrintProcessorName* parameter is in the list of print processors.
- Perform the validation steps that are specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Additional validation MAY<425> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Enumerate all data types that are supported by the specified print processor.
- Using the enumerated objects, perform the processing and response steps specified in INFO Structures Query Parameters, section 3.1.4.1.9.
- Return the status of the operation.

3.1.4.9 Document Printing Methods

This section specifies methods for printing documents, pages, and data.

Method	Description
RpcStartDocPrinter	RpcStartDocPrinter notifies the print spooler that a document is being spooled for printing. Opnum 17
RpcStartPagePrinter	RpcStartPagePrinter notifies the spooler that a page is about to be printed on the specified printer. Opnum 18
RpcWritePrinter	RpcWritePrinter sends data to the print spooler. Opnum 19
RpcEndPagePrinter	RpcEndPagePrinter notifies the print spooler that the application is at the end of a page in a print job. Opnum 20

Method	Description
RpcAbortPrinter	RpcAbortPrinter aborts the currently spooling print document. Opnum 21
RpcReadPrinter	RpcReadPrinter retrieves data from the specified printer. Opnum 22
RpcEndDocPrinter	RpcEndDocPrinter notifies the print spooler that the application is at the end of the current print job. Opnum 23
RpcFlushPrinter	RpcFlushPrinter is used by the printer driver to send a buffer of bytes to the port to cleanly abort a print job. It also allows delaying the I/O line to the printer. Opnum 96

3.1.4.9.1 RpcStartDocPrinter (Opnum 17)

RpcStartDocPrinter notifies the print server that a document is being spooled for printing.

```
DWORD RpcStartDocPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] DOC_INFO_CONTAINER* pDocInfoContainer,
  [out] DWORD* pJobId
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14). The printer handle MUST NOT be in use for printing another document at the time of this call.

pDocInfoContainer: A parameter specified in DOC_INFO_CONTAINER Parameters (section 3.1.4.1.8.2).

pJobId: A pointer to a variable that receives a nonzero print job identifier. The job MUST be created with an identifier that is unique for this printer.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Perform the validation steps that are specified in DOC_INFO_CONTAINER Parameters.
- The server MUST verify that RpcStartDocPrinter does not get called twice for a given printer or port object without an intervening call to RpcEndDocPrinter (section 3.1.4.9.7). If that verification fails, return ERROR_INVALID_HANDLE.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create the job object and associate it with a job queue.

- If *hPrinter* is a printer object handle, associate the job with the job queue of that printer object.
- If *hPrinter* is a port object handle, select an arbitrary printer object associated with that port object and associate the job with the job queue of that printer object.

- Associate a data type with the print job.
  - If the **pDatatype** member of the DOC_INFO_1 structure that is pointed to by the **pDocInfo1** pointer in the DOC_INFO_CONTAINER and pointed to by *pDocInfoContainer* is non-NULL, then use the data type specified in **pDatatype** as the data type for the print job.
  - If the **pDatatype** member of the DOC_INFO_1 structure is NULL, *hPrinter* is a printer object handle, and the data type contained in the context for *hPrinter* is non-NULL, then use the data type specified by the context for *hPrinter* for the print job.
  - If the **pDatatype** member of the DOC_INFO_1 structure is NULL and the context for *hPrinter* does not contain a data type, then use the printer's default data type for the print job.
- Write the ID of the created job in the variable that is pointed to by the *pJobId* parameter.
- If any clients that have registered for notification of the job object creation, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.9.2 RpcStartPagePrinter (Opnum 18)

RpcStartPagePrinter notifies the spooler that a page is about to be printed on the specified printer.

```
DWORD RpcStartPagePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that a job has been associated with *hPrinter* using RpcStartDocPrinter, and if that verification fails, return ERROR_SPL_NO_STARTDOC, as specified in [MS-ERREF].
- Verify that printing of the job has not been canceled, and if that verification fails, return ERROR_PRINT_CANCELLED, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Update print job statistics to reflect incremented page count.
- Return the status of the operation.

For each page of a print job, a print client SHOULD perform the following sequence of actions:

- call RpcStartPagePrinter.
- call RpcWritePrinter (section 3.1.4.9.3) zero or more times.
- call RpcEndPagePrinter (section 3.1.4.9.4).

The print server SHOULD treat the calls to RpcStartPagePrinter and RpcEndPagePrinter as informational only, with the only visible result being updating the page count of the print job. The server MUST NOT make any assumptions or perform any validation steps regarding the relative order or frequency of calls to these three methods.

3.1.4.9.3 RpcWritePrinter (Opnum 19)

RpcWritePrinter sends data to the print server.

```
DWORD RpcWritePrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in, size_is(cbBuf)] BYTE* pBuf,
  [in] DWORD cbBuf,
  [out] DWORD* pcWritten
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pBuf: A pointer to a buffer that contains the data to be written. This parameter can be NULL if the value of the *cbBuf* parameter is zero.

cbBuf: The number of bytes of data to be written.

pcWritten: A pointer to a value that receives the number of bytes of data that were written.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11. This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that a job has been associated with *hPrinter* using RpcStartDocPrinter, and if that verification fails, return ERROR_SPL_NO_STARTDOC, as specified in [MS-ERREF].
- Verify that printing of the job has not been canceled, and if that verification fails, return ERROR_PRINT_CANCELLED, as specified in [MS-ERREF].
- If *cbBuf* is not zero, verify that *pBuf* is not NULL.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the *hPrinter* parameter is a printer object handle, copy *cbBuf* bytes of data pointed to by *pBuf* to the job; depending on server policy and settings, the data is added to temporary storage of the job (for example, a spool file), or sent directly to the port.

- If the *hPrinter* parameter is a port object handle, copy *cbBuf* bytes of data pointed to by *pBuf* directly to the port.
- Write the number of bytes that were written to the variable that is pointed to by the *pcWritten* parameter.
- Return the status of the operation.
- If the operation is successful, the server MUST modify the job object to indicate the number of bytes written so far to that job.

3.1.4.9.4 RpcEndPagePrinter (Opnum 20)

RpcEndPagePrinter notifies the print server that the application is at the end of a page in a print job.

```
DWORD RpcEndPagePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that printing of the job has not been canceled, and if that verification fails, return ERROR_PRINT_CANCELLED, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- This method MAY trigger scheduling the job to the device, depending on server settings and policy, such as "Print while spooling".
- Update the count of the pages processed and printed so far in the job object.
- If any clients have registered for notification of job object changes, a notification MUST be broadcast to them.
- Return the status of the operation.

3.1.4.9.5 RpcAbortPrinter (Opnum 21)

RpcAbortPrinter aborts the currently spooling print document.

```
DWORD RpcAbortPrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that a job has been associated with *hPrinter* by using RpcStartDocPrinter, and if that verification fails, return ERROR_SPL_NO_STARTDOC, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- The current job is aborted. If it is in spool stage, spooling MUST stop. If it is in printing stage, printing MUST stop.
- Modify the job object to indicate that the job has been aborted.
- Delete the spool file, if one exists.
- Delete the job object.
- Modify the list of jobs to exclude this deleted job.
- Return the status of the operation.

3.1.4.9.6 RpcReadPrinter (Opnum 22)

RpcReadPrinter retrieves data from the specified job or port.

```
DWORD RpcReadPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [out, size is(cbBuf)] BYTE* pBuf,
  [in] DWORD cbBuf,
  [out] DWORD* pcNoBytesRead
);
```

hPrinter: A handle to a job object or port object that was opened by RpcOpenPrinter (section 3.1.4.2.2) or RpcOpenPrinterEx (section 3.1.4.2.14).

pBuf: A pointer to a buffer that receives the printer data. If the *hPrinter* parameter is the handle to a port object, this method returns the data that is returned by the port monitor.

This parameter can be NULL if the value of the *cbBuf* parameter is zero.

cbBuf: The size, in bytes, of data to be read into the buffer that is pointed to by the *pBuf* parameter.

pcNoBytesRead: A pointer to a variable that receives the number of bytes of data copied into the array to which *pBuf* points.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that printing of the job has not been canceled and if that verification fails, return ERROR_PRINT_CANCELLED, as specified in [MS-ERREF].
- If the value of the *cbBuf* parameter is not zero, verify that the *pBuf* parameter is not NULL.
- Additional validation MAY<426> be performed.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the *hPrinter* parameter is a job object handle, copy data from the temporary storage of the job object to the buffer pointed to by *pBuf*, up to the number of bytes indicated in *cbBuf*, or to the end of the temporary storage's data, whichever comes first.
- If the *hPrinter* parameter is a port object handle, read directly from the port and copy the read data to the buffer pointed to by *pBuf*, up to the number of bytes indicated in *cbBuf* or until no more data can be read, whichever comes first. This requires that the port monitor for the port supports reading data from the port; otherwise, return ERROR_INVALID_HANDLE. <427>
- Write the number of bytes that were copied to the variable that is pointed to by *pcNoBytesRead*.
- If reading from a job object, update the read pointer, so a subsequent read will continue at the correct location.
- Return the status of the operation.

3.1.4.9.7 RpcEndDocPrinter (Opnum 23)

RpcEndDocPrinter notifies the print server that the application is at the end of the current print job.

```
DWORD RpcEndDocPrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: A handle to a printer object or port object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11. This method SHOULD assume that the handle to the printer or port object can be used without further access checks.
- Verify that a print job has been associated with *hPrinter* using RpcStartDocPrinter (section 3.1.4.9.1), and if that verification fails, return ERROR_SPL_NO_STARTDOC, as specified in [MS-ERREF].

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If any clients have registered for notification of job object changes, a notification MUST be broadcast to them.
- Modify the state of the job object to indicate the job has completed spooling if *hPrinter* is a printer object handle; or has completed printing if *hPrinter* is a port object handle.
- If the *hPrinter* parameter is a printer object handle, schedule the job for printing—subject to the configuration of the server and implementation-specific policies—and modify the state of the job object to indicate the job is printing.
- Return the status of the operation.

3.1.4.9.8 RpcFlushPrinter (Opnum 96)

RpcFlushPrinter is used by printer drivers to send a buffer of bytes to a specified port to cleanly abort a print job.<428> It also allows delaying the I/O line to the printer.

```
DWORD RpcFlushPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in, size_is(cbBuf)] BYTE* pBuf,
  [in] DWORD cbBuf,
  [out] DWORD* pcWritten,
  [in] DWORD cSleep
);
```

hPrinter: A handle to a port object that was opened by RpcOpenPrinter (section 3.1.4.2.2) or RpcOpenPrinterEx (section 3.1.4.2.14).

pBuf: A pointer to the array of bytes containing the data to be written to the printer. This parameter can be NULL if the value of the *cbBuf* parameter is zero.

cbBuf: The size, in bytes, of the array pointed to by the *pBuf* parameter.

pcWritten: A pointer to a variable that receives the number of bytes of data that were written to the printer.

cSleep: The time, in milliseconds, to delay the I/O line to the printer port. A value of zero indicates no delay.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the port object can be used without further access checks.
- Verify that a previous RpcWritePrinter (section 3.1.4.9.3) on the same port object has failed due to job cancellation, and if that verification fails, return ERROR_INVALID_HANDLE as specified in [MS-ERREF].
- If the value of the *cbBuf* parameter is not zero, verify that the *pBuf* parameter is not NULL.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Send the contents of the buffer that is pointed to by the *pBuf* parameter to the port.

- If the value of the *cSleep* parameter is not zero, the server MUST halt operations to the port for the number of milliseconds specified by the value of the *cSleep* parameter.
- Write the number of bytes that were written to the port, to the variable that is pointed to by the *pcWritten* parameter.
- Return the status of the operation.

3.1.4.10 Notification Methods

This section specifies methods for obtaining notifications of printing events.

Method	Description
RpcWaitForPrinterChange	RpcWaitForPrinterChange retrieves information about the most recent change notification associated with a printer or print server. Opnum 28
RpcFindClosePrinterChangeNotification	RpcFindClosePrinterChangeNotification closes a change notification object created by calling either the RpcRemoteFindFirstPrinterChangeNotification or RpcRemoteFindFirstPrinterChangeNotificationEx function. The printer or print server associated with the change notification object will no longer be monitored by that object. Opnum 56
RpcRemoteFindFirstPrinterChangeNotification	RpcRemoteFindFirstPrinterChangeNotification creates a remote change notification object that monitors changes to printer objects, and sends change notifications to the client using the method RpcRouterReplyPrinter (see section 3.2.4.1.2). Opnum 62
RpcRemoteFindFirstPrinterChangeNotificationEx	RpcRemoteFindFirstPrinterChangeNotificationEx creates a remote change notification object that monitors changes to printer objects, and sends change notifications to the client using the method RpcRouterReplyPrinterEx (see section 3.2.4.1.4). Opnum 65
RpcRouterRefreshPrinterChangeNotification	RpcRouterRefreshPrinterChangeNotification returns change notification information. Opnum 67

3.1.4.10.1 RpcWaitForPrinterChange (Opnum 28)

RpcWaitForPrinterChange retrieves information about the most recent change notification that is associated with a printer or print server.

```
DWORD RpcWaitForPrinterChange(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Flags,
  [out] DWORD* pFlags
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Flags: The change notifications to wait for. The value of this parameter is a bitwise OR of one or more printer change values, defined in sections 2.2.3.6.1 and 2.2.3.6.2. For rules governing printer change values, see section 2.2.4.13.

pFlags: A pointer to a variable that receives the bitwise OR combination of one or more printer change values.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11. This method SHOULD assume that the handle to the printer or server object can be used without further access checks.

If parameter validation fails, the server MUST fail the operation immediately, returning a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Add the client and the requested change notifications to the list of notification clients for the printer or server object.
- Wait for an implementation-specific period of time,<429> or until one of the changes specified by the value of the *Flags* parameter occurs.
- Remove the client from the list of notification clients for the printer or server object.
- If the *hPrinter* handle is closed with RpcClosePrinter (section 3.1.4.2.9) during this wait period, return ERROR_INVALID_HANDLE.
- If one or more of the specified changes occurred within the time-out period, write a bitwise OR combination of the changes to the variable that is pointed to by *pFlags* and return zero.
- If the time-out period has expired without any of the specified changes, return **PRINTER_CHANGE_TIMEOUT**, as specified in [MS-ERREF].

Note: Because this method waits for an implementation-specific, potentially long, period of time, it can cause the client system to stop responding. Therefore, this method is deprecated and SHOULD NOT be used. The implementer of a protocol client SHOULD consider using RpcRemoteFindFirstPrinterChangeNotificationEx instead.

3.1.4.10.2 RpcFindClosePrinterChangeNotification (Opnum 56)

The RpcFindClosePrinterChangeNotification method closes a change notification object created by RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3) or RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4).<430> The printer or print server associated with the change notification object will no longer be monitored by that object.

```
DWORD RpcFindClosePrinterChangeNotification(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform validation steps as specified in PRINTER_HANDLE Parameters, section 3.1.4.1.11. This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that there is a change notification object associated with the printer object handle.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Clear all internal change notification objects associated with the *hPrinter*.
- Remove the client from the list of notification clients for the server or printer object.
- Return the status of the operation.

3.1.4.10.3 RpcRemoteFindFirstPrinterChangeNotification (Opnum 62)

RpcRemoteFindFirstPrinterChangeNotification creates a remote change notification object that monitors changes to printer objects and sends change notifications to a print client using either RpcRouterReplyPrinter (section 3.2.4.1.2) <431> or RpcRouterReplyPrinterEx (section 3.2.4.1.4).<432>

```
DWORD RpcRemoteFindFirstPrinterChangeNotification(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD fdwFlags,
  [in] DWORD fdwOptions,
  [in, string, unique] wchar t* pszLocalMachine,
  [in] DWORD dwPrinterLocal,
  [in, range(0,512)] DWORD cbBuffer,
  [in, out, unique, size_is(cbBuffer), disable_consistency_check]
    BYTE* pBuffer
);
```

hPrinter: A handle to a printer or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

fdwFlags: Flags that specify the conditions that are required for a change notification object to enter a signaled state. A change notification MUST occur when one or more of the specified conditions are met.

This parameter specifies a bitwise OR of zero or more Printer Change Values (section 2.2.4.13). The rules governing printer change values are specified in section 2.2.4.13.

fdwOptions: The category of printers for which change notifications are returned. This parameter MUST be one of the supported values specified in Printer Notification Values (section 2.2.3.8).

pszLocalMachine: A pointer to a string that represents the name of the client computer. The rules governing server names are specified in section 2.2.4.16.

dwPrinterLocal: An implementation-specific unique value that MUST be sufficient for the client to determine whether a call to RpcReplyOpenPrinter (section 3.2.4.1.1) by the server is associated with the *hPrinter* parameter in this call. <433>

cbBuffer: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

pBuffer: A pointer that MUST be set to NULL when sent and MUST be ignored on receipt.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the print server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that a notification object is not already associated with the current handle.

If parameter validation fails, the server MUST fail the operation immediately returning a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create and initialize a notification object that captures the notification settings requested by the user.
- Create and initialize a notification channel back to the client, over which the server can communicate the change notifications. This MUST be done by calling RpcReplyOpenPrinter on the client specified by the name pointed to by *pszLocalMachine*.
- Associate the notification object with the context for *hPrinter*.
- After the preceding steps have been performed, the server SHOULD add the client to the list of notification clients for the printer object or server object, and it SHOULD notify the client by using RpcRouterReplyPrinter or RpcRouterReplyPrinterEx when the object changes.

  The choice of notification methods does not depend on whether notifications have been requested using RpcRemoteFindFirstPrinterChangeNotification or RpcRemoteFindFirstPrinterChangeNotificationEx. It is based on whether notifications can be expressed in the *fdwFlags* parameter of RpcRouterReplyPrinter alone, or if additional information is required to be provided using the additional parameters of RpcRouterReplyPrinterEx.
- Return the status of the operation.

3.1.4.10.4 RpcRemoteFindFirstPrinterChangeNotificationEx (Opnum 65)

RpcRemoteFindFirstPrinterChangeNotificationEx creates a remote change notification object that monitors changes to printer objects and sends change notifications to a print client using either RpcRouterReplyPrinter (section 3.2.4.1.2) <434> or RpcRouterReplyPrinterEx (section 3.2.4.1.4).<435>

```
DWORD RpcRemoteFindFirstPrinterChangeNotificationEx(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD fdwFlags,
  [in] DWORD fdwOptions,
  [in, string, unique] wchar t* pszLocalMachine,
  [in] DWORD dwPrinterLocal,
  [in, unique] RPC_V2_NOTIFY_OPTIONS* pOptions
);
```

hPrinter: A handle to a printer or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

fdwFlags: Flags that specify the conditions that are required for a change notification object to enter a signaled state. A change notification MUST occur when one or more of the specified conditions are met.

This parameter specifies a bitwise OR of zero or more Printer Change Values (section 2.2.4.13). The rules governing printer change values are specified in section 2.2.4.13.

fdwOptions: The category of printers for which change notifications are returned. This parameter MUST be one of the supported values specified in Printer Notification Values (section 2.2.3.8).

pszLocalMachine: A pointer to a string that represents the name of the client computer. The rules governing server names are specified in section 2.2.4.16.

dwPrinterLocal: An implementation-specific unique value that MUST be sufficient for the client to determine whether a call to RpcReplyOpenPrinter (section 3.2.4.1.1) by the server is associated with the *hPrinter* parameter in this call. <436>

pOptions: A pointer to an RPC_V2_NOTIFY_OPTIONS (section 2.2.1.13.1) structure that specifies printer or job members that the client listens to for notifications. For lists of members that can be monitored, see Printer Notification Values (section 2.2.3.8) and Job Notification Values (section 2.2.3.3).

The value of this parameter can be NULL if the value of *fdwFlags* is nonzero.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that a notification object is not already associated with the current handle.
- Verify that either *pOptions* is not NULL or that the value *fdwFlags* is valid and not zero.

If parameter validation fails, the server MUST fail the operation immediately returning a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create and initialize a notification object that captures the notification settings requested by the user.
- Create and initialize a notification channel back to the client over which the server MUST communicate the change notifications. This MUST be done by calling RpcReplyOpenPrinter on the client specified by the name pointed to by *pszLocalMachine*.
- Associate the notification object with the context for *hPrinter*.
- After the preceding steps have been performed, the server SHOULD add the client to the list of notification clients for the printer object or server object, and it SHOULD notify the client by using RpcRouterReplyPrinter or RpcRouterReplyPrinterEx when the object changes.

  The choice of notification methods does not depend on whether notifications have been requested using RpcRemoteFindFirstPrinterChangeNotification or RpcRemoteFindFirstPrinterChangeNotificationEx. It is based on whether notifications can be

expressed in the *fdwFlags* parameter of RpcRouterReplyPrinter alone, or if additional information is required to be provided using the additional parameters of RpcRouterReplyPrinterEx.

- Return the status of the operation.

3.1.4.10.5 RpcRouterRefreshPrinterChangeNotification (Opnum 67)

RpcRouterRefreshPrinterChangeNotification returns change notification information.<437>

```
DWORD RpcRouterRefreshPrinterChangeNotification(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD dwColor,
  [in, unique] RPC_V2_NOTIFY_OPTIONS* pOptions,
  [out] RPC_V2_NOTIFY_INFO** ppInfo
);
```

hPrinter: A handle to a printer object or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

This handle MUST have been previously used successfully by the client in a call to RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3) or RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4), and it MUST NOT have been closed by calling RpcFindClosePrinterChangeNotification (section 3.1.4.10.2).

dwColor: An implementation-specific value that MAY be used by print clients to get an indication of the order of notifications.<438>

pOptions: A pointer to an RPC_V2_NOTIFY_OPTIONS (section 2.2.1.13.1) structure that specifies printer or job members that the client listens to for notifications. For lists of members that can be monitored, see Printer Notification Values (section 2.2.3.8) and Job Notification Values (section 2.2.3.3).

ppInfo: A pointer to a variable that receives a pointer to an RPC_V2_NOTIFY_INFO (section 2.2.1.13.3) structure that contains notification information.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate parameters as follows:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer or server object can be used without further access checks.
- Verify that the client is in the list of notification clients for the printer object or server object.
- Verify that a notification back channel to the client has been established and is still open.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Collect all the notification data requested for the printer objects.
- Allocate a buffer and write the collected notification data in the buffer.
- Associate the buffer with *ppInfo* output parameter.
- Return the status of the operation.

This method MUST be called when the client receives an RPC_V2_NOTIFY_INFO structure with the **PRINTER_NOTIFY_INFO_DISCARDED** bit set in its **Flags** member. This indicates that an overflow or other error has occurred and that notifications might have been lost, which sets the notification object to the discarded state. The server MUST NOT send any additional notifications until the client makes this method call. If the operation is successful, the server MUST modify the notification object to clear the discarded state.

3.1.4.11 Monitor Module Methods

A monitor module is a server-side executable object that provides a communication path between a print server and the drivers that access hardware on a machine. A port monitor module manages access to I/O port hardware.

Port monitor modules are implementation specific for a given port type.

A port monitor module provides the actual implementation used by the print spooler when one of the Port Management Methods (section 3.1.4.6) is called. The print spooler also uses methods provided by the port monitor module to communicate with the physical print device.

Port monitor modules MUST support the following methods:

- Either: OpenPort or OpenPortEx
- ClosePort
- StartDocPort
- WritePort
- ReadPort
- EndDocPort

Port monitor modules MAY<439> support the following optional methods:

- AddPort
- AddPortEx
- ConfigurePort
- DeletePort

Port monitor modules SHOULD support an additional set of methods, all of which MUST either be implemented together or not be present at all:

- XcvOpenPort
- XcvClosePort

XcvData SHOULD support the following actions (see section 3.1.4.6.5):

- AddPort
- DeletePort
- MonitorUI

Actions MUST be specified by the client in a string pointed to by the *pszDataName* parameter of RpcXcvData.

Additional actions MAY be supported in a given implementation.<440> The following sections, LOCALMON and LPRMON, describe the implementation of the **XcvData** method, its supported actions, and corresponding behaviors in the **LOCALMON** and **LPRMON** monitor modules. All method descriptions assume the standard buffer size validation pattern, as specified for the RpcXcvData method. Unless otherwise specified, for actions not using *pInputData*, *pInputData* MUST be NULL and *cbInputData* MUST be zero. Unless otherwise specified, for actions not using *pOutputData*, *pOutputData* MUST be NULL, *cbOutputData* MUST be zero, and *pcbOutputNeeded* MUST be NULL. For historical reasons, the names of some of the actions supported by **XcvData** are identical to some of the other port monitor module methods. The server method RpcXcvData routes calls to a port monitor's **XcvData** method, and the parameter lists of RpcXcvData and the port monitor's **XcvData** are identical.

3.1.4.11.1 LOCALMON

This section describes the implementation of **XcvData** methods in LOCALMON. This monitor module is used to control parallel and serial ports that could have a printer connected to them.<441>

For parallel and serial port naming, see section 2.2.4.10. For more information about the values listed in the left column in the following table, see section 3.1.4.6.5.

Value	Meaning
"MonitorUI"	This action returns the name of the client-side interface monitor. The *pOutputData* parameter MUST be a non-NULL pointer to the buffer that receives the string representing the name of the interface monitor.
"AddPort"	This action adds a local port. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port.
"DeletePort"	This action deletes a local port. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port.
"ConfigureLPTPortCommandOK"	This action determines configuration of the LPT port and sets the transmission retry timeout. The *pInputData* parameter MUST be a non-NULL pointer to a string representing a number from 1 through 3, inclusive. Those numbers map to the port to configure; that is, whether it is "LPT1" OR "LPT2" OR "LPT3". The *pOutputData* parameter is not used.
"PortExists"	This action checks whether a local port exists in the print server's list of port objects. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port. The *pOutputData* parameter MUST be a non-NULL pointer to a DWORD variable that MUST receive the value 0 if the port does not exist and a nonzero value if the port exists.
"PortIsValid"	This action determines whether a given name is a valid port name accepted by the port monitor. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port. If the port identifies a valid port name, the method MUST return ERROR_SUCCESS; otherwise, it MUST return a nonzero error code.

3.1.4.11.2 LPRMON

This section describes the implementation of **XcvData** methods in LPRMON. This monitor module is used to control printers over a network on machines that have implemented Unix print server functions and expose them through the **Line Printer (LPR) Protocol**, as defined in [RFC1179].<442>

For network port naming, see section 2.2.4.10. For more information about the values listed in the Value column in the following table, see section 3.1.4.6.5.

Value	Meaning
"MonitorUI"	This action returns the name of the client-side interface monitor. The *pOutputData* parameter MUST be a non-NULL pointer to a buffer that receives the string representing the name of the interface monitor.
"AddPort"	This action adds an LPR printer port. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port. The *pOutputData* parameter is not used.
"DeletePort"	This action deletes an LPR printer port. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port. The *pOutputData* parameter is not used.
"CheckPrinter"	This action checks on the LPR printer port. The *pInputData* parameter MUST be a non-NULL pointer to a string representing the name of the port connected to the printer.

3.1.4.11.3 TCPMON

Information about the implementation of the TCPMON monitor module can be found in the command value following table and in the subsections that follow.

The TCPMON monitor module is used to control printers directly connected to a **TCP/IP** network.<443>

For network port naming, see section 2.2.4.10.

For structures used with the TCPMON monitor module, see section 2.2.2.14

The following table discusses command values used by the TCPMON monitor module.

Value	Meaning
"AddPort"	This action adds a printer port. The *pInputData* parameter is a non-NULL pointer to a PORT_DATA_1 or PORT_DATA_2 structure.
"DeletePort"	This action deletes a printer port. The *pInputData* parameter is a non-NULL pointer to a DELETE_PORT_DATA_1 structure.
"MonitorUI"	This action returns the name of the client-side monitor interface module. The *pOutputData* parameter is a non-NULL pointer to a buffer that receives the string representing the name of the interface module.
"ConfigPort"	This action configures a printer port. The *pInputData* parameter is a non-NULL pointer to a PORT_DATA_1 or PORT_DATA_2 structure.
"GetConfigInfo"	This action gets configuration information for a printer port. The *pInputData* parameter is a non-NULL pointer to a CONFIG_INFO_DATA_1 structure. The *pOutputData* parameter points to a buffer that receives a PORT_DATA_1 or PORT_DATA_2 structure describing the port.
"HostAddress"	This action gets the printer's host name. *pOutputData* is a non-NULL pointer to a buffer that receives a string containing the printer's host name.
"IPAddress"	This action gets the printer's IP address. *pOutputData* is a non-NULL pointer to a buffer that receives a string containing the printer's IP address.
"SNMPCommunity"	This action gets the printer's Simple Network Management Protocol (SNMP) [RFC1157] community name. *pOutputData* is a non-NULL pointer to a buffer that receives a string containing the printer's SNMP community name.
"SNMPDeviceIndex"	This action gets the printer's SNMP device index. *pOutputData* is a non-NULL pointer to a

Value	Meaning
	variable that receives a DWORD value containing the printer's SNMP device index.
"SNMPEnabled"	This action determines whether SNMP is enabled for the printer. *pOutputData* is a non-NULL pointer to a variable that receives the DWORD value 0x00000000 if SNMP is disabled and a nonzero value otherwise.
"GetIdlePollingState"	This action determines whether TCPMON is set to poll automatically for the printer ("idle polling"). *pOutputData* is a non-NULL pointer to a variable that receives the DWORD value 0x00000000 if idle polling is disabled, and a nonzero value otherwise.
"SetIdlePollingState"	This action sets the idle polling state for the printer. *pInputData* is a non-NULL pointer to a DWORD which is set to 0x00000001 to enable idle polling, and to 0x00000000 to disable it.
"SetDeviceIDOid"	This action sets the object identifier (OID) used to query the IEEE 1284 device ID from the printer.<444> (For details, see [IEEE1284].) *pInputData* is a non-NULL pointer to a string specifying the OID. If this OID is not set, TCPMON uses a default of "1.3.6.1.4.1.2699.1.2.1.2.1.1.3.<1-based port index>".
"DeviceID"	This action initiates a query for the IEEE 1284 device ID.<445> (For details, see [IEEE1284].) *pOutputData* is a non-NULL pointer to a buffer that receives a string containing the IEEE 1284 device ID, as specified in [IEEE1284].
"GetPortList"	This action requests the list of supported ports on a device.<446>*pInputData* is a non-NULL pointer to a string containing the IP address of hostname of the device. *pOutputData* is a non-NULL pointer to a buffer that receives a PORT_DATA_LIST_1 structure.
"CleanupPort"	This action attempts to remove the TCP/IP port associated with the **hXcv** handle.<447> If printers are still using the port, it will not be removed.

3.1.4.11.4 WSDMON

This section describes the implementation of the XcvData method in **WSDMON**. This monitor module is used to control Web Services for Devices (WSD) printers.<448> WSDMON does not have a corresponding user interface module.

For network port naming, see section 2.2.4.10.

For structures used with the WSDMON monitor module, see section 2.2.2.15.

The following table discusses command values used by the WSDMON monitor module.

Value	Meaning
"CleanupPort"	Attempts to remove the Web Services for Devices (WSD) port associated with the **hXcv** handle. If printers are still using the port, it will not be removed.
"DeviceID"	Initiates a query for the WSD **DeviceID**. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a string containing the WSD **DeviceID** (PKEY_PNPX_GlobalIdentity).
"PnPXID"	Initiates a query for the WSD **PnPXID**. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a string containing the WSD **PnPXID** (PKEY_PNPX_ID).
"ResetCommunication"	Attempts to make sure that the communication between the printer and the operating

Value	Meaning
	system is working properly.
"ServiceID"	Initiates a query for the WSD **ServiceID**. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a string containing the WSD **ServiceID** (PKEY_PNPX_ServiceID).
"CheckCluster"	Determines whether the queried server is a stand-alone server or a cluster node.<449> The required value for the *pOutputData* parameter is a non-NULL pointer to a variable that is required to receive the DWORD value 0x00000000 if the queried server is not a cluster node and a nonzero value otherwise.
"DiscoverDevice"	Uses WS-discovery directed **unicast** search to try to find a WSD–enabled device at the supplied **URI** of the endpoint.<450> The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the URI of the WSD endpoint. If a WSD device is found, and it supports the WSD print service definition, ERROR_SUCCESS is returned; otherwise, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF].
"DriverAvailable"	Determines whether a printer driver for the queried device is available in the server's driver store.<451> The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the URI of the WSD endpoint. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a WSD_DRIVER_DATA structure if the specified endpoint supports a WSD Printer Service and a driver is available. If the endpoint does not support the WSD Printer Service, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF]. If no driver can be found, ERROR_CANNOT_DETECT_DRIVER_FAILURE is returned.
"AssocDevice"	Searches for a WSD Printer Service at the supplied URI of the endpoint and installs the printer if found.<452> This command is only supported on stand-alone servers. The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the URI of the WSD endpoint. If a WSD Printer Service is found, a **PnPX** installation of the printer is initiated; otherwise, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF].
"AddPrinterPort"	Searches for a WSD Printer Service at the supplied URI of the endpoint, and if one is found, creates a new WSD port connected to the discovered device.<453> This command is only supported on cluster servers. The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the URI of the WSD endpoint. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a string identifying the new port name if a WSD Printer Service is found; otherwise, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF].
"BackupPort"	Initiates a query for the WSD port backup data.<454> The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a WSD_BACKUP_PORT_DATA structure.
"AssocDeviceMulticast"	Searches for a WSD Printer Service at the device endpoint that is specified by the **GlobalID**, which in turn is specified by the string pointed to by *pInputData*, and installs the printer if found.<455> This command is only supported on stand-alone servers. The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the **GlobalID** of the WSD endpoint. If a WSD Printer Service is found, a **PnPX** installation of the printer is initiated; otherwise, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF].
"RestorePort"	Searches for a WSD printer service that is identified by the **ServiceID** at the device endpoint specified by the **GlobalID** and optionally **RemoteURL**, depending on the value of the **DiscoveryMethod**.<456>If the value of **DiscoveryMethod** is *kMulticast*, only the **ServiceID** and **GlobalID** values MUST be used for the search. If the value of **DiscoveryMethod** is *kDirected*, the **ServiceID**, **GlobalID**, and **RemoteURL** values MUST be used for the search. The required value for the *pInputData* parameter is a WSD_BACKUP_PORT_DATA_EX structure (section 2.2.2.15.3) that contains the values relevant to the printer port to restore. If a matching WSD printer service is found, a

Value	Meaning
	PnPX installation of the printer is initiated; otherwise a new port with the specified information is created with a status of PORT_STATUS_OFFLINE (section 2.2.1.9.3). The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that receives a string identifying the new port by name.
"AddMulticastPort"	Searches for a WSD Printer Service at the device endpoint that is specified by the **GlobalID**, which in turn is specified by the string pointed to by *pInputData*, and if one is found, creates a new WSD port connected to the discovered device.<457> <458>

3.1.4.12 Job Named Property Management Methods

This section specifies methods for creating, updating, deleting, and enumerating **Job Named Properties** (section 3.1.1) for a specified print job.

Method	Description
RpcGetJobNamedPropertyValue (section 3.1.4.12.1)	RpcGetJobNamedPropertyValue retrieves the value of the specified **Job Named Property** for the print job. Opnum 110
RpcSetJobNamedProperty (section 3.1.4.12.2)	RpcSetJobNamedProperty creates a new **Job Named Property** or changes the value of an existing **Job Named Property** for the print job. Opnum 111
RpcDeleteJobNamedProperty (section 3.1.4.12.3)	RpcDeleteJobNamedProperty deletes a **Job Named Property** for the print job. Opnum 112
RpcEnumJobNamedProperties (section 3.1.4.12.4)	RpcEnumJobNamedProperties enumerates the **Job Named Properties** for the print job. Opnum 113

3.1.4.12.1 RpcGetJobNamedPropertyValue (Opnum 110)

RpcGetJobNamedPropertyValue retrieves the current value of the specified **Job Named Property** (section 3.1.1).<459>

```
DWORD RpcGetJobNamedPropertyValue(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in, string] const wchar_t* pszName,
  [out] RPC PrintPropertyValue* pValue
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of a print job. This value MUST NOT be zero.

pszName: A pointer to a string that specifies the **Job Named Property** to be queried. This pointer MUST NOT be NULL.

pValue: A pointer to an RPC_PrintPropertyValue (section 2.2.1.14.1) structure that on return from this call contains the value of the **Job Named Property** specified by the *pszName* argument.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

On receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the value of the *JobId* parameter corresponds to a job in the list of jobs. If the object specified by the *hPrinter* parameter is a server object, search for a print job in each printer in the list of printers on the print server. If the object specified by the *hPrinter* parameter is a printer object, search for a print job only in the list of print jobs for the specified printer. If the object specified by the *hPrinter* parameter is a job object, compare the identifier of this print job with the specified *JobId*. If this verification fails, return ERROR_INVALID_PARAMETER.
- Verify that the value of the *pValue* parameter is a non-NULL pointer to a string. If this verification fails, return ERROR_INVALID_PARAMETER.
- Verify that the value of the *pszName* parameter corresponds to an existent **Job Named Property** for the print job specified with the *JobId* parameter. If this verification fails, return ERROR_NOT_FOUND.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the print client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Allocate and initialize a buffer with the **PrintPropertyValue** structure to be returned to the client via the *pValue* parameter.
- Return the status of the operation.

On successful completion of this call, the client SHOULD free the buffer specified by the *pValue* parameter.

3.1.4.12.2 RpcSetJobNamedProperty (Opnum 111)

RpcSetJobNamedProperty creates a new **Job Named Property** (section 3.1.1), or changes the value of an existing **Job Named Property** for the specified print job.<460>

```
DWORD RpcSetJobNamedProperty(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in] RPC PrintNamedProperty* pProperty
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of a print job. This value MUST NOT be zero.

pProperty: A pointer to an RPC_PrintNamedProperty (section 2.2.1.14.2) structure specifies the property to be created if it not exists for the print job specified by the *JobId* parameter, or update an existing property with a new value. This pointer MUST NOT be NULL.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the value of the *JobId* parameter corresponds to a job in the list of jobs. If the object specified by the *hPrinter* parameter is a server object, search for a print job in each printer in the list of printers on the print server. If the object specified by the *hPrinter* parameter is a printer object, search for a print job only in the list of print jobs for the specified printer. If the object specified by the *hPrinter* parameter is a job object, compare the identifier of this print job with the specified *JobId*. If this verification fails, return ERROR_INVALID_PARAMETER.
- Verify that the RPC_PrintNamedProperty structure specified by the *pProperty* parameter contains an **ePropertyType** member set to a valid value as specified in section 2.2.1.14.3. If this verification fails, return ERROR_INVALID_FLAGS.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- If the property specified by the *pProperty* parameter does not exist for the print job specified by the *JobId* parameter, create a new property with the name and value specified by the *pProperty* parameter.
- If the property specified by the *pProperty* parameter does exist for the print job specified by the *JobId* parameter, update the property with the value specified by the *pProperty* parameter.
- Return the status of the operation.

3.1.4.12.3 RpcDeleteJobNamedProperty (Opnum 112)

RpcDeleteJobNamedProperty deletes an existing **Job Named Property** (section 3.1.1) for the specified print job.<461>

```
DWORD RpcDeleteJobNamedProperty(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in, string] const wchar_t* pszName
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of a print job. This value MUST NOT be zero.

pszName: A pointer to a string that specifies the **Job Named Property** to be deleted. This pointer MUST NOT be NULL.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the value of the *JobId* parameter corresponds to a job in the list of jobs. If the object specified by the *hPrinter* parameter is a server object, search for a print job in each printer in the list of printers on the print server. If the object specified by the *hPrinter* parameter is a printer

object, search for a print job only in the list of print jobs for the specified printer. If the object specified by the *hPrinter* parameter is a job object, compare the identifier of this print job with the specified *JobId*. If this verification fails, return ERROR_INVALID_PARAMETER.

- Verify that the value of the *pszName* parameter is a non-NULL pointer to a string. If this verification fails, return ERROR_INVALID_PARAMETER.
- Verify that the value of the *pszName* parameter corresponds to an existent **Job Named Property** for the print job specified with the *JobId* parameter. If this verification fails, return ERROR_NOT_FOUND.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Delete the property specified by the *pszName* parameter for the print job specified by the *JobId* parameter.
- Return the status of the operation.

3.1.4.12.4 RpcEnumJobNamedProperties (Opnum 113)

RpcEnumJobNamedProperties enumerates the **Job Named Properties** (section 3.1.1) for the specified print job.<462>

```
DWORD RpcEnumJobNamedProperties(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [out] DWORD* pcProperties,
  [out, size_is(,*pcProperties)] RPC_PrintNamedProperty** ppProperties
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object, job object, or server object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

JobId: The identifier of a print job. This value MUST NOT be zero.

pcProperties: On successful return from this call, this parameter is a pointer to the address of an array of RPC_PrintNamedProperty (section 2.2.1.14.2) structures returned. This pointer MUST NOT be NULL.

ppProperties: On successful return from this call, this parameter is a pointer to the address of an array of RPC_PrintNamedProperty structures returned. This pointer MUST NOT be NULL.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11).
- Verify that the value of the *JobId* parameter corresponds to a job in the list of jobs. If the object specified by the *hPrinter* parameter is a server object, search for a print job in each printer in the list of printers on the print server. If the object specified by the *hPrinter* parameter is a printer object, search for a print job only in the list of print jobs for the specified printer. If the object specified by the *hPrinter* parameter is a job object, compare the identifier of this print job with the specified *JobId*. If this verification fails, return ERROR_INVALID_PARAMETER.

- Verify that the *pcProperties* and *ppProperties* pointers are not NULL. If this verification fails, return ERROR_INVALID_PARAMETER.
- Verify that the buffer to contain the array of **RPC_PrintNamedProperty** structures can be successfully allocated. If this verification fails, return ERROR_NOT_ENOUGH_MEMORY.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Allocate and initialize a buffer containing the array of **RPC_PrintNamedProperty** structures to be returned to the client via the *ppProperties* parameter.
- Return the number of enumerated properties (the number of returned **RPC_PrintNamedProperty** structures) in the *pcProperties* parameter.
- Return the status of the operation.

Upon a successful completion of this call the client SHOULD free the buffer specified by the *ppProperties* parameter.

3.1.4.13 Branch Office Print Remote Logging Methods

This section specifies methods for processing **Branch Office Print Remote Log Entries** (section 3.1.1) for a specified printer.

Method	Description
RpcLogJobInfoForBranchOffice (section 3.1.4.13.1)	RpcLogJobInfoForBranchOffice processes one or more **Branch Office Print Remote Log Entries** by writing them to the Microsoft-Windows-PrintService/Admin and Microsoft-Windows-PrintService/Operations event channels. Opnum: 116

3.1.4.13.1 RpcLogJobInfoForBranchOffice (Opnum 116)

RpcLogJobInfoForBranchOffice processes one or more **Branch Office Print Remote Log Entries** (section 3.1.1).<463>

```
DWORD RpcLogJobInfoForBranchOffice(
  [in] PRINTER_HANDLE hPrinter,
  [in, ref] RPC_BranchOfficeJobDataContainer* pBranchOfficeJobDataContainer
);
```

hPrinter: A PRINTER_HANDLE (section 2.2.1.1.4) to a printer object that was opened by RpcAddPrinter (section 3.1.4.2.3), RpcAddPrinterEx (section 3.1.4.2.15), RpcOpenPrinter (section 3.1.4.2.2), or RpcOpenPrinterEx (section 3.1.4.2.14).

pBranchOfficeJobDataContainer: A pointer to an RPC_BranchOfficeJobDataContainer (section 2.2.1.2.17) structure that contains one or more RPC_BranchOfficeJobData (section 2.2.1.15.2) structures, each of which holds a single **Branch Office Print Remote Log Entry**.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

On receiving this message, the server MUST validate the following:

- Perform the validation steps that are specified in PRINTER_HANDLE Parameters (section 3.1.4.1.11). This method SHOULD assume that the handle to the printer object can be used without further access checks.
- Verify that the value of the *pBranchOfficeJobDataContainer* parameter is a non-NULL pointer, and if that verification fails, return ERROR_INVALID_PARAMETER.

If parameter validation fails, the server MUST fail the operation immediately and return a nonzero error response to the client. Otherwise, the server MUST process the message and compose a response to the client as follows:

- Create event channel entries that correspond to the data in the array of RPC_BranchOfficeJobData structures in the RPC_BranchOfficeJobDataContainer structure specified by the *pBranchOfficeJobDataContainer* parameter. The number of structures is specified by the **cJobDataEntries** member of the container structure.
- Return the status of the operation.

3.1.5 Timer Events

No protocol timer events are required on the server beyond the timers required in the underlying remote procedure call (RPC) protocol.

3.1.6 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying remote procedure call (RPC) protocol.

3.2 Client Details

3.2.1 Abstract Data Model

This section describes a conceptual model of a possible data organization that a print client implementation might need to maintain in order to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this specification.

The following abstract data model is used to support a print client operating in branch office print mode.<464>

Branch Office Print Remote Log Offline Archive: When branch office print remote logging is enabled (section 2.2.3.11), a print client can create certain event channel entries on the print server in response to Windows Events while processing a print job. Those entries are in the form of **Branch Office Print Remote Log Entries** (section 3.1.1) in Branch Office Print Remote Logging Structures (section 2.2.1.15), and they are sent to the server in a RPC_BranchOfficeJobDataContainer (section 2.2.1.2.17) by using the RpcLogJobInfoForBranchOffice (section 3.1.4.13.1) method.

If a print client is unable to contact the print server when processing a branch office print job, printing-related Windows Events are written to the **Branch Office Print Remote Log Offline Archive**. When contact with the print server is restored, those entries can be retransmitted.

Branch Office Print Remote Log Offline Archive Overflow Flag: If the Branch Office Print Remote Log Offline Archive exceeds its maximum size (section 2.2.3.11), the Branch Office Print Remote Log Offline Archive Overflow Flag is set to TRUE.<465> It is initialized to FALSE, and it is reset to

FALSE after the connection with the print server has been restored and the server has been notified of the overflow.

To notify the print server of an overflow of the offline archive, the print client creates a **Branch Office Print Remote Log Entry** in a RPC_BranchOfficeLogOfflineFileFull (section 2.2.1.15.7) structure and sends it to the server.

3.2.2 Timers

No protocol timers are required beyond those used internally by remote procedure call (RPC) ([MS-RPCE] section 3.2.3.2) to implement resiliency to network outages.

3.2.3 Initialization

The client MUST perform initialization according to the following rules when calling an RPC method:

- Either create an RPC binding handle to the server or use an RPC context handle. Details concerning binding handles are as specified in [C706].
- Use context handles across multiple calls to the server for methods taking a PRINTER_HANDLE.
- Use handles bound to a single call to the server for name-based methods taking a STRING_HANDLE. A STRING_HANDLE_BIND method MUST be implemented by the client.
- Reuse a context handle in multiple invocations when creating a print job, such as in a call to RpcOpenPrinter followed by multiple calls to RpcStartPagePrinter and RpcWritePrinter. For an example of this sequence of calls, see section 3.2.4.2.1.
- A context handle SHOULD be reused in multiple invocations when getting or setting information on a printer, such as in a call to RpcOpenPrinter followed by multiple calls to RpcGetPrinter, RpcGetPrinterData, RpcSetPrinter, or other methods taking a PRINTER_HANDLE or GDI_HANDLE.
- When creating the RPC binding handle on the named pipe **\pipe\spoolss**, the client MUST specify an **ImpersonationLevel** of 2 (**Impersonation** [MS-SMB2] (section 2.2.13).

3.2.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the remote procedure call (RPC) runtime ([MS-RPCE] section 3) that:

- It is to perform a strict network data representation (NDR) data consistency check at target level 6.0.
- It is to reject a NULL unique or full pointer with a nonzero conformant value.

The print client SHOULD ignore errors returned from the RPC server and SHOULD notify the application invoker of the error received in the higher layer. Unless otherwise specified, no special message processing is required on the client beyond that required in the underlying RPC protocol. <466>

3.2.4.1 Client-Side Notification Processing Methods

This section specifies processing rules for the notification-processing methods that a print client system MUST implement in its *Local Server* component to handle notifications from a print server.

Method	Description
RpcReplyOpenPrinter	Establishes a context handle from a print server to a print client.

Method	Description
	Opnum 58
RpcRouterReplyPrinter	Handles a notification from a print server. Opnum 59
RpcReplyClosePrinter	Closes the notification channel between a print server and a print client. Opnum 60
RpcRouterReplyPrinterEx	Handles a notification from a print server. Opnum 66

All these methods are request/response remote procedure call (RPC) methods. They MUST return zero to indicate successful completion and nonzero values to indicate failure, except where specifically described.

3.2.4.1.1 RpcReplyOpenPrinter (Opnum 58)

RpcReplyOpenPrinter establishes a context handle from a print server to a print client.<467> The server uses the RPC context handle returned by this method to send notification data to the client machine.

```
DWORD RpcReplyOpenPrinter(
  [in, string] STRING_HANDLE pMachine,
  [out] PRINTER HANDLE* phPrinterNotify,
  [in] DWORD dwPrinterRemote,
  [in] DWORD dwType,
  [in, range(0,512)] DWORD cbBuffer,
  [in, unique, size is(cbBuffer), disable consistency check]
    BYTE* pBuffer
);
```

pMachine: A string that specifies the print client computer name. It is synonymous with *pName*, as specified in Print Server Name Parameters (section 3.1.4.1.4).

phPrinterNotify: A pointer to a remote printer RPC context handle that will be used by a print server to send notifications to a print client. RPC context handles are specified in [C706].

dwPrinterRemote: A value that is supplied to the server by the *dwPrinterLocal* parameter of a corresponding call to RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3) or RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4). This value MUST NOT be zero.

dwType: A value that MUST be 0x00000001.

cbBuffer: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

pBuffer: A pointer that SHOULD be set to NULL when sent and MUST be ignored on receipt.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the client MUST validate parameters by verifying that the *pMachine* parameter corresponds to the current machine.

This method SHOULD execute without further access checks.

If parameter validation fails, the client MUST fail the operation immediately and return a nonzero error response to the server. Otherwise, the client MUST process the message as follows:

- Locate the notification state that is identified by the *dwPrinterRemote* parameter.
- Create a back channel RPC context handle and associate it with this notification state.
- Store the back channel RPC context handle in the handle pointed to by *phPrinterNotify*.
- Return the status of the operation.<468>

3.2.4.1.2 RpcRouterReplyPrinter (Opnum 59)

RpcRouterReplyPrinter handles a notification from a print server.<469>

```
DWORD RpcRouterReplyPrinter(
  [in] PRINTER_HANDLE hNotify,
  [in] DWORD fdwFlags,
  [in, range(0,512)] DWORD cbBuffer,
  [in, unique, size_is(cbBuffer), disable_consistency_check]
    BYTE* pBuffer
);
```

hNotify: A notification handle that was opened by the server using RpcReplyOpenPrinter (section 3.2.4.1.1).

fdwFlags: A value that contains Printer Change Flags (section 2.2.3.6), which indicate changes in printer configuration values.

cbBuffer: A value that SHOULD be set to zero when sent and MUST be ignored on receipt.

pBuffer: A pointer that SHOULD be set to NULL when sent and MUST be ignored on receipt.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the print client MUST validate parameters as follows:

- Verify that the *hNotify* parameter is an RPC context handle to a notification object opened by calling RpcReplyOpenPrinter. If that verification fails, ERROR_INVALID_HANDLE MUST be returned. This method SHOULD assume that this handle can be used without further access checks.

If parameter validation fails, the client MUST fail the operation immediately and return a nonzero error response to the server. Otherwise, the client MUST process the message as follows:

- Capture the *fdwFlags* in the notification state it maintains.
- If the operation is successful, the client MUST send the received data to the caller that registered for the notifications, by calling RpcRemoteFindFirstPrinterChangeNotification (section 3.1.4.10.3) or RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4).

3.2.4.1.3 RpcReplyClosePrinter (Opnum 60)

RpcReplyClosePrinter closes the notification channel between a print server and a print client.<470>

```
DWORD RpcReplyClosePrinter(
  [in, out] PRINTER HANDLE* phNotify
);
```

phNotify: A pointer to the notification context handle to close that was opened by RpcReplyOpenPrinter (section 3.2.4.1.1).

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the client MUST validate parameters as follows:

- Verify that the *phNotify* parameter is an RPC context handle to a notification object that was opened using RpcReplyOpenPrinter. If that verification fails, ERROR_INVALID_HANDLE MUST be returned. This method SHOULD assume that this handle can be used without further access checks.

If parameter validation fails, the client MUST fail the operation immediately and return a nonzero error response to the server. Otherwise, the client MUST process the message as follows:

- Free the context handle associated with the notification state.
- Return a response to the client containing the output parameters and the status of the operation.
- If the operation is successful, the client MUST modify the notification state by removing the back channel context handle associated with it.

3.2.4.1.4 RpcRouterReplyPrinterEx (Opnum 66)

RpcRouterReplyPrinterEx handles a notification from a print server.<471>

```
DWORD RpcRouterReplyPrinterEx(
  [in] PRINTER_HANDLE hNotify,
  [in] DWORD dwColor,
  [in] DWORD fdwFlags,
  [out] DWORD* pdwResult,
  [in] DWORD dwReplyType,
  [in, switch is(dwReplyType)] RPC V2 UREPLY PRINTER Reply
);
```

hNotify: A notification RPC context handle that was opened by RpcReplyOpenPrinter (section 3.2.4.1.1).

dwColor: The value that was most recently specified by the client in the *dwColor* parameter of a call to RpcRouterRefreshPrinterChangeNotification (section 3.1.4.10.5).

fdwFlags: A value that contains Printer Change Flags (section 2.2.3.6), which indicate changes in printer configuration values.

pdwResult: A pointer to a value that contains Change Notification Flags (section 2.2.3.2), which indicate how the client processed the notification.

dwReplyType: A value that MUST be zero.

Reply: A pointer to an RPC_V2_UREPLY_PRINTER union, which contains a pointer to an RPC_V2_NOTIFY_INFO structure, which contains available notification data that matched the set of notifications that the client previously requested.

Return Values: This method MUST return zero (ERROR_SUCCESS) to indicate successful completion or a nonzero Windows error code to indicate failure, as specified in [MS-ERREF].

Upon receiving this message, the print client MUST validate parameters as follows:

- Verify that the *hNotify* parameter is an RPC context handle to a notification object that was opened using RpcReplyOpenPrinter, and if that verification fails, return ERROR_INVALID_HANDLE, as specified in [MS-ERREF]. This method SHOULD assume that this handle can be used without further access checks.

- Verify that the value of the *dwColor* parameter matches the last value that was passed in the *dwColor* parameter in the call to RpcRouterRefreshPrinterChangeNotification; if that verification fails, set the **PRINTER_NOTIFY_INFO_COLORMISMATCH** bit in the variable pointed to by *pdwResult* and return 0.

If parameter validation fails, the client MUST fail the operation immediately and return a nonzero error response to the server. Otherwise, the client MUST process the message as follows:

- Capture the *fdwFlags* in the notification state it maintains.
- Capture the notification data provided in the *Reply* parameter in the notification state.
- Store the result of processing the notification (Change Notification Flags) to the variable pointed to by *pdwResult*.

3.2.4.2 Client Interaction with the Print Server

This section contains sequence specifications that the client MUST follow to perform specific tasks on the print server. See Protocol Examples (section 4) for additional information and sequence diagrams.

3.2.4.2.1 Printing a Document Using RpcStartDocPrinter

To print a document using RpcStartDocPrinter (section 3.1.4.9.1), the client MUST perform the following steps:

1. Invoke RpcOpenPrinter (section 3.1.4.2.2), supplying the name of the target printer in the *pPrinterName* parameter and an *AccessRequired* Access Value (section 2.2.3.1) that includes **PRINTER_ACCESS_USE**.
2. Using the printer handle obtained from RpcOpenPrinter:
    1. Invoke RpcStartDocPrinter to initiate the print job.
    2. For each page in the print job:
        1. Optionally invoke RpcStartPagePrinter to begin the page.
        2. Invoke RpcWritePrinter to send the client's print data to the printer.
        3. Optionally invoke RpcEndPagePrinter to end the page.
    3. Optionally invoke RpcEndDocPrinter to end the print job.
3. If the client called RpcEndDocPrinter in step 2.3, the client can perform the following:
    1. Step 2 can be repeated for additional print jobs.
    2. Optionally invoke RpcClosePrinter when finished.
4. If the client did not call RpcEndDocPrinter in step 2.3, the client MUST invoke RpcClosePrinter, and the server MUST process the call, assuming an implicit call to RpcEndDocPrinter.

3.2.4.2.2 Enumerating Printers on a Print Server

To enumerate the printers on a print server, the client MUST perform the following steps:

1. Invoke RpcEnumPrinters, supplying the name of the target print server in the *Name* parameter, the types of printers to enumerate in the *Flags* parameter, an information level value in *Level*, zero in *cbBuf*, and a pointer to a variable to store the required buffer size in *pcbNeeded*.

2. While RpcEnumPrinters returns with ERROR_INSUFFICIENT_BUFFER, the client MUST:

   - Allocate new printer information buffer space with a size from the returned value for *pcbNeeded*.
   - Invoke RpcEnumPrinters, supplying the name of the target print server in the *Name* parameter, the types of printers to enumerate in the *Flags* parameter, an information level value in *Level*, a pointer to the printer information buffer in *pPrinterEnum*, the allocated size of the printer information buffer in *cbBuf*, a pointer to a variable to store the required buffer size in *pcbNeeded*, and a location to store the number of printer information items returned in *pcReturned*.

   Note: Because the number of printers can change at any time, the client SHOULD be prepared to receive ERROR_INSUFFICIENT_BUFFER even after allocating the correct buffer size the first time.

3.2.4.2.3 Enumerating Jobs on a Printer

To enumerate the jobs that are currently queued to a printer, the client MUST perform the following steps:

1. Invoke RpcOpenPrinter, supplying the name of the target printer in the *pPrinterName* parameter and an *AccessRequired* value that includes PRINTER_ACCESS_USE.
2. Using the printer handle that was obtained from RpcOpenPrinter, the client MUST:
   1. Set a local job position context to the desired starting index, typically zero.
   2. Set a local number of jobs to return in a single operation.<472>
   3. Until an RpcEnumJobs call returns with a success status and a **pcReturned** pointing to a value of zero, or until the expected set of jobs has been returned, the client MUST:
      1. Invoke RpcEnumJobs, supplying the job position context in *FirstJob*, the number of jobs to return in a call in *NoJobs*, the desired information level in *Level*, a pointer to the job information buffer in *pJob*, the size of the job information buffer in *cbBuf*, a pointer to a variable to store the required buffer size in *pcbNeeded*, and a pointer to a variable to store the number of job information structures returned in *pcReturned*.
      2. While RpcEnumJobs returns with ERROR_INSUFFICIENT_BUFFER, as specified in [MS-ERREF], the client MUST:
         1. Allocate a new job information buffer with the size returned in *pcbNeeded*.
         2. Invoke RpcEnumJobs, supplying the job position context in *FirstJob*, the number of jobs to return in a call in *NoJobs*, the desired information level in *Level*, a pointer to the job information buffer in *pJob*, the size of the job information buffer in *cbBuf*, a pointer to a variable to store the required buffer size in *pcbNeeded*, and a pointer to a variable to store the number of job information structures in *pcReturned*.

         Note: Because the number of jobs can change at any time, the client SHOULD be prepared to receive ERROR_INSUFFICIENT_BUFFER even after allocating the correct buffer size the first time.
      3. Increase the local job position context by the value supplied in *pcReturned*.
3. The client SHOULD invoke RpcClosePrinter with the printer handle obtained from RpcOpenPrinter or SHOULD repeat step 2 if there are further job enumeration requests to make.

3.2.4.2.4 Receiving Notifications from a Print Server

To receive notifications for a printing event, a print client MUST perform the following steps:

1. Invoke RpcOpenPrinter (section 3.1.4.2.2), supplying the name of the target printer in the *pPrinterName* parameter and an *AccessRequired* value that includes **PRINTER_ACCESS_USE** from Access Values (section 2.2.3.1).

2. Using the printer handle that was obtained from RpcOpenPrinter, invoke RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4), supplying the notification flags and the job and printer fields that notifications are to be delivered for. The call MUST also supply a value in *dwPrinterLocal* that the client will use to identify the source for the later notifications.

   The print server opens a channel to the client as a result of processing this call by calling the client's RpcReplyOpenPrinter (section 3.2.4.1.1) method. Therefore, the client MUST implement the Print System Remote Protocol interface Client-Side Notification Processing Methods (section 3.2.4.1).

3. The client MUST process notifications as follows:

   - Process an RpcReplyOpenPrinter call, using the value in *dwPrinterRemote* to determine the client context established by the *dwPrinterLocal* parameter in a previous RpcRemoteFindFirstPrinterChangeNotificationEx call.

   - This call MUST produce an remote procedure call (RPC) handle that the later notifications use.

   - Process RpcRouterReplyPrinter (section 3.2.4.1.2) and RpcRouterReplyPrinterEx (section 3.2.4.1.4) method calls. The server calls RpcRouterReplyPrinter for notifications that do not have associated data and RpcRouterReplyPrinterEx for notifications that do have associated data. The value of *fdwFlags* MUST specify the notification type and any associated data.

4. To terminate the notifications, the client SHOULD invoke RpcFindClosePrinterChangeNotification, supplying the printer handle obtained from RpcOpenPrinter.

   The server closes the channel to the client as a result of processing this call by calling RpcReplyClosePrinter on the client.

5. The client MUST process an RpcReplyClosePrinter to terminate the notification sequence. This provides the RPC handle from the associated RpcReplyOpenPrinter call.

6. The client SHOULD call RpcClosePrinter. If the client has not called RpcFindClosePrinterChangeNotification in step 4, the server MUST implicitly close the notification channel, and in the processing of that, call RpcReplyClosePrinter, which MUST be processed by the client as specified in step 5.

3.2.4.2.5 Announcing Shared Printers to Print Servers

To announce its shared printers to print servers, the client MUST perform these steps:

1. Make a policy-specific determination whether shared printers should be enumerated to print servers.<473>

2. If shared printers should be enumerated, for each printer installed on the client that has the PRINTER_ATTRIBUTE_SHARED set, create a PRINTER_CONTAINER with **Level** set to 0x00000001, and populate it with a PRINTER_INFO_1 describing the printer, and then call the print server's RpcAddPrinter or RpcAddPrinterEx method.<474>

3.2.4.2.6 Adding a Printer to a Print Server

To add a printer to a print server, the client performs these steps:

1. The client MAY use methods defined by this protocol to query the print server for information used to initialize other data structures.<475>

2. The client SHOULD call the print server's RpcEnumPrinterDrivers to determine whether a printer driver for the new printer is already installed on the server.

3. If a printer driver is not already installed, the client SHOULD call RpcAddPrinterDriver or RpcAddPrinterDriverEx to install a printer driver for the new printer.

4. The client MUST allocate a PRINTER_CONTAINER structure and populate it with a PRINTER_INFO_2 structure describing the new printer.

5. The client MUST allocate a DEVMODE_CONTAINER and populate it with the default DEVMODE for the new printer.

6. The client MUST allocate a SECURITY_CONTAINER and populate it with a SECURITY_DESCRIPTOR containing the security information for the new printer.

7. The client MUST call the print server's RpcAddPrinter with the print server's name, and the CONTAINER parameters from steps 4, 5, and 6. Alternatively, the client can use the RpcAddPrinterEx and specify an additional SPLCLIENT_CONTAINER that describes the client in more detail. RpcAddPrinterEx returns a PRINTER_HANDLE to the newly added printer in the variable pointed to by *pHandle*. The client SHOULD close that handle using RpcClosePrinter when it no longer requires it.

3.2.5 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying RPC protocol.

3.2.6 Other Local Events

A client's invocation of each method is typically the result of local application activity. The local application on the client computer specifies values for all input parameters. No other higher-layer triggered events are processed. The values specified for input parameters are described in section 2.

No additional local events are used on the client beyond the events maintained in the underlying remote procedure call (RPC) protocol.

4 Protocol Examples

4.1 Adding a Printer to a Server

To add a printer ("HP LaserJet 4") to a print server ("CORPSERV"), a client ("TESTCLT") performs the following steps:

1. Enumerate existing printer drivers using RpcEnumPrinterDrivers.

    - The client obtains the server platform by calling RpcGetPrinterData on a server object with the "Architecture" key value.
    - The client calls RpcEnumPrinterDrivers, specifying the environment parameter to match the server platform.

```
RpcEnumPrinterDrivers( L"\\\\CORPSERV"., L"Windows NT x86", 1, NULL, 0,
&countBytesNeeded, &driversFound )
```

    - The server returns ERROR_INSUFFICIENT_BUFFER and sets countBytesNeeded large enough to store _DRIVER_INFO_1 structures for all drivers matching the specified environment "Windows NT x86".
    - The client allocates memory, setting the size of driverInfo1[] to countBytesNeeded, and calls RpcEnumPrinterDrivers again.

```
RpcEnumPrinterDrivers ( L"\\\\CORPSERV", L"Windows NT x86", 1, driverInfo1,
countBytesNeeded, &countBytesNeeded, &driversFound )
```

    - The server writes a _DRIVER_INFO_1 structure for each driver matching the specified environment ("Windows NT x86") to the output buffer, writes the number of _DRIVER_INFO_1 structures to driversFound, and returns 0 (success).

    Note: If the number of drivers on the server has increased between the first and second call to RpcEnumPrinterDrivers, the server returns ERROR_INSUFFICIENT_BUFFER from the second call as well. In that case, the server updates countBytesNeeded and the client needs to allocate more memory and call RpcEnumPrinterDrivers again.

2. Select an existing printer driver or add a new printer driver using RpcAddPrinterDriver.

    - Assume the server returned a driver named "HP LaserJet 4".
    - The client calls RpcEnumPorts to enumerate the available ports. This process is analogous to the previous step, which enumerated printer drivers using RpcEnumPrinterDrivers.
    - The client displays a dialog box so the end user can pick the driver, enter a desired port, and optionally enter a share name for the new printer.
    - If the driver does not already exist, or the client requests to update the driver, use RpcAddPrinterDriver to add the driver to the print server, as shown in 4.2.

3. Populate a PRINTER_INFO_2 structure with information about the new printer and call RpcAddPrinter.

    - The client allocates and zero-initializes a PRINTER_INFO_2 structure.
    - The client sets the following members of the structure:

```
pPrinterName = L"HP LaserJet 4"  /* Typically set to the driver name */
pShareName = L"My Printer"       /* Any name the user selects */
pPortName = L"172.10.10.10"      /* A port that exists on the server */
pDriverName = L"HP LaserJet 4"   /* Driver selected in previous step */
```

For pPortName, the client may have previously enumerated the server's ports using RpcEnumPorts or the user may know a valid port name.

The client initializes all other members of the structure to zero or NULL, as appropriate. Or, the client specifies higher DRIVER_INFO levels in the call to RpcEnumPrinterDrivers to obtain more details to initialize these structure members. Or, the client prompts the user to specify values such as priorities or times at which the printer is available and initializes these members with the values specified by the user.

- The client allocates a PRINTER_CONTAINER (section 2.2.1.2.9) structure and initializes it to contain the prepared PRINTER_INFO_2.
- The client allocates a DEVMODE_CONTAINER devmodeContainer structure, and optionally initializes it with a DEVMODE structure.
- The client allocates a SECURITY_CONTAINER securityContainer structure, and optionally initializes it with a SECURITY_DESCRIPTOR.
- The client calls RpcAddPrinter to add the printer.

```
RpcAddPrinter( L"\\\\CORPSERV", &printerContainer, &devmodeContainer, &securityContainer,
&hPrinter )
```

- The server creates the print queue, writes the handle to hPrinter, and returns 0 (success).

4. Close the returned PRINTER_HANDLE using RpcClosePrinter.
   - When the client is done using the print queue, the client closes it.

```
RpcClosePrinter( &hPrinter )
```

   - The server frees the memory associated with the print queue handle, sets hPrinter to NULL, and returns 0 (success).

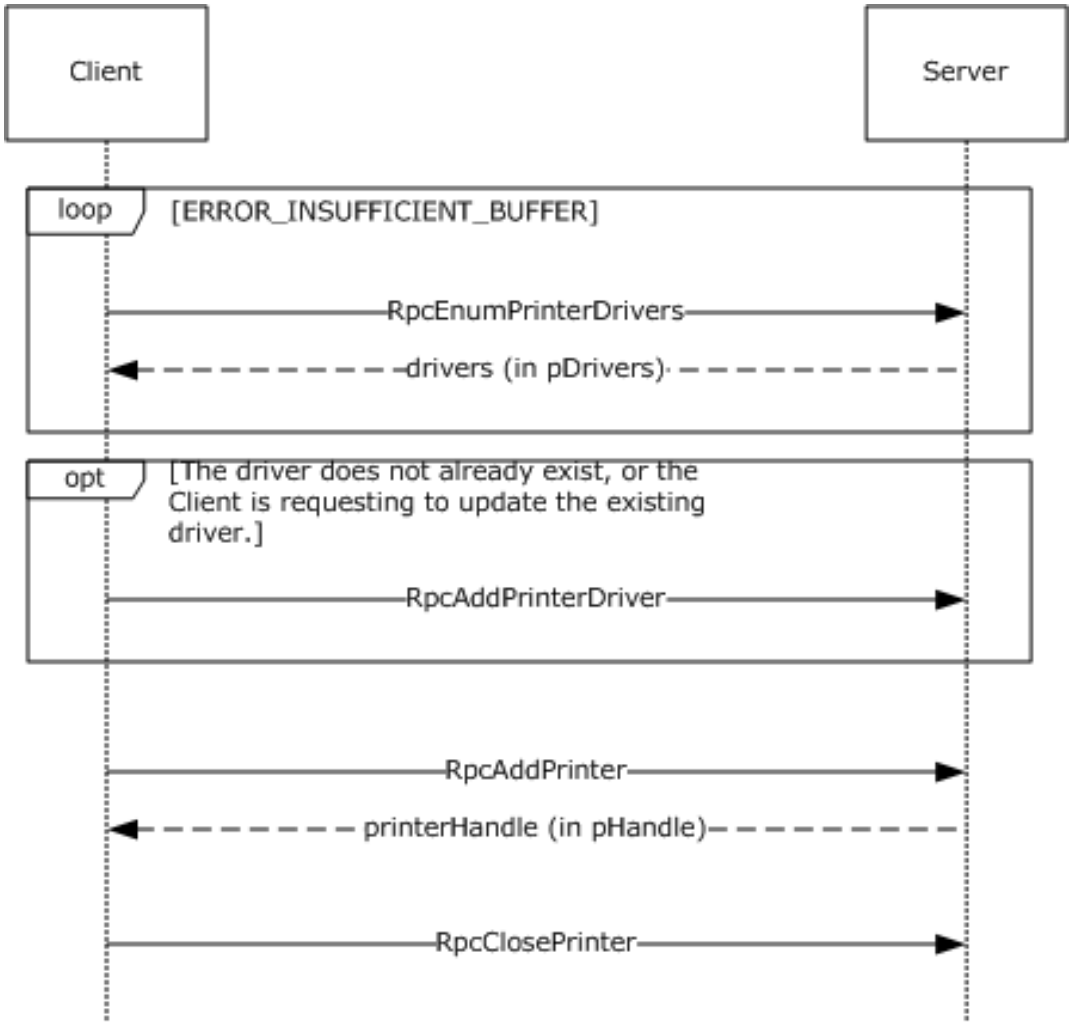


Figure 5: Adding a printer to a server

4.2 Adding a Printer Driver to a Server

To add or update a printer driver ("OEM Printer Driver") to a print server ("CORPSERV"), a client ("TESTCLT") performs the following steps.

1. Enumerate existing printer drivers using RpcEnumPrinterDrivers.

   See 4.1 for an example using RpcEnumPrinterDrivers.

2. If the printer driver does not already exist or the client requests to update the printer driver, use RpcAddPrinterDriver to add the driver to the print server.

   - The client must make sure the files for the printer driver are in a location accessible to the server. For that purpose, the client can share a local directory containing the files, or use [MS-SMB] to place the files into a directory on the server.
   - The client then allocates and populates a DRIVER_INFO_2 structure as follows:

```
pName = L"OEM Printer Driver";
pEnvironment = L"Windows NT x86";    /* Environment the driver is compatible with */
pDriverPath = "\\\\CORPSERV\\C$\\DRIVERSTAGING\\OEMDRV.DLL";
```

```
pDataFile = "\\\\CORPSERV\\C$\\DRIVERSTAGING\\OEMDATA.DLL";
pConfigFile = "\\\\CORPSERV\\C$\\DRIVERSTAGING\\OEMUI.DLL";
```

- The client allocates a DRIVER_CONTAINER driverContainer structure and initializes it to contain the DRIVER_INFO_2 structure.
- The client calls RpcAddPrinterDriver.

```
RpcAddPrinterDriver( L"\\\\CORPSERV", &driverContainer );
```

- The server adds the printer driver and returns 0 (success).

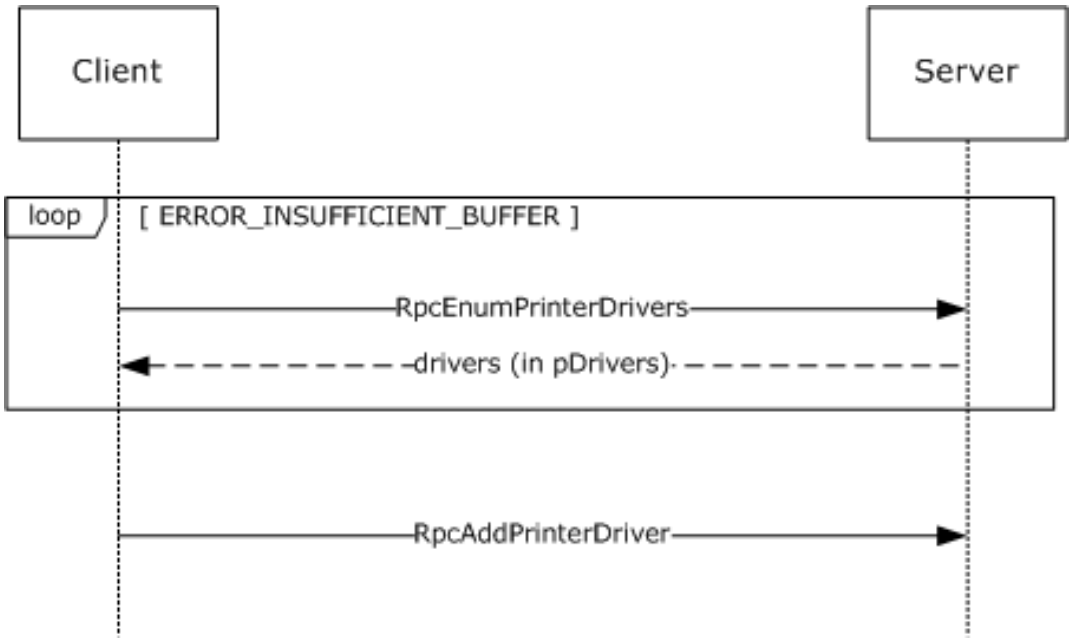


Figure 6: Adding a printer driver to a server

4.3 Enumerating and Managing Printers

To manage printers on a print server ("CORPSERV"), a client ("TESTCLT") performs the following steps.

1. Enumerate existing printers using RpcEnumPrinters.
   - The client calls RpcEnumPrinters.

```
RpcEnumPrinters( PRINTER_ENUM_NAME, L"\\\\CORPSERV", 2, NULL, 0, &countBytesNeeded,
&printersFound );
```

   - The server returns ERROR_INSUFFICIENT_BUFFER and sets countBytesNeeded to the size needed to store _PRINTER_INFO_2 structures for all shared print queues.
   - The client allocates memory in printerInfo2[] with the size set to countBytesNeeded.
   - The client calls RpcEnumPrinters.

```
RpcEnumPrinters( PRINTER_ENUM_NAME, L"\\\\CORPSERV", 2, printerInfo2, countBytesNeeded,
&countBytesNeeded, &printersFound );
```

- The server writes a _PRINTER_INFO_2 structure for each shared print queue to the output buffer, writes the number of _PRINTER_INFO_2 structures to printersFound, and returns 0 (success).

  Note: If the number of shared print queues on the server has increased between the first and second call to RpcEnumPrinters, the server returns ERROR_INSUFFICIENT_BUFFER from the second call as well. In that case, the server will update countBytesNeeded and the client needs to reallocate more memory and repeat the call to RpcEnumPrinters.

2. Open a handle to the print queue using RpcOpenPrinter.

   - The client selects a print queue from the _PRINTER_INFO_2 structure and uses the **pPrinterName** or **pShareName** to open the print queue handle as follows:

     - The client allocates and initializes a DEVMODE_CONTAINER devmodeContainer structure.
     - The client calls RpcOpenPrinter.

```
RpcOpenPrinter( L"\\\\CORPSERV\\My Printer", &hPrinter, L"RAW", &devmodeContainer,
PRINTER_ACCESS_USE);
```

   - The server allocates printer handle, writes it to hPrinter, and returns 0 (success).

3. Retrieve current information about a printer using RpcGetPrinter.

   - The client calls RpcGetPrinter.

```
RpcGetPrinter(hPrinter, 2, NULL, 0, &countBytesNeeded);
```

   - The server returns ERROR_INSUFFICIENT_BUFFER and sets countBytesNeeded to store a _PRINTER_INFO_2 structure for the print queue.
   - The client allocates memory in printerInfo2[] with size set to countBytesNeeded.
   - The client calls RpcGetPrinter.

```
RpcGetPrinter( hPrinter, 2, printerInfo2, countBytesNeeded, &countBytesNeeded );
```

   - The server writes a _PRINTER_INFO_2 structure for the print queue to the output buffer and returns 0 (success).

     Note: If the size of data for the print queue on the server has increased between the first and second call to RpcGetPrinter, the server returns ERROR_INSUFFICIENT_BUFFER from the second call as well. That can happen under a race condition if another client changes the print queue data. In that case, the server will update countBytesNeeded and the client needs to reallocate more memory and repeat the call to RpcGetPrinter.

4. Use RpcSetPrinter to modify the state of the printer.

   - The client allocates a PRINTER_INFO_2 structure and populates it with members from the previously acquired _PRINTER_INFO_2. The client changes those members that require change:

```
pLocation = L"Building 84, Room 1129";
```

- The client allocates a PRINTER_CONTAINER printerContainer structure and initializes it to contain the prepared PRINTER_INFO_2.
- The client allocates a DEVMODE_CONTAINER devmodeContainer structure, and optionally initializes it with a DEVMODE structure.
- The client allocates a SECURITY_CONTAINER securityContainer structure, and optionally initializes it with a SECURITY_DESCRIPTOR.
- The client calls RpcSetPrinter.

```
RpcSetPrinter( hPrinter, &printerContainer, &devmodeContainer, &securityContainer, 0 );
```

- The server modifies the print queue and returns 0 (success).

5. Close the printer using RpcClosePrinter.
   - The client calls RpcClosePrinter.

```
RpcClosePrinter( &hPrinter );
```

   - The server frees the memory associated with the print queue handle, sets hPrinter to NULL, and returns 0 (success).

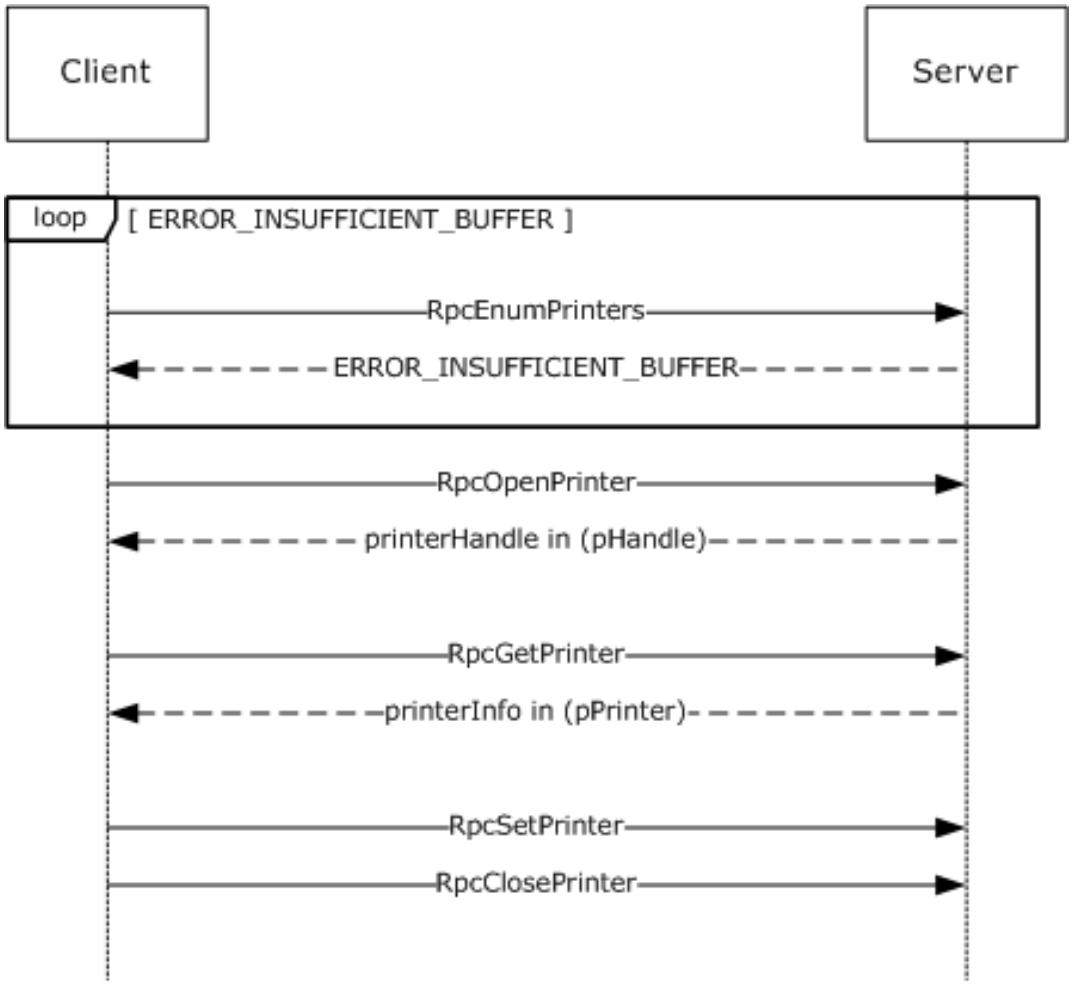


Figure 7: Enumerating and managing printers on a server

4.4 Enumerating Jobs and Modifying Job Settings

To enumerate print jobs on a server ("CORPSERV"), modify job settings, or change job priorities, the client ("TESTCLT") performs the following steps.

1. Open the printer using RpcOpenPrinter.

    - The client allocates and initializes a *devmodeContainer* structure (see DEVMODE_CONTAINER (section 2.2.1.2.1).
    - The client calls RpcOpenPrinter.

```
RpcOpenPrinter( L"\\\\CORPSERV\\My Printer", &hPrinter, L"RAW", &devmodeContainer,
PRINTER_ACCESS_USE );
```

    - The server allocates a printer handle, writes it to *hPrinter*, and returns 0 (success).

2. Enumerate jobs scheduled for printing on the printer using RpcEnumJobs.

    - The client calls RpcEnumJobs with *FirstJob* set to 0 and *NoJobs* set to the maximum unsigned integer to return all jobs.

```
RpcEnumJobs( hPrinter, 0, 0xFFFFFFFF, 1, NULL, 0, &countBytesNeeded, &jobsFound );
```

- The server returns ERROR_INSUFFICIENT_BUFFER and sets *countBytesNeeded* to store _JOB_INFO_1 structures for all shared print queues.
- The client allocates memory in *jobInfo1[]* with size set to *countBytesNeeded*.
- The client calls RpcEnumJobs.

```
RpcEnumJobs( hPrinter, 0, 0xFFFFFFFF, 1, jobInfo1, countBytesNeeded, &countBytesNeeded,
&jobsFound );
```

- The server writes _JOB_INFO_1 for all jobs on the print queue to the output buffer, writes the number of _JOB_INFO_1 structures to *jobsFound*, and returns 0 (success).

  Note: If the number of jobs on the print queue on the server has increased between the first and second call to RpcEnumJobs, the server returns ERROR_INSUFFICIENT_BUFFER from the second call as well. In that case, the server will update *countBytesNeeded* and the client is required to reallocate more memory and repeat the call to RpcEnumJobs.

3. Modify job settings or job priority using RpcSetJob.
   - The client picks a job from the list of _JOB_INFO_1 structures that it requests to modify. For this example, we assume the **JobId** is 12 and we want to cancel the job.
   - The client allocates and zero-initializes a *jobContainer* structure (see JOB_CONTAINER (section 2.2.1.2.5).
   - To modify job settings, the client calls RpcSetJob with the *Command* parameter set to zero.

```
RpcSetJob( hPrinter, 12, &jobContainer, 0 );
```

   - To control the processing of a job, such as to cancel it, the client calls RpcSetJob with a non-zero *Command* parameter.

```
RpcSetJob( hPrinter, 12, &jobContainer, JOB_CONTROL_CANCEL );
```

   - The server modifies the specified print job and returns 0 (success).
4. The client closes the printer using RpcClosePrinter.

```
RpcClosePrinter( &hPrinter );
```

   The server frees the memory associated with the print queue handle, sets *hPrinter* to NULL, and returns 0 (success).

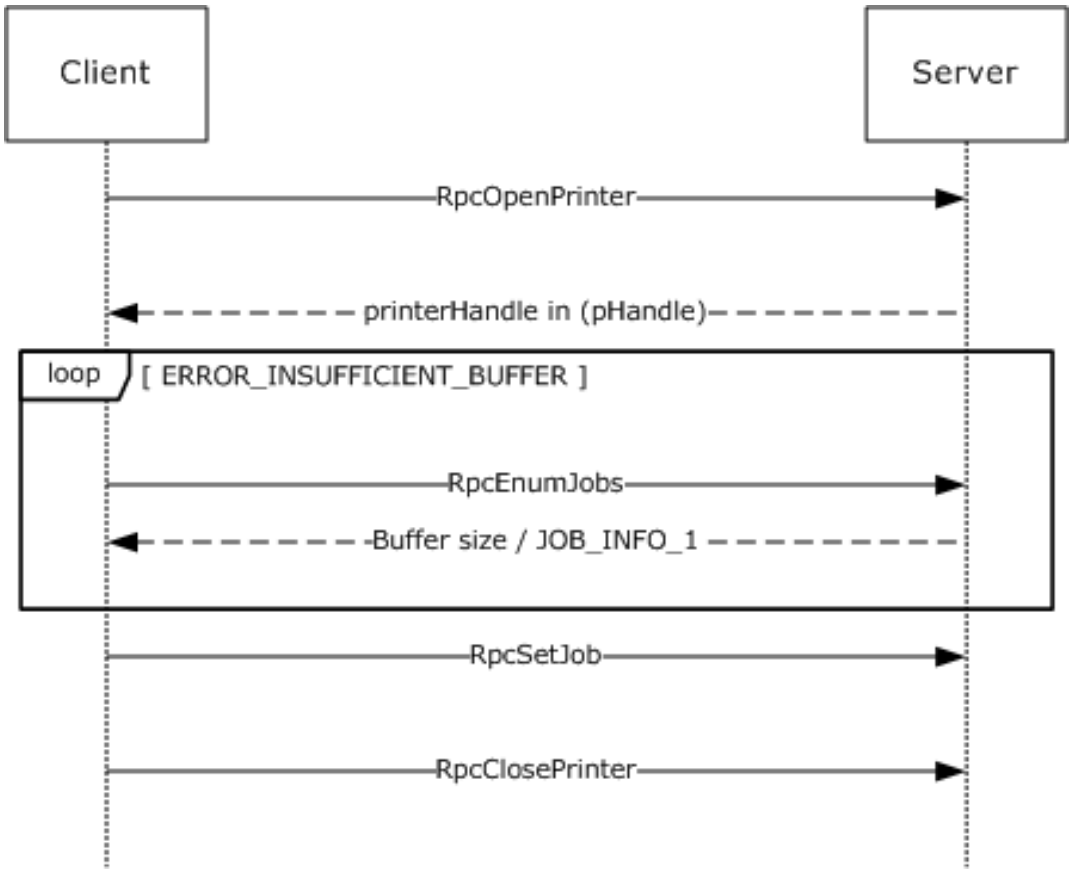


Figure 8: Enumerating jobs and modifying job settings

4.5 Receiving Notifications on Printing Events

To receive notifications concerning state changes of print servers, printers, and print jobs, a client ("TESTCLT") can perform the following steps, as shown in the figure in Notification of Print System Changes (section 1.3.3).

1. Open a handle to a printer using RpcOpenPrinter (section 3.1.4.2.2).

   - The client calls RpcOpenPrinter.

     ```
     RpcOpenPrinter( L"\\\\CORPSERV\\My Printer", &hPrinter, L"RAW", &devmodeContainer,
     PRINTER_ACCESS_USE );
     ```

   - The server allocates a printer handle, writes it to hPrinter, and returns 0 (success).

2. Register for change notifications using RpcRemoteFindFirstPrinterChangeNotificationEx (section 3.1.4.10.4):

   - The client allocates and initializes a RPC_V2_NOTIFY_OPTIONS notifyOptions structure as follows:

     ```
     WORD notifyFieldsJob[] = { 0x000A /*JOB_NOTIFY_FIELD_STATUS*/, 0x000D
     /*JOB_NOTIFY_FIELD_DOCUMENT*/ };
     RPC_V2_NOTIFY_OPTIONS_TYPE notifyTypes[1] = {{1 /*JOB_NOTIFY_TYPE*/, 0, 0, 0, 2,
     notifyFieldsJob }};
     ```

```
RPC_V2_NOTIFY_OPTIONS notifyOptions = {0x00000002,0x00000000,1,notifyTypes};
```

- The client calls RpcRemoteFindFirstPrinterChangeNotificationEx.

```
RpcRemoteFindFirstPrinterChangeNotificationEx( hPrinter, 0x00000100 /*
PRINTER CHANGE ADD JOB */, 0, L"\\\\TESTCLT", 4711, &notifyOptions );
/* The number 4711 is a unique number used as a cookie to match the server's response. */
```

- The server calls the client's RpcReplyOpenPrinter (section 3.2.4.1.1) method to open a reverse channel, which will be used to send change notifications to the client. The client has to return a remote procedure call (RPC) binding handle that identifies the reverse channel.
  - The server calls RpcReplyOpenPrinter.

```
RpcReplyOpenPrinter( L"\\\\TESTCLT", &hPrinterNotify, 4711, 1, 0, NULL );
```

  - The client opens a notification context and associates it with the open printer handle hPrinter. In order to do so, client matches the dwPrinterRemote value (4711 in this example).
  - The client writes the notification context handle to hPrinterNotify and returns 0 (success) to the server.
  - The server returns 0 (success) from processing RpcRemoteFindFirstPrinterChangeNotificationEx.
- As long as the client stays registered for notifications, the server calls the client's RpcRouterReplyPrinter (section 3.2.4.1.2) or RpcRouterReplyPrinterEx (section 3.2.4.1.4) method for each change of the requested type that occurs on the server.

  Whenever the monitored print queue changes on the server, the server filters the change according to the filter options specified by the client in the registration call and:
  - The server allocates and initializes an RPC_V2_UREPLY_PRINTER reply structure:

```
RPC_V2_NOTIFY_INFO notifyInfo;  /* Note: Pseudo-code only, assumes sufficient memory has
been allocated for aData[] array at end of structure */
notifyInfo.Version = 2;
notifyInfo.Flags = 0;
notifyInfo.Count = 1;
notifyInfo.aData[0].Type = 1;   /* JOB NOTIFY TYPE */
notifyInfo.aData[0].Field = 0xD /* JOB_NOTIFY_FIELD_DOCUMENT */
notifyInfo.aData[0].String.pszString = L"My Test Print Job Name";
notifyInfo.aData[0].Id = 12;    /* This print job has ID 12 */
RPC V2 UREPLY PRINTER reply;
Reply.pInfo = &notifyInfo;
```

  - The server calls the client's RpcRouterReplyPrinterEx.

```
RpcRouterReplyPrinterEx( hPrinterNotify, 1, 0x00000100 /* PRINTER_CHANGE_ADD_JOB */,
&result, 0, &reply );
```

  - The client reflects the change in its internal state.
  - The client writes processing flags to the variable result and returns 0 (success).

3. Route the change notifications to applications or process it to reflect state changes.

   The client makes local calls to applications or processes to notify them of the state change.

4. When state changes are no longer accepted, the client unregisters from notifications by calling RpcFindClosePrinterChangeNotification (section 3.1.4.10.2) with the handle returned by the call to RpcOpenPrinter.

```
RpcFindClosePrinterChangeNotification( hPrinter );
```

   - The server calls the client's RpcReplyClosePrinter (section 3.2.4.1.3) with the handle previously obtained by RpcReplyOpenPrinter, notifying the client to close the binding handle for the reverse channel.

```
RpcReplyClosePrinter( &hPrinterNotify );
```

   - The client cleans up the notification context, writes NULL to hPrinterNotify, and returns 0 (success).
   - The server returns 0 (success) from RpcFindClosePrinterChangeNotification.

5. The client closes the handle to the printer or server object using RpcClosePrinter (section 3.1.4.2.9).

```
RpcClosePrinter( &hPrinter );
```

   The server frees the memory associated with the print queue handle, sets hPrinter to NULL, and returns 0 (success).

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

Security considerations for both authenticated and unauthenticated RPC are specified in [C706] chapters Introduction to the RPC API and Security.<476>

6 Appendix A: Full IDL

For ease of implementation, the full stand-alone Interface Definition Language (IDL) is provided. Some of the data types and structures used by this protocol are defined in other documents. In order for this IDL to stand alone, those types and structures, from [MS-DTYP], are included below.

```
// [MS-RPRN] interface
[
    uuid(12345678-1234-ABCD-EF00-0123456789AB),
    version(1.0),
    ms union,
    endpoint("ncacn_np:[\\pipe\\spoolss]"),
    pointer_default(unique)
]
interface winspool {

import "ms-dtyp.idl";

#if __midl < 700
#define disable_consistency_check
#endif

// [MS-RPRN] common constants
#define TABLE_DWORD              0x1
#define TABLE_STRING             0x2
#define TABLE_DEVMODE            0x3
#define TABLE_TIME               0x4
#define TABLE_SECURITYDESCRIPTOR 0x5
#define SPLFILE_CONTENT_TYPE_PROP_NAME L"Spool File Contents"

// [MS-RPRN] common enumerations
typedef enum {
    BIDI_NULL   = 0,
    BIDI_INT    = 1,
    BIDI_FLOAT  = 2,
    BIDI_BOOL   = 3,
    BIDI_STRING = 4,
    BIDI_TEXT   = 5,
    BIDI_ENUM   = 6,
    BIDI_BLOB   = 7
} BIDI_TYPE;

typedef enum {
    kRpcPropertyTypeString = 1,
    kRpcPropertyTypeInt32,
    kRpcPropertyTypeInt64,
    kRpcPropertyTypeByte,
    kRpcPropertyTypeBuffer
} RPC_EPrintPropertyType;

// [MS-RPRN] common data types
typedef unsigned short LANGID;
typedef [context_handle] void* GDI_HANDLE;
typedef [context_handle] void* PRINTER_HANDLE;
typedef [handle] wchar_t* STRING_HANDLE;

// [MS-RPRN] common utility structures
typedef struct {
    long cx;
    long cy;
} SIZE;

typedef struct {
    long left;
    long top;
    long right;
    long bottom;
```

```
} RECTL;


// [MS-RPRN] common device state structure
typedef struct _devicemode {
    wchar_t  dmDeviceName[32];

    unsigned short dmSpecVersion;
    unsigned short dmDriverVersion;
    unsigned short dmSize;
    unsigned short dmDriverExtra;

    DWORD dmFields;

    short dmOrientation;
    short dmPaperSize;
    short dmPaperLength;
    short dmPaperWidth;
    short dmScale;
    short dmCopies;
    short dmDefaultSource;
    short dmPrintQuality;
    short dmColor;
    short dmDuplex;
    short dmYResolution;
    short dmTTOption;
    short dmCollate;

    wchar_t  dmFormName[32];

    unsigned short reserved0;

    DWORD reserved1;
    DWORD reserved2;
    DWORD reserved3;
    DWORD dmNup;
    DWORD reserved4;
    DWORD dmICMMethod;
    DWORD dmICMIntent;
    DWORD dmMediaType;
    DWORD dmDitherType;
    DWORD reserved5;
    DWORD reserved6;
    DWORD reserved7;
    DWORD reserved8;
} DEVMODE;

// [MS-RPRN] common info structures
typedef struct  DOC INFO 1 {
    [string] wchar_t* pDocName;
    [string] wchar_t* pOutputFile;
    [string] wchar_t* pDatatype;
} DOC INFO 1;

typedef struct  DRIVER INFO 1 {
    [string] wchar_t* pName;
} DRIVER_INFO_1;

typedef struct _DRIVER_INFO_2 {
    DWORD cVersion;
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar_t* pDriverPath;
    [string] wchar_t* pDataFile;
    [string] wchar_t* pConfigFile;
} DRIVER INFO 2;

typedef struct _RPC_DRIVER_INFO_3 {
```

```
    DWORD cVersion;
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar_t* pDriverPath;

    [string] wchar_t* pDataFile;
    [string] wchar_t* pConfigFile;
    [string] wchar_t* pHelpFile;
    [string] wchar_t* pMonitorName;
    [string] wchar_t* pDefaultDataType;
    DWORD cchDependentFiles;
    [size_is(cchDependentFiles), unique]
      wchar_t* pDependentFiles;
} RPC DRIVER INFO 3;

typedef struct  RPC DRIVER INFO 4 {
    DWORD cVersion;
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar_t* pDriverPath;
    [string] wchar_t* pDataFile;
    [string] wchar_t* pConfigFile;
    [string] wchar_t* pHelpFile;
    [string] wchar_t* pMonitorName;
    [string] wchar_t* pDefaultDataType;
    DWORD cchDependentFiles;
    [size_is(cchDependentFiles), unique]
      wchar_t* pDependentFiles;
    DWORD cchPreviousNames;
    [size_is(cchPreviousNames), unique]
      wchar_t* pszzPreviousNames;
} RPC_DRIVER_INFO_4;

typedef struct _RPC_DRIVER_INFO_6 {
    DWORD cVersion;
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar_t* pDriverPath;
    [string] wchar_t* pDataFile;
    [string] wchar_t* pConfigFile;
    [string] wchar_t* pHelpFile;
    [string] wchar_t* pMonitorName;
    [string] wchar_t* pDefaultDataType;
    DWORD cchDependentFiles;
    [size_is(cchDependentFiles), unique]
      wchar_t* pDependentFiles;
    DWORD cchPreviousNames;
    [size_is(cchPreviousNames), unique]
      wchar_t* pszzPreviousNames;
    FILETIME ftDriverDate;
    DWORDLONG dwlDriverVersion;
    [string] wchar_t* pMfgName;
    [string] wchar_t* pOEMUrl;
    [string] wchar_t* pHardwareID;
    [string] wchar_t* pProvider;
} RPC DRIVER INFO 6;

typedef struct _RPC_DRIVER_INFO_8 {
    DWORD cVersion;
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar_t* pDriverPath;
    [string] wchar_t* pDataFile;
    [string] wchar_t* pConfigFile;
    [string] wchar_t* pHelpFile;
    [string] wchar_t* pMonitorName;
    [string] wchar_t* pDefaultDataType;
    DWORD cchDependentFiles;
    [size_is(cchDependentFiles), unique]
```

```
      wchar_t* pDependentFiles;
    DWORD cchPreviousNames;
    [size_is(cchPreviousNames), unique]
      wchar_t* pszzPreviousNames;
    FILETIME ftDriverDate;

    DWORDLONG dwlDriverVersion;
    [string] wchar t* pMfgName;
    [string] wchar_t* pOEMUrl;
    [string] wchar t* pHardwareID;
    [string] wchar_t* pProvider;
    [string] wchar_t* pPrintProcessor;
    [string] wchar_t* pVendorSetup;
    DWORD cchColorProfiles;
    [size_is(cchColorProfiles), unique]
      wchar t* pszzColorProfiles;
    [string] wchar_t* pInfPath;
    DWORD dwPrinterDriverAttributes;
    DWORD cchCoreDependencies;
    [size_is(cchCoreDependencies), unique]
      wchar t* pszzCoreDriverDependencies;
    FILETIME ftMinInboxDriverVerDate;
    DWORDLONG dwlMinInboxDriverVerVersion;
} RPC_DRIVER_INFO_8;

typedef struct _FORM_INFO_1 {
    DWORD Flags;
    [string] wchar_t* pName;
    SIZE Size;
    RECTL ImageableArea;
} FORM INFO 1;

typedef struct  RPC FORM INFO 2 {
    DWORD Flags;
    [string, unique] const wchar_t* pName;
    SIZE Size;
    RECTL ImageableArea;
    [string, unique] const char* pKeyword;
    DWORD StringType;
    [string, unique] const wchar_t* pMuiDll;
    DWORD dwResourceId;
    [string, unique] const wchar t* pDisplayName;
    LANGID wLangID;
} RPC FORM INFO 2;

typedef struct _JOB_INFO_1 {
    DWORD JobId;
    [string] wchar t* pPrinterName;
    [string] wchar_t* pMachineName;
    [string] wchar t* pUserName;
    [string] wchar_t* pDocument;
    [string] wchar_t* pDatatype;
    [string] wchar_t* pStatus;
    DWORD Status;
    DWORD Priority;
    DWORD Position;
    DWORD TotalPages;
    DWORD PagesPrinted;
    SYSTEMTIME Submitted;
} JOB INFO 1;

typedef struct  JOB INFO 2 {
    DWORD JobId;
    [string] wchar_t* pPrinterName;
    [string] wchar_t* pMachineName;
    [string] wchar_t* pUserName;
    [string] wchar t* pDocument;
    [string] wchar_t* pNotifyName;
    [string] wchar_t* pDatatype;
```

```
    [string] wchar_t* pPrintProcessor;
    [string] wchar_t* pParameters;
    [string] wchar_t* pDriverName;
    ULONG_PTR pDevMode;
    [string] wchar_t* pStatus;
    ULONG_PTR pSecurityDescriptor;

    DWORD Status;
    DWORD Priority;
    DWORD Position;
    DWORD StartTime;
    DWORD UntilTime;
    DWORD TotalPages;
    DWORD Size;
    SYSTEMTIME Submitted;
    DWORD Time;
    DWORD PagesPrinted;
} JOB_INFO_2;

typedef struct _JOB_INFO_3 {
    DWORD JobId;
    DWORD NextJobId;
    DWORD Reserved;
} JOB_INFO_3;

typedef struct _JOB_INFO_4 {
    DWORD JobId;
    [string] wchar_t* pPrinterName;
    [string] wchar_t* pMachineName;
    [string] wchar_t* pUserName;
    [string] wchar t* pDocument;
    [string] wchar_t* pNotifyName;
    [string] wchar t* pDatatype;
    [string] wchar_t* pPrintProcessor;
    [string] wchar_t* pParameters;
    [string] wchar_t* pDriverName;
    ULONG PTR pDevMode;
    [string] wchar_t* pStatus;
    ULONG PTR pSecurityDescriptor;
    DWORD Status;
    DWORD Priority;
    DWORD Position;
    DWORD StartTime;
    DWORD UntilTime;
    DWORD TotalPages;
    DWORD Size;
    SYSTEMTIME Submitted;
    DWORD Time;
    DWORD PagesPrinted;
    long SizeHigh;
} JOB_INFO_4;

typedef struct _MONITOR_INFO_1 {
    [string] wchar t* pName;
} MONITOR_INFO_1;

typedef struct _MONITOR_INFO_2 {
    [string] wchar_t* pName;
    [string] wchar_t* pEnvironment;
    [string] wchar t* pDLLName;
} MONITOR_INFO_2;

typedef struct _PORT_INFO_1 {
    [string] wchar_t* pPortName;
} PORT_INFO_1;

typedef struct  PORT INFO 2 {
    [string] wchar_t* pPortName;
    [string] wchar_t* pMonitorName;
```

```
    [string] wchar_t* pDescription;
    DWORD fPortType;
    DWORD Reserved;
} PORT_INFO_2;

typedef struct  PORT INFO 3 {
    DWORD dwStatus;

    [string] wchar_t* pszStatus;
    DWORD dwSeverity;
} PORT_INFO_3;

typedef struct _PORT_INFO_FF {
    [string] wchar t* pPortName;
    DWORD cbMonitorData;
    BYTE* pMonitorData;
} PORT_INFO_FF;

typedef struct  PRINTER INFO STRESS {
    [string] wchar_t* pPrinterName;
    [string] wchar t* pServerName;
    DWORD cJobs;
    DWORD cTotalJobs;
    DWORD cTotalBytes;
    SYSTEMTIME stUpTime;
    DWORD MaxcRef;
    DWORD cTotalPagesPrinted;
    DWORD dwGetVersion;
    DWORD fFreeBuild;
    DWORD cSpooling;
    DWORD cMaxSpooling;
    DWORD cRef;
    DWORD cErrorOutOfPaper;
    DWORD cErrorNotReady;
    DWORD cJobError;
    DWORD dwNumberOfProcessors;
    DWORD dwProcessorType;
    DWORD dwHighPartTotalBytes;
    DWORD cChangeID;
    DWORD dwLastError;
    DWORD Status;
    DWORD cEnumerateNetworkPrinters;
    DWORD cAddNetPrinters;
    unsigned short wProcessorArchitecture;
    unsigned short wProcessorLevel;
    DWORD cRefIC;
    DWORD dwReserved2;
    DWORD dwReserved3;
} PRINTER_INFO_STRESS;

typedef struct _PRINTER_INFO_1 {
    DWORD Flags;
    [string] wchar_t* pDescription;
    [string] wchar t* pName;
    [string] wchar_t* pComment;
} PRINTER INFO 1;

typedef struct _PRINTER_INFO_2 {
    [string] wchar_t* pServerName;
    [string] wchar t* pPrinterName;
    [string] wchar_t* pShareName;
    [string] wchar t* pPortName;
    [string] wchar_t* pDriverName;
    [string] wchar_t* pComment;
    [string] wchar_t* pLocation;
    ULONG_PTR pDevMode;
    [string] wchar t* pSepFile;
    [string] wchar_t* pPrintProcessor;
    [string] wchar_t* pDatatype;
```

```
    [string] wchar_t* pParameters;
    ULONG_PTR pSecurityDescriptor;
    DWORD Attributes;
    DWORD Priority;
    DWORD DefaultPriority;
    DWORD StartTime;
    DWORD UntilTime;
    DWORD Status;

    DWORD cJobs;
    DWORD AveragePPM;
} PRINTER_INFO_2;

typedef struct  PRINTER INFO 3 {
    ULONG_PTR pSecurityDescriptor;
} PRINTER INFO 3;

typedef struct _PRINTER_INFO_4 {
    [string] wchar t* pPrinterName;
    [string] wchar_t* pServerName;
    DWORD Attributes;
} PRINTER_INFO_4;

typedef struct _PRINTER_INFO_5 {
    [string] wchar t* pPrinterName;
    [string] wchar_t* pPortName;
    DWORD Attributes;
    DWORD DeviceNotSelectedTimeout;
    DWORD TransmissionRetryTimeout;
} PRINTER_INFO_5;

typedef struct _PRINTER_INFO_6 {
    DWORD dwStatus;
} PRINTER_INFO_6;

typedef struct _PRINTER_INFO_7 {
    [string] wchar t* pszObjectGUID;
    DWORD dwAction;
} PRINTER INFO 7;

typedef struct _PRINTER_INFO_8 {
    ULONG PTR pDevMode;
} PRINTER_INFO_8;

typedef struct _PRINTER_INFO_9 {
    ULONG_PTR pDevMode;
} PRINTER_INFO_9;

typedef struct _SPLCLIENT_INFO_1 {
    DWORD dwSize;
    [string] wchar_t* pMachineName;
    [string] wchar_t* pUserName;
    DWORD dwBuildNum;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    unsigned short wProcessorArchitecture;
} SPLCLIENT_INFO_1;

typedef struct _SPLCLIENT_INFO_2 {
    LONG PTR notUsed;
} SPLCLIENT_INFO_2;

typedef struct _SPLCLIENT_INFO_3 {
    unsigned int cbSize;
    DWORD dwFlags;
    DWORD dwSize;
    [string] wchar t* pMachineName;
    [string] wchar_t* pUserName;
    DWORD dwBuildNum;
```

```
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    unsigned short wProcessorArchitecture;
    unsigned __int64 hSplPrinter;
} SPLCLIENT_INFO_3;

// [MS-RPRN] common info container structures
typedef struct _DEVMODE_CONTAINER {
    DWORD cbBuf;

    [size_is(cbBuf), unique] BYTE* pDevMode;
} DEVMODE_CONTAINER;

typedef struct  DOC INFO CONTAINER {
    DWORD Level;
    [switch is(Level)] union {
      [case(1)]
        DOC_INFO_1* pDocInfo1;
    } DocInfo;
} DOC_INFO_CONTAINER;

typedef struct _DRIVER_CONTAINER {
    DWORD Level;
    [switch_is(Level)] union {
      [case(1)]
        DRIVER_INFO_1* pNotUsed;
      [case(2)]
        DRIVER_INFO_2* Level2;
      [case(3)]
        RPC_DRIVER_INFO_3* Level3;
      [case(4)]
        RPC_DRIVER_INFO_4* Level4;
      [case(6)]
        RPC_DRIVER_INFO_6* Level6;
      [case(8)]
        RPC_DRIVER_INFO_8* Level8;
    } DriverInfo;
} DRIVER_CONTAINER;

typedef struct _FORM_CONTAINER {
    DWORD Level;
    [switch is(Level)] union {
      [case(1)]
        FORM INFO 1* pFormInfo1;
      [case(2)]
        RPC_FORM_INFO_2* pFormInfo2;
    } FormInfo;
} FORM CONTAINER;

typedef struct  JOB CONTAINER {
    DWORD Level;
    [switch_is(Level)] union {
      [case(1)]
        JOB INFO 1* Level1;
      [case(2)]
        JOB INFO 2* Level2;
      [case(3)]
        JOB_INFO_3* Level3;
      [case(4)]
        JOB INFO 4* Level4;
    } JobInfo;
} JOB CONTAINER;

typedef struct _MONITOR_CONTAINER {
    DWORD Level;
    [switch_is(Level)] union {
      [case(1)]
        MONITOR_INFO_1* pMonitorInfo1;
      [case(2)]
```

```
        MONITOR_INFO_2* pMonitorInfo2;
    } MonitorInfo;
} MONITOR_CONTAINER;

typedef struct _PORT_CONTAINER {
    DWORD Level;
    [switch_is(0x00FFFFFF & Level)]
      union {
      [case(1)]
        PORT_INFO_1* pPortInfo1;

      [case(2)]
        PORT_INFO_2* pPortInfo2;
      [case(3)]
        PORT_INFO_3* pPortInfo3;
      [case(0x00FFFFFF)]
        PORT_INFO_FF* pPortInfoFF;
    } PortInfo;
} PORT CONTAINER;

typedef struct  PORT VAR CONTAINER {
    DWORD cbMonitorData;
    [size_is(cbMonitorData), unique, disable_consistency_check] BYTE*
      pMonitorData;
} PORT VAR CONTAINER;

typedef struct  PRINTER CONTAINER {
    DWORD Level;
    [switch_is(Level)] union {
      [case(0)]
        PRINTER INFO STRESS* pPrinterInfoStress;
      [case(1)]
        PRINTER INFO 1* pPrinterInfo1;
      [case(2)]
        PRINTER_INFO_2* pPrinterInfo2;
      [case(3)]
        PRINTER INFO 3* pPrinterInfo3;
      [case(4)]
        PRINTER INFO 4* pPrinterInfo4;
      [case(5)]
        PRINTER_INFO_5* pPrinterInfo5;
      [case(6)]
        PRINTER_INFO_6* pPrinterInfo6;
      [case(7)]
        PRINTER_INFO_7* pPrinterInfo7;
      [case(8)]
        PRINTER_INFO_8* pPrinterInfo8;
      [case(9)]
        PRINTER_INFO_9* pPrinterInfo9;
    } PrinterInfo;
} PRINTER_CONTAINER;

typedef struct _RPC_BINARY_CONTAINER {
    DWORD cbBuf;
    [size_is(cbBuf), unique] BYTE* pszString;
} RPC BINARY CONTAINER;

typedef struct _RPC_BIDI_DATA {
    DWORD dwBidiType;
    [switch is(dwBidiType)] union {
    [case(BIDI_NULL, BIDI_BOOL)]
        int bData;
    [case(BIDI_INT)]
        long iData;
    [case(BIDI_STRING, BIDI_TEXT, BIDI_ENUM)]
        [string,unique] wchar_t* sData;
    [case(BIDI FLOAT)]
        float fData;
    [case(BIDI_BLOB)]
```

```
        RPC_BINARY_CONTAINER biData;
    } u;
} RPC_BIDI_DATA;

typedef struct _RPC_BIDI_REQUEST_DATA {
    DWORD dwReqNumber;
    [string, unique] wchar_t* pSchema;
    RPC_BIDI_DATA data;
} RPC_BIDI_REQUEST_DATA;

typedef struct _RPC_BIDI_RESPONSE_DATA {

    DWORD dwResult;
    DWORD dwReqNumber;
    [string, unique] wchar_t* pSchema;
    RPC BIDI DATA data;
} RPC_BIDI_RESPONSE_DATA;

typedef struct  RPC BIDI REQUEST CONTAINER {
    DWORD Version;
    DWORD Flags;
    DWORD Count;
    [size_is(Count), unique] RPC_BIDI_REQUEST_DATA aData[];
} RPC_BIDI_REQUEST_CONTAINER;

typedef struct _RPC_BIDI_RESPONSE_CONTAINER {
    DWORD Version;
    DWORD Flags;
    DWORD Count;
    [size_is(Count), unique] RPC_BIDI_RESPONSE_DATA aData[];
} RPC BIDI RESPONSE CONTAINER;

typedef struct SECURITY CONTAINER {
    DWORD cbBuf;
    [size_is(cbBuf), unique] BYTE* pSecurity;
} SECURITY_CONTAINER;

typedef struct _SPLCLIENT_CONTAINER {
    DWORD Level;
    [switch_is(Level)] union {
      [case(1)]
        SPLCLIENT INFO 1* pClientInfo1;
      [case(2)]
        SPLCLIENT INFO 2* pNotUsed1;
      [case(3)]
        SPLCLIENT_INFO_3* pNotUsed2;
    } ClientInfo;
} SPLCLIENT CONTAINER;

typedef struct  STRING CONTAINER {
    DWORD cbBuf;
    [size_is(cbBuf/2), unique] WCHAR* pszString;
} STRING_CONTAINER;

typedef struct _SYSTEMTIME_CONTAINER {
    DWORD cbBuf;
    SYSTEMTIME* pSystemTime;
} SYSTEMTIME_CONTAINER;

typedef struct  RPC V2 NOTIFY OPTIONS TYPE {
    unsigned short Type;
    unsigned short Reserved0;
    DWORD Reserved1;
    DWORD Reserved2;
    DWORD Count;
    [size_is(Count), unique] unsigned short* pFields;
} RPC V2 NOTIFY OPTIONS TYPE;

typedef struct _RPC_V2_NOTIFY_OPTIONS {
```

```
    DWORD Version;
    DWORD Reserved;
    DWORD Count;
    [size_is(Count), unique] RPC_V2_NOTIFY_OPTIONS_TYPE* pTypes;
} RPC_V2_NOTIFY_OPTIONS;

typedef
[switch_type (DWORD)]
    union _RPC_V2_NOTIFY_INFO_DATA_DATA {
      [case(TABLE_STRING)]
        STRING CONTAINER String;
      [case(TABLE_DWORD)]

        DWORD dwData[2];
      [case(TABLE_TIME)]
        SYSTEMTIME CONTAINER SystemTime;
      [case(TABLE_DEVMODE)]
        DEVMODE_CONTAINER DevMode;
      [case(TABLE SECURITYDESCRIPTOR)]
        SECURITY_CONTAINER SecurityDescriptor;
} RPC V2 NOTIFY INFO DATA DATA;

typedef struct _RPC_V2_NOTIFY_INFO_DATA {
    unsigned short Type;
    unsigned short Field;
    DWORD Reserved;
    DWORD Id;
    [switch_is(Reserved & 0xffff)]
      RPC_V2_NOTIFY_INFO_DATA_DATA Data;
} RPC_V2_NOTIFY_INFO_DATA;

typedef struct _RPC_V2_NOTIFY_INFO {
    DWORD Version;
    DWORD Flags;
    DWORD Count;
    [size_is(Count), unique] RPC_V2_NOTIFY_INFO_DATA aData[];
} RPC V2 NOTIFY INFO;

typedef [switch type(DWORD)] union  RPC V2 UREPLY PRINTER {
    [case (0)]
      RPC_V2_NOTIFY_INFO* pInfo;
} RPC V2 UREPLY PRINTER;

typedef struct  CORE PRINTER DRIVER {
    GUID         CoreDriverGUID;
    FILETIME     ftDriverDate;
    DWORDLONG    dwlDriverVersion;
    wchar t      szPackageID[260];
} CORE_PRINTER_DRIVER;

typedef struct {
    RPC_EPrintPropertyType ePropertyType;

    [switch is(ePropertyType)]
    union {
        [case(kRpcPropertyTypeString)]
            [string] wchar_t *propertyString;
        [case(kRpcPropertyTypeInt32)]
            LONG              propertyInt32;
        [case(kRpcPropertyTypeInt64)]
            LONGLONG          propertyInt64;
        [case(kRpcPropertyTypeByte)]
            BYTE              propertyByte;
        [case(kRpcPropertyTypeBuffer)]
            struct {
                              DWORD  cbBuf;
                [size is(cbBuf)] BYTE   *pBuf;
            } propertyBlob;
    } value;
```

```
} RPC_PrintPropertyValue;

typedef struct {
    [string] wchar_t               *propertyName;
             RPC_PrintPropertyValue propertyValue;
} RPC PrintNamedProperty;

typedef enum {
    kInvalidJobState = 0,
    kLogJobPrinted,
    kLogJobRendered,
    kLogJobError,
    kLogJobPipelineError,

    kLogOfflineFileFull
} EBranchOfficeJobEventType;

typedef struct {
    DWORD             Status;
    [string] wchar_t* pDocumentName;
    [string] wchar t* pUserName;
    [string] wchar_t* pMachineName;
    [string] wchar_t* pPrinterName;
    [string] wchar_t* pPortName;
    LONGLONG          Size;
    DWORD             TotalPages;
} RPC BranchOfficeJobDataPrinted;

typedef struct {
    LONGLONG    Size;
    DWORD       ICMMethod;
    short       Color;
    short       PrintQuality;
    short       YResolution;
    short       Copies;
    short       TTOption;
} RPC BranchOfficeJobDataRendered;

typedef struct {
    DWORD             LastError;
    [string] wchar_t* pDocumentName;
    [string] wchar t* pUserName;
    [string] wchar_t* pPrinterName;
    [string] wchar t* pDataType;
    LONGLONG          TotalSize;
    LONGLONG          PrintedSize;
    DWORD             TotalPages;
    DWORD             PrintedPages;
    [string] wchar_t* pMachineName;
    [string] wchar t* pJobError;
    [string] wchar_t* pErrorDescription;
} RPC_BranchOfficeJobDataError;

typedef struct {
    [string] wchar_t* pDocumentName;
    [string] wchar t* pPrinterName;
    [string] wchar_t* pExtraErrorInfo;
} RPC_BranchOfficeJobDataPipelineFailed;

typedef struct {
    [string] wchar_t* pMachineName;
} RPC BranchOfficeLogOfflineFileFull;

typedef struct {
    EBranchOfficeJobEventType  eEventType;
    DWORD                      JobId;

    [switch_type(EBranchOfficeJobEventType), switch_is(eEventType)]
    union {
```

```
        [case(kLogJobPrinted)]
             RPC_BranchOfficeJobDataPrinted             LogJobPrinted;
        [case(kLogJobRendered)]
             RPC_BranchOfficeJobDataRendered            LogJobRendered;
        [case(kLogJobError)]
             RPC BranchOfficeJobDataError               LogJobError;
        [case(kLogJobPipelineError)]
             RPC_BranchOfficeJobDataPipelineFailed      LogPipelineFailed;
        [case(kLogOfflineFileFull)]
             RPC_BranchOfficeLogOfflineFileFull         LogOfflineFileFull;
    } JobInfo;
} RPC_BranchOfficeJobData;

typedef struct {

    DWORD cJobDataEntries;
    [size_is(cJobDataEntries), unique] RPC_BranchOfficeJobData JobData[];
} RPC_BranchOfficeJobDataContainer;

// [MS-RPRN] methods
DWORD
RpcEnumPrinters(
   [in] DWORD Flags,
   [in, string, unique]  STRING_HANDLE Name,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
       pPrinterEnum,
   [in] DWORD  cbBuf,
   [out] DWORD*  pcbNeeded,
   [out] DWORD*  pcReturned
);

DWORD
RpcOpenPrinter(
   [in, string, unique] STRING_HANDLE pPrinterName,
   [out] PRINTER_HANDLE* pHandle,
   [in, string, unique] wchar t* pDatatype,
   [in] DEVMODE_CONTAINER* pDevModeContainer,
   [in] DWORD AccessRequired
);

DWORD
RpcSetJob(
    [in] PRINTER HANDLE  hPrinter,
    [in] DWORD JobId,
    [in, unique] JOB_CONTAINER* pJobContainer,
    [in] DWORD Command
);

DWORD
RpcGetJob(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD JobId,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
       pJob,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded
);

DWORD
RpcEnumJobs(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD FirstJob,
   [in] DWORD NoJobs,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
       pJob,
   [in] DWORD cbBuf,
```

```
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAddPrinter(
   [in, string, unique] STRING_HANDLE pName,
   [in] PRINTER_CONTAINER* pPrinterContainer,
   [in] DEVMODE_CONTAINER* pDevModeContainer,
   [in] SECURITY_CONTAINER* pSecurityContainer,
   [out] PRINTER HANDLE* pHandle
);

DWORD
RpcDeletePrinter(

   [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcSetPrinter(
   [in] PRINTER_HANDLE hPrinter,
   [in] PRINTER_CONTAINER* pPrinterContainer,
   [in] DEVMODE_CONTAINER* pDevModeContainer,
   [in] SECURITY CONTAINER* pSecurityContainer,
   [in] DWORD Command
);

DWORD
RpcGetPrinter(
    [in] PRINTER HANDLE hPrinter,
    [in] DWORD Level,
    [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
        pPrinter,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
    );

DWORD
RpcAddPrinterDriver(
   [in, string, unique] STRING_HANDLE pName,
   [in] DRIVER CONTAINER* pDriverContainer
);

DWORD
RpcEnumPrinterDrivers(
   [in, string, unique] STRING_HANDLE pName,
   [in, string, unique] wchar t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
       pDrivers,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcGetPrinterDriver(
   [in] PRINTER_HANDLE hPrinter,
   [in, string, unique] wchar t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
        pDriver,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded
);

DWORD
RpcGetPrinterDriverDirectory(
```

```
    [in, string, unique] STRING_HANDLE  pName,
    [in, string, unique] wchar_t*  pEnvironment,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
         pDriverDirectory,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcDeletePrinterDriver(
   [in, string, unique] STRING_HANDLE pName,
   [in, string] wchar_t* pEnvironment,
   [in, string] wchar_t* pDriverName
);


DWORD
RpcAddPrintProcessor(
   [in, string, unique] STRING_HANDLE pName,
   [in, string] wchar t* pEnvironment,
   [in, string] wchar_t* pPathName,
   [in, string] wchar_t* pPrintProcessorName
);

DWORD
RpcEnumPrintProcessors(
   [in, string, unique] STRING_HANDLE pName,
   [in, string, unique] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
        pPrintProcessorInfo,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcGetPrintProcessorDirectory(
    [in, string, unique] STRING_HANDLE  pName,
    [in, string, unique] wchar_t*  pEnvironment,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
         pPrintProcessorDirectory,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcStartDocPrinter(
   [in] PRINTER_HANDLE hPrinter,
   [in] DOC_INFO_CONTAINER* pDocInfoContainer,
   [out] DWORD* pJobId
);

DWORD
RpcStartPagePrinter(
   [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcWritePrinter(
   [in] PRINTER_HANDLE hPrinter,
   [in, size_is(cbBuf)] BYTE* pBuf,
   [in] DWORD cbBuf,
   [out] DWORD* pcWritten
);

DWORD
```

```
RpcEndPagePrinter(
   [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAbortPrinter(
   [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcReadPrinter(
   [in] PRINTER_HANDLE hPrinter,
   [out, size_is(cbBuf)] BYTE* pBuf,
   [in] DWORD cbBuf,
   [out] DWORD* pcNoBytesRead
);


DWORD
RpcEndDocPrinter(
   [in] PRINTER HANDLE hPrinter
);

DWORD
RpcAddJob(
    [in] PRINTER_HANDLE  hPrinter,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
         pAddJob,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcScheduleJob(
    [in] PRINTER_HANDLE  hPrinter,
    [in] DWORD JobId
);

DWORD
RpcGetPrinterData(
    [in] PRINTER HANDLE hPrinter,
    [in, string] wchar_t* pValueName,
    [out] DWORD* pType,
    [out, size_is(nSize)] BYTE* pData,
    [in] DWORD nSize,
    [out] DWORD* pcbNeeded
);

DWORD
RpcSetPrinterData(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar_t*  pValueName,
    [in] DWORD Type,
    [in, size_is(cbData)] BYTE* pData,
    [in] DWORD cbData
);

DWORD
RpcWaitForPrinterChange(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD Flags,
   [out] DWORD* pFlags
);

DWORD
RpcClosePrinter(
   [in, out] PRINTER_HANDLE*phPrinter
);
```

```
DWORD
RpcAddForm(
    [in] PRINTER_HANDLE hPrinter,
    [in] FORM_CONTAINER* pFormInfoContainer
);

DWORD
RpcDeleteForm(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar t* pFormName
);

DWORD
RpcGetForm(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar t* pFormName,
    [in] DWORD Level,

    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
         pForm,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcSetForm(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar_t* pFormName,
    [in] FORM_CONTAINER* pFormInfoContainer
);

DWORD
RpcEnumForms(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
        pForm,
   [in] DWORD  cbBuf,
   [out] DWORD*  pcbNeeded,
   [out] DWORD*  pcReturned
);

DWORD
RpcEnumPorts(
   [in, string, unique] STRING_HANDLE pName,
   [in] DWORD Level,
   [in, out, unique, size is(cbBuf), disable consistency check] BYTE*
        pPort,
   [in] DWORD  cbBuf,
   [out] DWORD*  pcbNeeded,
   [out] DWORD*  pcReturned
);

DWORD
RpcEnumMonitors(
   [in, string, unique] STRING_HANDLE pName,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
        pMonitor,
   [in] DWORD  cbBuf,
   [out] DWORD*  pcbNeeded,
   [out] DWORD*  pcReturned
);

void
Opnum37NotUsedOnWire();

void
```

```
Opnum38NotUsedOnWire();

DWORD
RpcDeletePort(
   [in, string, unique] STRING_HANDLE pName,
   [in] ULONG PTR hWnd,
   [in, string] wchar_t* pPortName
);

DWORD
RpcCreatePrinterIC(
   [in] PRINTER_HANDLE hPrinter,
   [out] GDI_HANDLE* pHandle,
   [in] DEVMODE_CONTAINER* pDevModeContainer
);

DWORD
RpcPlayGdiScriptOnPrinterIC(
    [in] GDI_HANDLE  hPrinterIC,

    [in, size is(cIn)] BYTE* pIn,
    [in] DWORD cIn,
    [out, size_is(cOut)] BYTE* pOut,
    [in] DWORD cOut,
    [in] DWORD ul
);

DWORD
RpcDeletePrinterIC(
    [in, out] GDI_HANDLE* phPrinterIC
);

void
Opnum43NotUsedOnWire();

void
Opnum44NotUsedOnWire();

void
Opnum45NotUsedOnWire();

DWORD
RpcAddMonitor(
   [in, string, unique] STRING HANDLE Name,
   [in] MONITOR_CONTAINER* pMonitorContainer
);

DWORD
RpcDeleteMonitor(
   [in, string, unique] STRING HANDLE Name,
   [in, string, unique] wchar_t* pEnvironment,
   [in, string] wchar_t* pMonitorName
);

DWORD
RpcDeletePrintProcessor(
   [in, string, unique] STRING_HANDLE Name,
   [in, string, unique] wchar_t* pEnvironment,
   [in, string] wchar_t* pPrintProcessorName
);

void
Opnum49NotUsedOnWire();

void
Opnum50NotUsedOnWire();

DWORD
RpcEnumPrintProcessorDatatypes(
```

```
    [in, string, unique] STRING_HANDLE pName,
    [in, string, unique] wchar_t* pPrintProcessorName,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
        pDatatypes,
    [in] DWORD  cbBuf,
    [out] DWORD*  pcbNeeded,
    [out] DWORD*  pcReturned
);

DWORD
RpcResetPrinter(
    [in] PRINTER_HANDLE hPrinter,
    [in, string, unique] wchar_t* pDatatype,
    [in] DEVMODE_CONTAINER* pDevModeContainer
);

DWORD
RpcGetPrinterDriver2(
    [in] PRINTER_HANDLE hPrinter,

    [in, string, unique] wchar_t* pEnvironment,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
        pDriver,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded,
    [in] DWORD dwClientMajorVersion,
    [in] DWORD dwClientMinorVersion,
    [out] DWORD* pdwServerMaxVersion,
    [out] DWORD* pdwServerMinVersion
);

void
Opnum54NotUsedOnWire();

void
Opnum55NotUsedOnWire();

DWORD
RpcFindClosePrinterChangeNotification(
    [in] PRINTER HANDLE hPrinter
);

void
Opnum57NotUsedOnWire();

DWORD
RpcReplyOpenPrinter(
    [in, string] STRING HANDLE pMachine,
    [out] PRINTER_HANDLE*phPrinterNotify,
    [in] DWORD dwPrinterRemote,
    [in] DWORD dwType,
    [in, range(0, 512)] DWORD cbBuffer,
    [in, unique, size_is(cbBuffer), disable_consistency_check] BYTE*
        pBuffer
);

DWORD
RpcRouterReplyPrinter(
    [in] PRINTER_HANDLE hNotify,
    [in] DWORD fdwFlags,
    [in, range(0, 512)] DWORD cbBuffer,
    [in, unique, size_is(cbBuffer), disable_consistency_check] BYTE*
        pBuffer
);

DWORD
RpcReplyClosePrinter(
```

```
   [in, out] PRINTER_HANDLE*phNotify
);

DWORD
RpcAddPortEx(
    [in, string, unique] STRING HANDLE pName,
    [in] PORT_CONTAINER* pPortContainer,
    [in] PORT_VAR_CONTAINER* pPortVarContainer,
    [in, string] wchar_t* pMonitorName
);

DWORD
RpcRemoteFindFirstPrinterChangeNotification(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD fdwFlags,
   [in] DWORD fdwOptions,
   [in, string, unique] wchar t* pszLocalMachine,
    [in] DWORD dwPrinterLocal,
   [in, range(0, 512)] DWORD cbBuffer,
   [in, out, unique, size_is(cbBuffer), disable_consistency_check]
         BYTE* pBuffer

);

void
Opnum63NotUsedOnWire();

void
Opnum64NotUsedOnWire();

DWORD
RpcRemoteFindFirstPrinterChangeNotificationEx(
   [in] PRINTER HANDLE hPrinter,
   [in] DWORD fdwFlags,
   [in] DWORD fdwOptions,
   [in, string, unique] wchar_t* pszLocalMachine,
   [in] DWORD dwPrinterLocal,
   [in, unique] RPC_V2_NOTIFY_OPTIONS* pOptions
);

DWORD
RpcRouterReplyPrinterEx(
   [in] PRINTER_HANDLE hNotify,
   [in] DWORD dwColor,
   [in] DWORD fdwFlags,
   [out] DWORD* pdwResult,
   [in] DWORD dwReplyType,
   [in, switch is(dwReplyType)] RPC V2 UREPLY PRINTER Reply
);

DWORD
RpcRouterRefreshPrinterChangeNotification(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD dwColor,
   [in, unique] RPC_V2_NOTIFY_OPTIONS* pOptions,
   [out] RPC V2 NOTIFY INFO** ppInfo
);

void
Opnum68NotUsedOnWire();

DWORD
RpcOpenPrinterEx(
   [in, string, unique] STRING_HANDLE pPrinterName,
   [out] PRINTER_HANDLE* pHandle,
   [in, string, unique] wchar_t* pDatatype,
   [in] DEVMODE CONTAINER* pDevModeContainer,
   [in] DWORD AccessRequired,
   [in] SPLCLIENT_CONTAINER* pClientInfo
```

```
);

DWORD
RpcAddPrinterEx(
   [in, string, unique] STRING_HANDLE pName,
   [in] PRINTER_CONTAINER* pPrinterContainer,
   [in] DEVMODE_CONTAINER* pDevModeContainer,
   [in] SECURITY_CONTAINER* pSecurityContainer,
   [in] SPLCLIENT_CONTAINER* pClientInfo,
   [out] PRINTER_HANDLE* pHandle
);

DWORD
RpcSetPort(
    [in, string, unique] STRING_HANDLE pName,
    [in, string, unique] wchar_t* pPortName,
    [in] PORT_CONTAINER* pPortContainer
);

DWORD
RpcEnumPrinterData(
    [in] PRINTER_HANDLE hPrinter,

    [in] DWORD dwIndex,
    [out, size_is(cbValueName/sizeof(wchar_t))] wchar_t* pValueName,
    [in]  DWORD cbValueName,
    [out] DWORD* pcbValueName,
    [out] DWORD* pType,
    [out, size_is(cbData)] BYTE* pData,
    [in]  DWORD cbData,
    [out] DWORD* pcbData
);

DWORD
RpcDeletePrinterData(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar_t* pValueName
);

void
Opnum74NotUsedOnWire();

void
Opnum75NotUsedOnWire();

void
Opnum76NotUsedOnWire();

DWORD
RpcSetPrinterDataEx(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName,
    [in, string] const wchar_t* pValueName,
    [in] DWORD Type,
    [in, size_is(cbData)] BYTE* pData,
    [in] DWORD cbData
);

DWORD
RpcGetPrinterDataEx(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName,
    [in, string] const wchar_t* pValueName,
    [out] DWORD* pType,
    [out, size_is(nSize)] BYTE* pData,
    [in] DWORD nSize,
    [out] DWORD* pcbNeeded
);
```

```
DWORD
RpcEnumPrinterDataEx(
    [in] PRINTER_HANDLE hPrinter,
    [in, string]  const wchar_t* pKeyName,
    [out, size_is(cbEnumValues)] BYTE* pEnumValues,
    [in]  DWORD cbEnumValues,
    [out] DWORD* pcbEnumValues,
    [out] DWORD* pnEnumValues
);

DWORD
RpcEnumPrinterKey(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName,
    [out, size_is(cbSubkey/sizeof(wchar_t))] wchar_t* pSubkey,
    [in]  DWORD cbSubkey,
    [out] DWORD* pcbSubkey
);

DWORD
RpcDeletePrinterDataEx(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName,

    [in, string] const wchar t* pValueName
);

DWORD
RpcDeletePrinterKey(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar t* pKeyName
);

void
Opnum83NotUsedOnWire();

DWORD
RpcDeletePrinterDriverEx(
    [in, string, unique] STRING HANDLE pName,
    [in, string] wchar_t* pEnvironment,
    [in, string] wchar_t* pDriverName,
    [in] DWORD dwDeleteFlag,
    [in] DWORD dwVersionNum
);

DWORD
RpcAddPerMachineConnection(
   [in, string, unique] STRING HANDLE pServer,
   [in, string] const wchar_t*  pPrinterName,
   [in, string] const wchar t*  pPrintServer,
   [in, string] const wchar_t*  pProvider
);

DWORD
RpcDeletePerMachineConnection(
   [in, string, unique] STRING HANDLE pServer,
   [in, string] const wchar_t*  pPrinterName
);

DWORD
RpcEnumPerMachineConnections(
   [in, string, unique]  STRING HANDLE pServer,
   [in, out, unique, size_is(cbBuf), disable_consistency_check] BYTE*
        pPrinterEnum,
   [in]  DWORD  cbBuf,
   [out] DWORD*  pcbNeeded,
   [out] DWORD*  pcReturned
);
```

```
DWORD
RpcXcvData(
    [in] PRINTER_HANDLE hXcv,
    [in, string]  const wchar_t* pszDataName,
    [in, size_is(cbInputData)] BYTE* pInputData,
    [in] DWORD cbInputData,
    [out, size_is(cbOutputData)] BYTE* pOutputData,
    [in] DWORD cbOutputData,
    [out] DWORD* pcbOutputNeeded,
    [in, out] DWORD* pdwStatus
);

DWORD
RpcAddPrinterDriverEx(
   [in, string, unique] STRING_HANDLE pName,
   [in] DRIVER_CONTAINER* pDriverContainer,
   [in] DWORD dwFileCopyFlags
);

void
Opnum90NotUsedOnWire();

void
Opnum91NotUsedOnWire();


void
Opnum92NotUsedOnWire();

void
Opnum93NotUsedOnWire();

void
Opnum94NotUsedOnWire();

void
Opnum95NotUsedOnWire();

DWORD
RpcFlushPrinter(
   [in] PRINTER_HANDLE hPrinter,
   [in, size is(cbBuf)] BYTE* pBuf,
   [in] DWORD cbBuf,
   [out] DWORD* pcWritten,
   [in] DWORD cSleep
);

DWORD RpcSendRecvBidiData(
    [in] PRINTER_HANDLE hPrinter,
    [in, string, unique] const wchar t* pAction,
    [in] RPC_BIDI_REQUEST_CONTAINER* pReqData,
    [out] RPC_BIDI_RESPONSE_CONTAINER** ppRespData);

void
Opnum98NotUsedOnWire();

void
Opnum99NotUsedOnWire();

void
Opnum100NotUsedOnWire();

void
Opnum101NotUsedOnWire();

HRESULT RpcGetCorePrinterDrivers(
    [in, string, unique]               STRING HANDLE         pszServer,
    [in, string]                 const wchar_t *             pszEnvironment,
    [in]                               DWORD                 cchCoreDrivers,
```

```
    [in, size_is(cchCoreDrivers)] const wchar_t *             pszzCoreDriverDependencies,
    [in]                          DWORD                       cCorePrinterDrivers,
    [out, size_is(cCorePrinterDrivers)] CORE_PRINTER_DRIVER * pCorePrinterDrivers);

void
Opnum103NotUsedOnWire();

HRESULT RpcGetPrinterDriverPackagePath(
    [in, string, unique]       STRING_HANDLE  pszServer,
    [in, string]         const wchar_t *      pszEnvironment,
    [in, string, unique] const wchar_t *      pszLanguage,
    [in, string]         const wchar_t *      pszPackageID,
    [in, out, unique, size_is(cchDriverPackageCab)]
                               wchar_t *      pszDriverPackageCab,
    [in]                       DWORD          cchDriverPackageCab,
    [out]                      LPDWORD        pcchRequiredSize);

void
Opnum105NotUsedOnWire();

void
Opnum106NotUsedOnWire();

void
Opnum107NotUsedOnWire();


void
Opnum108NotUsedOnWire();

void
Opnum109NotUsedOnWire();

DWORD RpcGetJobNamedPropertyValue(
    [in]  PRINTER_HANDLE         hPrinter,
    [in]  DWORD                  JobId,
    [in, string] const wchar_t   *pszName,
    [out] RPC_PrintPropertyValue *pValue);

DWORD RpcSetJobNamedProperty(
    [in] PRINTER_HANDLE          hPrinter,
    [in] DWORD                   JobId,
    [in] RPC_PrintNamedProperty  *pProperty);

DWORD RpcDeleteJobNamedProperty(
    [in]  PRINTER_HANDLE         hPrinter,
    [in]  DWORD                  JobId,
    [in, string] const wchar_t   *pszName);

DWORD RpcEnumJobNamedProperties(
    [in]                           PRINTER_HANDLE         hPrinter,
    [in]                           DWORD                  JobId,
    [out]                          DWORD                  *pcProperties,
    [out, size_is(,*pcProperties)] RPC_PrintNamedProperty **ppProperties);

void
Opnum114NotUsedOnWire();}

void
Opnum115NotUsedOnWire();

DWORD
RpcLogJobInfoForBranchOffice(
    [in]      PRINTER_HANDLE                   hPrinter,
    [in, ref] RPC_BranchOfficeJobDataContainer *pBranchOfficeJobDataContainer);
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

Note: Some of the information in this section is subject to change because it applies to a preliminary product version, and thus may differ from the final version of the software when released. All behavior notes that pertain to the preliminary product version contain specific references to it as an aid to the reader.

- Windows NT 3.1 operating system
- Windows NT 3.5 operating system
- Windows NT 3.51 operating system
- Windows NT 4.0 operating system
- Microsoft Windows 98 operating system
- Windows 2000 operating system
- Windows Millennium Edition operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 Technical Preview operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.4: In Windows NT 3.51 and Windows NT 4.0, print clients and servers do not interact with Active directory. Otherwise, print servers can publish printers to the Active Directory, and print clients can search the Active Directory for printers.

<2> Section 1.7: The dwBuildNumber value for OSVERSIONINFO and OSVERSIONINFOEX (section 2.2.3.10.2) for specific versions of Windows is shown in the table that follows.

Version	dwBuildNumber value
Windows 8.1 and Windows Server 2012 R2	>= 9431
Windows 8 and Windows Server 2012	>= 9200
Windows 7 and Windows Server 2008 R2	>= 7007
Windows Vista operating system with Service Pack 1 (SP1) and Windows Server 2008	>= 6001
Windows Vista and Windows Server 2008	>= 6000
Windows XP operating system Service Pack 1 (SP1)	>= 2196
Windows XP and Windows Server 2003	>= 2196
Windows 2000	>= 1382
Windows NT 4.0	>= 1381

<3> Section 2.1: The Windows server impersonates the client when processing a method, and it registers **security providers** as follows:

- Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, and Windows 2000 servers do not register a security provider.
- Windows Server 2003 server registers the NT LAN Manager (NTLM) security provider.
- Otherwise, servers register the **Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)** security provider.

<4> Section 2.2.1.2.7: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<5> Section 2.2.1.2.8: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<6> Section 2.2.1.2.10: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<7> Section 2.2.1.2.11: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<8> Section 2.2.1.2.12: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<9> Section 2.2.1.2.14: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<10> Section 2.2.1.2.15: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista

- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<11> Section 2.2.1.2.16: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<12> Section 2.2.1.2.17: The RPC_BranchOfficeJobDataContainer (section 2.2.1.2.17) structure is supported on the following Windows versions:

- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<13> Section 2.2.1.3.1: The Windows operating system uses the following values to indicate printer drivers on different OS versions:

Value	Meaning
0x00000000	The printer driver for Windows 95 operating system, Windows 98, and Windows Millennium Edition.
0x00000001	The printer driver for Windows NT 3.51.
0x00000002	Kernel-mode printer driver for Windows NT 4.0.
0x00000003	User-mode printer driver for Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.
0x00000004	User-mode printer driver for Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<14> Section 2.2.1.3.1: In Windows, a language monitor is specified for printers capable of bidirectional communication. The name is specific to a printer manufacturer. For example, the name of a language monitor could be "PJL monitor".

<15> Section 2.2.1.3.1: Windows print servers use this ordering for **pDependentFiles** members of returned DRIVER_INFO (section 2.2.1.5) and DRIVER_INFO and RPC_DRIVER_INFO Members (section 2.2.1.3.1) structures on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<16> Section 2.2.1.3.1: Windows operating systems use a combination of the OS major and minor numbers, the build number, and the revision. For example, the printer driver version number 0x000500020ECE0726 represents:

- OS Major Version: 0x0005
- OS Minor Version: 0x0002
- Build number: 0x0ECE (3790)
- Revision: 0x0726 (1830)

Windows clients use this value to check the versions of server printer driver, and when a version does not match, the user is prompted to update the driver.

<17> Section 2.2.1.3.6: In Windows, if non-NULL, the string length must be less than or equal to 1,041 characters.

<18> Section 2.2.1.3.6: In Windows, if non-NULL, the string length must be less than or equal to 256 characters.

<19> Section 2.2.1.3.7: The Windows implementation uses the following major version values.

Value	Meaning
0x00000004	The operating system is Windows 95, Windows NT 4.0, Windows 98, or Windows Millennium Edition.
0x00000005	The operating system is Windows 2000, Windows XP, or Windows Server 2003.
0x00000006	The operating system is Windows Vista, Windows Server 2008, Windows Server 2008 R2, Windows 7, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.

<20> Section 2.2.1.3.7: The Windows implementation uses the following minor version values.

Value	Meaning
0x00000000	The operating system is Windows 95, Windows NT 4.0, Windows 2000, Windows Vista, Windows Server 2008, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.
0x00000001	The operating system is Windows XP, Windows 7, Windows Server 2008, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.
0x00000002	The operating system is Windows XP Professional x64 Edition operating system or Windows Server 2003.
0x0000000A	The operating system is Windows 98.
0x0000005A	The operating system is Windows Millennium Edition.

<21> Section 2.2.1.3.7: The Windows implementation uses the following processor architecture values:

Name/Value	Meaning
PROCESSOR_ARCHITECTURE_INTEL 0x0000	x86 architecture
PROCESSOR_ARCHITECTURE_IA64 0x0006	Itanium architecture
PROCESSOR_ARCHITECTURE_AMD64 0x0009	AMD64 architecture
PROCESSOR_ARCHITECTURE_ARM 0x0005	ARM architecture

<22> Section 2.2.1.4: In Windows the default job name is "No Document Name".

<23> Section 2.2.1.5.3: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<24> Section 2.2.1.5.4: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<25> Section 2.2.1.5.5: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<26> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<27> Section 2.2.1.5.6: Windows uses INF files to define the printer driver configuration.

<28> Section 2.2.1.5.6: This flag was introduced with Windows Vista.

<29> Section 2.2.1.5.6: Windows print servers determine that a printer driver supports the Microsoft XML Paper Specification (XPS) format described in [MSFT-XMLPAPER] if and only if the list of dependent files associated with that printer driver contains the **file** "PipelineConfig.xml".

This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<30> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<31> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<32> Section 2.2.1.5.6: This flag is supported on Windows 7 and Windows Server 2008 R2.

<33> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<34> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<35> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<36> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<37> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<38> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<39> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<40> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<41> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<42> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<43> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<44> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<45> Section 2.2.1.5.6: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<46> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<47> Section 2.2.1.5.6: When the print queue is associated with a printer port corresponding to a USB printer, Windows print servers use the USB soft reset mechanism described in [USBPRINT] section 4.2.3. This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<48> Section 2.2.1.5.6: This flag is supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<49> Section 2.2.1.5.6: 3D printers are supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<50> Section 2.2.1.5.6: This flag is supported on Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<51> Section 2.2.1.5.6: The concepts of core printer driver and core driver dependencies are Windows–specific implementation details.

<52> Section 2.2.1.5.6: In Windows, this member applies to only package-aware printer driver.

<53> Section 2.2.1.5.6: Windows reads this value from the printer driver INF file.

<54> Section 2.2.1.6.2: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<55> Section 2.2.1.6.2: Printer drivers generate an implementation-specific unique identifier. Windows print servers generate a unique GUID.

<56> Section 2.2.1.6.2: Windows provides Multilingual User Interface (MUI) DLLs that contain localized string resources for inbox printer drivers. For third-party printer drivers, whether to localize strings is an implementation decision.

<57> Section 2.2.1.7.3: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<58> Section 2.2.1.7.4: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<59> Section 2.2.1.9.2: Windows specifies a descriptive name for the port monitor. For example: "Standard TCP/IP Port", "Fax Monitor Port", or "Local Port".

<60> Section 2.2.1.9.3: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<61> Section 2.2.1.9.4: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<62> Section 2.2.1.10.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008

- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<63> Section 2.2.1.10.1: Windows calculates the version by storing the build version in the high-order 16 bits, and the operation system release number in the low-order 16 bits. For example, 0x0A280005 corresponds to XP build 2600.

<64> Section 2.2.1.10.1: The Windows debugging build of the print server sets **fFreeBuild** to 0, and the release build of the print server sets **fFreeBuild** to 1.

<65> Section 2.2.1.10.1: Windows uses the following values:

Value	Meaning
PROCESSOR_INTEL_386 0x00000182	Intel 80386 compatible
PROCESSOR_INTEL_486 0x000001E6	Intel 80486 compatible
PROCESSOR_INTEL_PENTIUM 0x0000024A	Intel Pentium compatible
PROCESSOR_INTEL_IA64 0x00000898	Intel Itanium-based compatible
PROCESSOR_AMD_X8664 0x000022A0	AMD x64 compatible
PROCESSOR_ARM 0x00000000	ARM compatible

<66> Section 2.2.1.10.1: Windows returns a nonzero Win32 error code to indicate failure ([MS-ERREF] section 2.2).

<67> Section 2.2.1.10.1: Windows uses the following values:

Value	Meaning
PROCESSOR_ARCHITECTURE_INTEL 0x0000	x86 architecture
PROCESSOR_ARCHITECTURE_IA64	Itanium architecture

Value	Meaning
0x0006	
PROCESSOR_ARCHITECTURE_AMD64 0x0009	AMD64 architecture
PROCESSOR_ARCHITECTURE_ARM 0x0005	ARM architecture

<68> Section 2.2.1.10.1: Windows uses the value of 1 for **PROCESSOR_ARCHITECTURE_IA64** and **PROCESSOR_ARCHITECTURE_AMD64**.

For **PROCESSOR_ARCHITECTURE_INTEL** and **PROCESSOR_ARCHITECTURE_ARM**, Windows uses the value defined by the CPU vendor.

<69> Section 2.2.1.10.3: Windows servers select the first data type in the list of data types obtained from the print processor. This list is obtained by the same mechanism as in RpcEnumPrintProcessorDatatypes (section 3.1.4.8.5).

<70> Section 2.2.1.10.5: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<71> Section 2.2.1.10.6: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<72> Section 2.2.1.10.7: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<73> Section 2.2.1.10.8: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<74> Section 2.2.1.10.9: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<75> Section 2.2.1.10.10: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<76> Section 2.2.1.11.1: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<77> Section 2.2.1.11.3: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<78> Section 2.2.1.12.1: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<79> Section 2.2.1.12.1: The bidirectional communications schema is a hierarchy of printer attributes, some of which are properties, with the rest being values or value entries. Bidirectional communications interfaces are implemented by printer-specific components. A detailed description of printer drivers and the bidirectional communications schema can be found in the Windows Device Driver Kit. See [MSDN-MPD] and [MSDN-BIDI] for further information.

<80> Section 2.2.1.12.2: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<81> Section 2.2.1.12.2: Windows returns a nonzero error code to indicate failure, as specified in [MS-ERREF].

<82> Section 2.2.1.12.2: The bidirectional communications schema is a hierarchy of printer attributes, some of which are properties, with the rest being values or value entries. Bidirectional communications interfaces are implemented by printer-specific components. A detailed description of printer drivers and the bidirectional communications schema can be found in the Windows Device Driver Kit.

<83> Section 2.2.1.12.3: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<84> Section 2.2.1.13.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<85> Section 2.2.1.13.2: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008

- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<86> Section 2.2.1.13.2: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<87> Section 2.2.1.13.3: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<88> Section 2.2.1.13.4: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<89> Section 2.2.1.13.5: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<90> Section 2.2.1.13.6: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<91> Section 2.2.1.14.1: The **RPC_PrintPropertyValue** structure is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<92> Section 2.2.1.14.2: The **RPC_PrintNamedProperty** structure is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<93> Section 2.2.1.14.3: The **RPC_EPrintPropertyType** enumeration is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<94> Section 2.2.1.14.4: The **Job Named Property** with the name defined by the SPL_FILE_CONTENT_TYPE_PROP_NAME constant is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<95> Section 2.2.1.15.1: The EBranchOfficeJobEventType (section 2.2.1.15.1) enumeration is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<96> Section 2.2.1.15.2: The RPC_BranchOfficeJobData (section 2.2.1.15.2) structure is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<97> Section 2.2.1.15.3: The RPC_BranchOfficeJobDataError (section 2.2.1.15.3) structure is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<98> Section 2.2.1.15.4: The RPC_BranchOfficeJobDataPipelineFailed (section 2.2.1.15.4) structure is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<99> Section 2.2.1.15.5: The RPC_BranchOfficeJobDataPrinted (section 2.2.1.15.5) structure is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<100> Section 2.2.1.15.6: The RPC_BranchOfficeJobDataRendered (section 2.2.1.15.6) structure is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<101> Section 2.2.1.15.7: The RPC_BranchOfficeLogOfflineFileFull (section 2.2.1.15.7) structure is supported on the following Windows versions:

- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<102> Section 2.2.1.15.7: The maximum allowed size is determined by the implementation. On Windows, the default value is 10 MB.

<103> Section 2.2.2: The 64-bit Edition of the following Windows versions do not correctly handle custom-marshaled **INFO** structures where the unused space is not between the end of the last **Fixed_Portion** block and the beginning of the first **Variable_Data** field.

- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<104> Section 2.2.2.1: Versions of initialization data specifications correspond to versions of Windows operating systems as follows.

Value	Meaning
0x0320	Windows NT 3.1, Windows NT 3.5, and Windows NT 3.51.
0x0400	Windows 95, Windows 98, and Windows Millennium Edition.
0x0401	Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<105> Section 2.2.2.1: Versions of printer drivers correspond to versions of Windows operating systems as follows.

Value	Meaning
0x0301 — 0x03FF	Windows NT 3.1, Windows NT 3.5, and Windows NT 3.51 user-mode printer drivers, and Windows NT 4.0 kernel-mode printer drivers.
0x0500 — 0x05FF	Windows 2000, Windows XP, and Windows Server 2003 user-mode printer drivers.

Value	Meaning
0x0600 — 0x06FF	Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview printer drivers.

<106> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<107> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<108> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<109> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<110> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<111> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<112> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<113> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<114> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<115> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<116> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008,

Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<117> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<118> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<119> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<120> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<121> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<122> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<123> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<124> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<125> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<126> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<127> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<128> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008,

Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<129> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<130> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<131> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<132> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<133> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<134> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<135> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<136> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<137> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<138> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<139> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<140> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008,

Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<141> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<142> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<143> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<144> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<145> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<146> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<147> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<148> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<149> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<150> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<151> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<152> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008,

Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<153> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<154> Section 2.2.2.1: Supported on Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<155> Section 2.2.2.1: Supported on Windows 95, Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

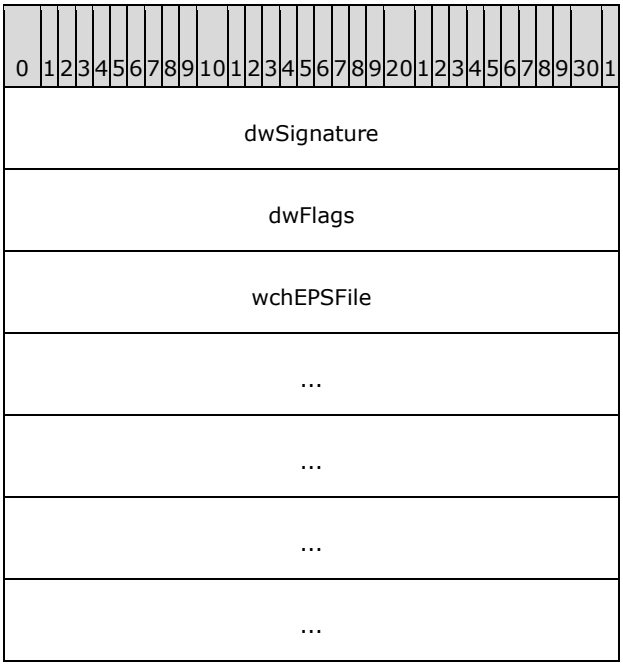
<156> Section 2.2.2.1: The value of this field is set by printer manufacturers, depending on the requirements of the printer driver.

<157> Section 2.2.2.1.1: **PSCRIPT** is the Windows PostScript core printer driver. It stores the private data that is required in the _DEVMODE **dmDriverExtraData** field with the following structures.

The **dmDriverExtraData** field contains one of the **PSDRVEXTRA** structures, immediately followed by zero or one JTEXP (section 2.2.2.1.4) structures, followed by zero or more OEM_DMEXTRA (section 2.2.2.1.3) structures.

These structures are not part of the protocol defined in this specification, and they are subject to change without notice. Implementations of compatible drivers must check the structure version and discard any data they do not handle.

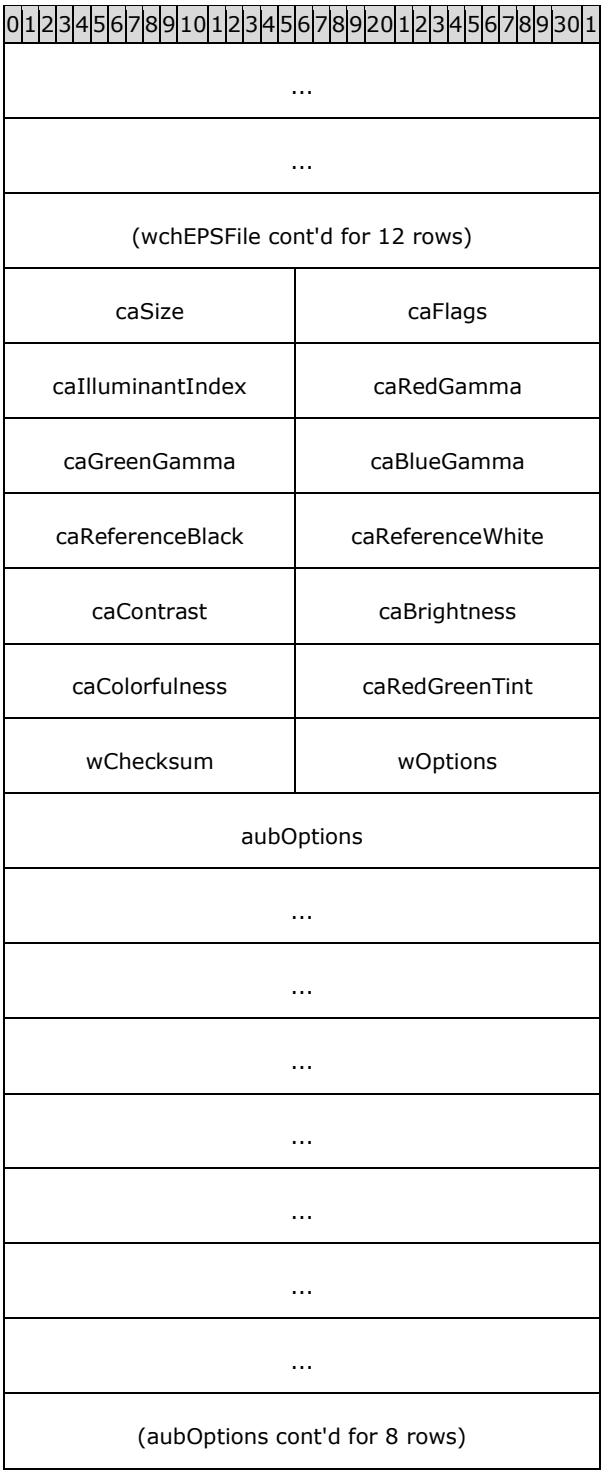
PSDRVEXTRA351: Defined by the **PSCRIPT** driver released with Windows NT 3.51. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** of the _DEVMODE structure is 0x0350.



0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1	
...	
...	
...	
(wchEPSFile cont'd for 12 rows)	
caSize	caFlags
caIlluminantIndex	caRedGamma
caGreenGamma	caBlueGamma
caReferenceBlack	caReferenceWhite
caContrast	caBrightness
caColorfulness	caRedGreenTint

PSDRVEXTRA400: Defined by the **PSCRIPT** driver released with Windows NT 4.0. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** of the _DEVMODE structure is 0x0400.

0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
dwSignature
dwFlags
wchEPSFile
...
...
...
...
...



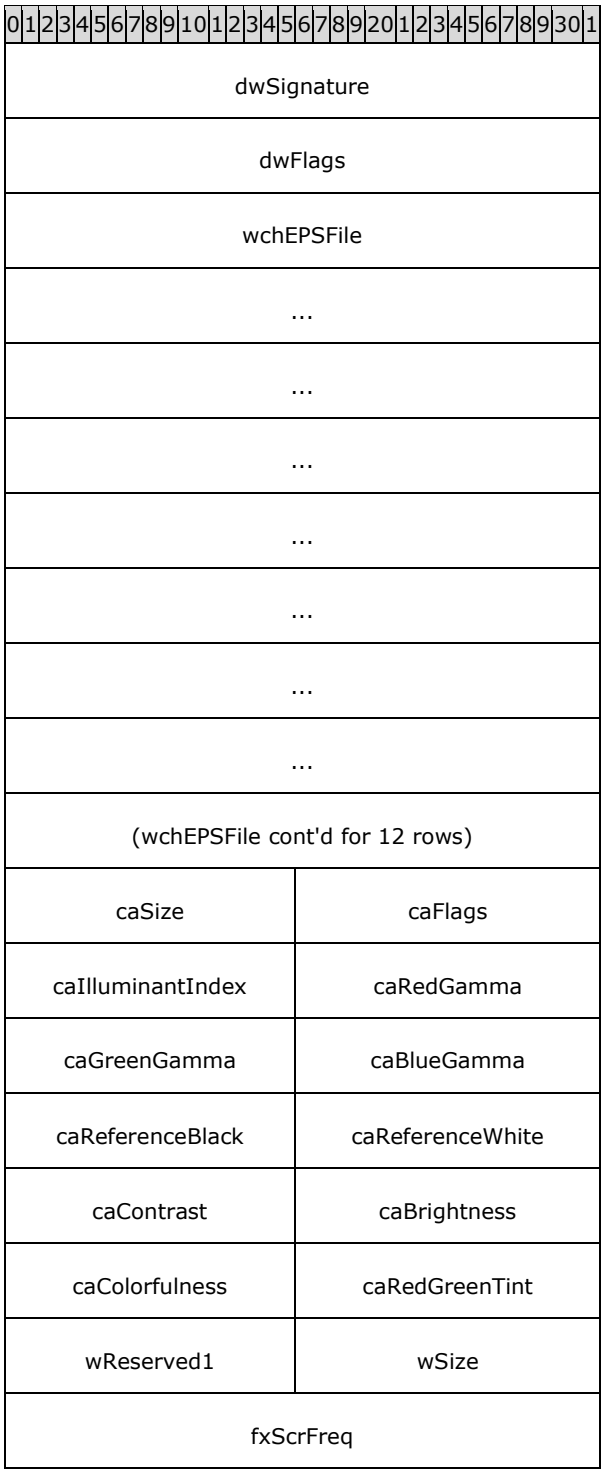
wChecksum: The value of this field is a checksum of the **aubOptions** array.

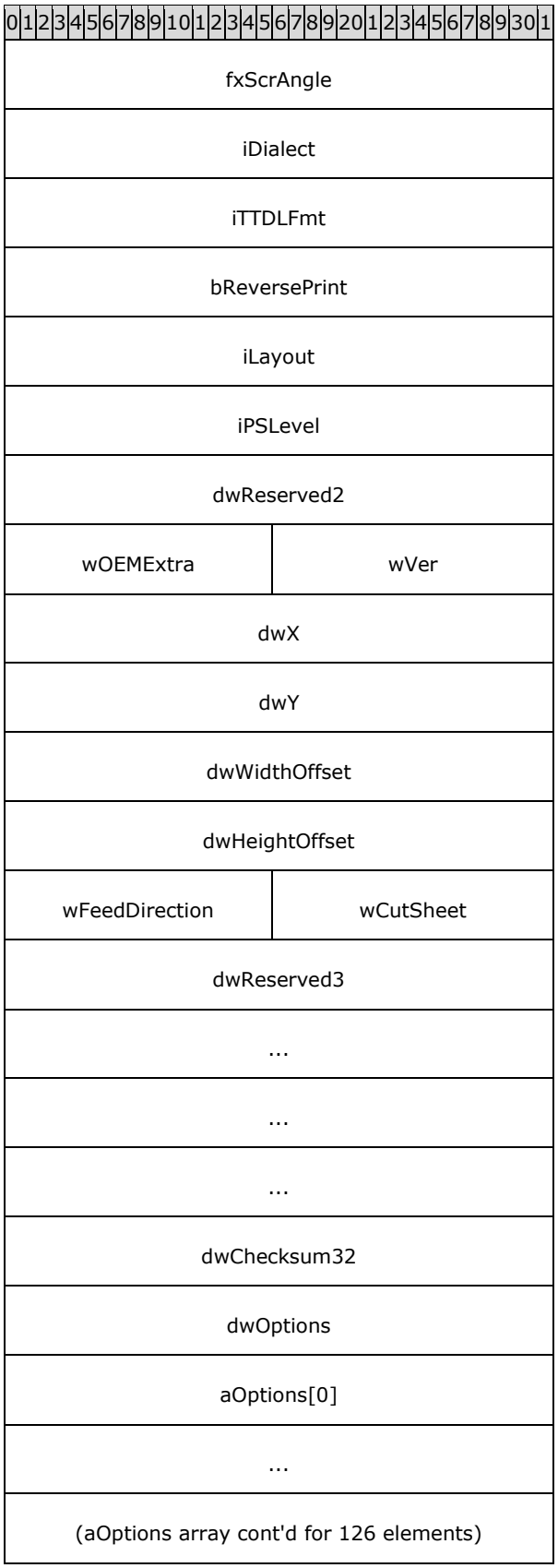
wOptions: The value of this field is the number of entries in the **aubOptions** array that are initialized.

aubOptions: This field is an array of 64 bytes in length and contains user interface selections. Unused fields should be initialized to zero. The meaning of the entries in this array differs for each

supported printer model. Upon receipt, the checksum of this array is computed and compared to **wChecksum**. The array is used only if the checksums match.

PSDRVEXTRA500: Defined by the **PSCRIPT** driver released with Windows 2000 and Windows XP. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** of the _DEVMODE structure is 0x0501.



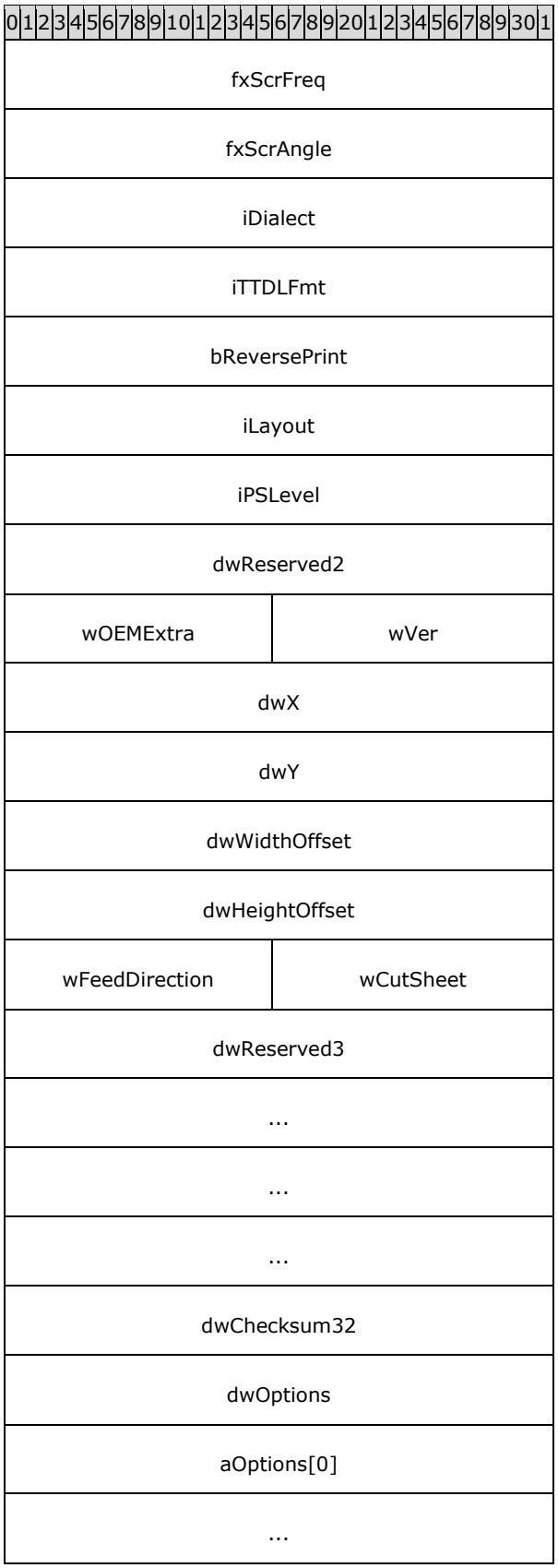


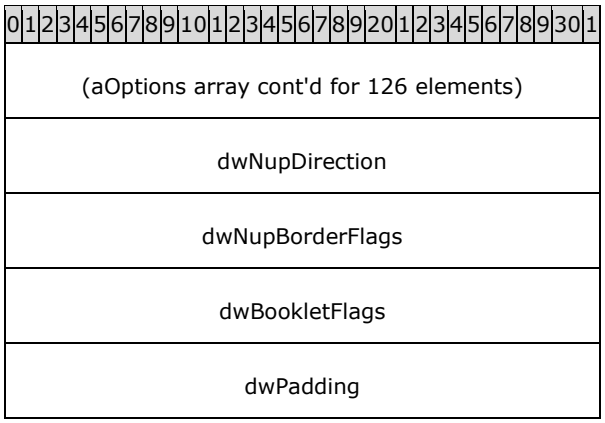
wReserved1: The value of this field should be set to zero when sent and must be ignored on receipt.

wSize: This field is the same as **wCoreFullSize** in **PSDRVEXTRA**.

PSDRVEXTRA: Defined by the **PSCRIPT** driver released with Windows Vista, Windows Server 2008, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** of the _DEVMODE structure is 0x0600.

0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1	
dwSignature	
dwFlags	
wchEPSFile	
...	
...	
...	
...	
...	
...	
...	
(wchEPSFile cont'd for 12 rows)	
caSize	caFlags
caIlluminantIndex	caRedGamma
caGreenGamma	caBlueGamma
caReferenceBlack	caReferenceWhite
caContrast	caBrightness
caColorfulness	caRedGreenTint
wCoreJTExpSize	wCoreFullSize





wCoreJTExpSize: The value of this field specifies the size of the JTEXP structure if one is present, following directly after this structure.

wCoreFullSize: The value of this field specifies the size of the **PSDRVEXTRA** structure plus the value of **wCoreJTExpSize**.

dwNupDirection: This field is used only if N-Up printing is selected, and the value of this field must be one of the following.

Value	Meaning
0x00000001	Print N-Up pages left-to-right, top-to-bottom.
0x00000002	Print N-Up pages top-to-bottom, left-to-right.
0x00000004	Print N-Up pages right-to-left, top-to-bottom.
0x00000008	Print N-Up pages top-to-bottom, right-to-left.

dwNupBorderFlags: This field is used only if N-Up printing is selected, and the value of this field must be one of the following.

Value	Meaning
0x00000000	Print borders around N-Up pages.
0x00000001	Do not print borders around N-Up pages.

dwBookletFlags: This field is used only if booklet printing is selected, and the value of this field must be one of the following.

Value	Meaning
0x00000000	Print booklet so that pages flip to the left (western style).
0x00000001	Print booklet so that pages flip to the right.

dwPadding: The value of this field should be set to zero when sent and must be ignored on receipt.

Description of common members for **PSDRVEXTRA** structures:

dwSignature: The value of this field must be 0x56495250.

dwFlags: The value of this field must be the result of the bitwise OR of zero or more of the following values.

Value	Meaning
0x00000002	Send PostScript driver error handler code.
0x00000004	Print mirror image.
0x00000010	Print negative image of page.
0x00000040	Compress bitmaps.
0x00000200	If _DEVMODE field **dmOrientation** is set to 2 (LANDSCAPE), rotate page by an additional 180 degrees.
0x00002000	If driver is initialized by GDI, inform GDI that metafile spooling is requested.

wchEPSFile: This field is not used.

caSize: The value of this field must be 24.

caFlags: This field is not used.

caIlluminantIndex: This field is not used.

caRedGamma: This field is not used.

caGreenGamma: This field is not used.

caBlueGamma: This field is not used.

caReferenceBlack: This field is not used.

caReferenceWhite: This field is not used.

caContrast: This field is not used.

caBrightness: This field is not used.

caColorfulness: This field is not used.

caRedGreenTint: This field is not used.

fxScrFreq: This field is not used.

fxScrAngle: This field is not used.

iDialect: The value of this field must be one of the following.

Value	Meaning
0x00000000	Optimize generated PostScript for speed.
0x00000001	Optimize generated PostScript for portability.
0x00000002	Optimize generated PostScript for EPS use.
0x00000003	Optimize generated PostScript for archival.

iTTDLFmt: The value of this field must be one of the following.

Value	Meaning
0x00000000	Download fonts in default format, pick the best suited format for font.
0x00000001	Download fonts as Type 1 outlines.
0x00000002	Download fonts as Type 3 bitmaps.
0x00000003	Download fonts as Type 42 fonts.
0x00000004	Same as 0x00000000.
0x00000005	Same as 0x00000000.

bReversePrint: If the value of this field is nonzero, print pages in reverse order; otherwise, print pages in normal order.

iLayout: The value of this field must be one of the following.

Value	Meaning
0x00000000	N-Up printing disabled.
0x00000001	Print 2-Up.
0x00000002	Print 4-Up.
0x00000003	Print 6-Up.
0x00000004	Print 9-Up.
0x00000005	Print 16-Up.
0x00000006	Print as a booklet.

iPSLevel: The value of this field must be one of the following.

Value	Meaning
0x00000001	Use only PostScript level 1 features.
0x00000002	Use only PostScript level 1 and level 2 features.
0x00000003	Use all PostScript features available for level 1, level 2, and level 3.

dwReserved2: The value of this field should be set to zero when sent and MUST be ignored on receipt.

wOEMExtra: The value of this field must be the total size of the private data that is used by the vendor-supplied driver **plug-in**. This data immediately follows this structure.

wVer: The value of this field must be 0x0010.

dwX: The value of this field specifies the width, in 1/1000th millimeter units, of the custom paper size. Used only if the **dmPaperSize** field of _DEVMODE is set to 0x7FFF.

dwY: The value of this field specifies the height, in 1/1000th millimeter units, of the custom paper size. Used only if the **dmPaperSize** field of _DEVMODE is set to 0x7FFF.

dwWidthOffset: The value of this field specifies the left unprintable margin, in 1/1000th of a millimeter, of the custom paper size. Used only if the **dmPaperSize** field of _DEVMODE is set to 0x7FFF.

dwHeightOffset: The value of this field specifies the top unprintable margin, in 1/1000th of a millimeter, of the custom paper size. Used only if the **dmPaperSize** field of _DEVMODE is set to 0x7FFF.

wFeedDirection: The value of this field must be one of the following.

Value	Meaning
0x0000	The paper is physically fed into the print mechanism with its long edge first.
0x0001	The paper is physically fed into the print mechanism with its short edge first.
0x0002	The paper is physically fed into the print mechanism with its long edge first, but upside down.
0x0003	The paper is physically fed into the print mechanism with its long edge first, but upside down.

wCutSheet: The value of this field is zero for roll-fed custom paper sizes and nonzero for cut sheet custom paper. This field is used only if the **dmPaperSize** field of _DEVMODE is set to 0x7FFF.

dwReserved3: The value of this field should be set to zero when sent and MUST be ignored on receipt.

dwChecksum32: The value of this field is the checksum of the names of the vendor-defined features and feature options that are supported by the printer model, as provided by the printer driver. The checksum is calculated using the 32-bit **cyclic redundancy check (CRC)** function defined in section 3.2.9 of [IEEE802.3-2008]. For each feature, the checksum is calculated on the null-terminated **ASCII** string representations of the feature name and each of the feature options in order. The checksum is accumulated in this manner over all of the features supported by the printer model.

dwOptions: The value of this field specifies the number of entries in the **aOptions** array.

aOptions: This field is an array that is 512 bytes long and contains the options selected by the user for each vendor-defined feature. Unused fields should be initialized to zero. The meaning of the entries differs for each supported printer model.

<158> Section 2.2.2.1.2: **UNIDRV** is the generic Windows core printer driver for all printers that do not use PostScript. It stores its private data in the _DEVMODE **dmDriverExtraData** area using the following structures.

The **dmDriverExtraData** field contains one of the **UNIDRVEXTRA** structures, immediately followed by zero or one JTEXP (section 2.2.2.1.4) structure, followed by zero or more OEM_DMEXTRA (section 2.2.2.1.3) structures.

These structures are not part of the protocol defined in this specification, and they are subject to change without notice. Implementations of compatible drivers must check the structure version and discard any data they do not handle.

UNIDRVEXTRA3_4: defined by the **UNIDRV** driver released with Windows NT 3.5 and Windows NT 4.0. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** field of the _DEVMODE structure is 0x0301.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
wReserved[0]																...															
...																(repeats for total of 56 reserved words)															

wReserved: This field is no longer used by **UNIDRV**.

UNIDRVEXTRA500: defined by the **UNIDRV** driver released with Windows 2000 and Windows XP. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** field of the DEVMODE structure is 0x0500.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
dwSignature																															
wVer																sPadding															
wSize																wOEMExtra															
dwChecksum32																															
dwFlags																															
bReversePrint																															
iLayout																															
iQuality																															
wReserved																...															
...																...															
...																...															
dwOptions																															
aOptions[0]																															
...																															

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
(repeats for a total of 128 aOptions array elements)																															
aOptions[127]																															

sPadding: The value of this field SHOULD be set to zero when sent and MUST be ignored on receipt.

wSize: This field is the same as **wCoreFullSize** in **UNIDRVEXTRA**.

UNIDRVEXTRA: defined by the **UNIDRV** driver released with Windows Vista, Windows Server 2008, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview. This structure is used for **dmDriverExtraData** if the **dmDriverVersion** field of the DEVMODE structure is 0x0600.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
dwSignature																															
wVer																wCoreJTExpSize															
wCoreFullSize																wOEMExtra															
dwChecksum32																															
dwFlags																															
bReversePrint																															
iLayout																															
iQuality																															
wReserved																...															
...																...															
...																...															
dwOptions																															

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
aOptions[0]																															
...																															
(repeats for a total of 128 aOptions array elements)																															
aOptions[127]																															
dwNupDirection																															
dwNupBorderFlags																															
dwBookletFlags																															

wCoreJTExpSize: The value of this field specifies the size of the JTEXP structure if one is present, following directly after this structure.

wCoreFullSize: The value of this field specifies the size of the **UNIDRVEXTRA** structure plus the value of **wCoreJTExpSize**.

dwNupDirection: This field is used only if N-Up printing is selected, and it must be one of the following values:

Value	Meaning
0x00000001	Print N-Up pages left-to-right, top-to-bottom.
0x00000002	Print N-Up pages top-to-bottom, left-to-right.
0x00000004	Print N-Up pages right-to-left, top-to-bottom.
0x00000008	Print N-Up pages top-to-bottom, right-to-left.

dwNupBorderFlags: This field is used only if N-Up Printing is selected, and it must be one of the following values:

Value	Meaning
0x00000000	Print borders around N-Up pages.
0x00000001	Do not print borders around N-Up pages.

dwBookletFlags: This field is used only if booklet printing is selected, and it must be one of the following values:

Value	Meaning
0x00000000	Print booklet so that pages flip to the left (western style).

Value	Meaning
0x00000001	Print booklet so that pages flip to the right.

Description of common members for **UNIDRVEXTRA** structures:

dwSignature: The value of this field must be 0x44494E55.

dwFlags: The value of this field must be the result of the bitwise OR of zero or more of the following values:

Value	Meaning
0x00000002	Print text as graphics (do not use fonts).
0x00000010	Do not use EMFSPOOL spooling.
0x00000080	Use Custom Quality **halftoning**.

bReversePrint: If the value of this field is nonzero, print pages in reverse order; otherwise, print pages in normal order.

iLayout: The value of this field must be one of the following values

Value	Meaning
0x00000000	N-Up printing disabled.
0x00000001	Print 2-Up.
0x00000002	Print 4-Up.
0x00000003	Print 6-Up.
0x00000004	Print 9-Up.
0x00000005	Print 16-Up.
0x00000006	Print as a booklet.

iQuality: The value of this field must be one of the following values:

Value	Meaning
0x00000000	Best Quality
0x00000001	Medium Quality
0x00000002	Draft Quality

wReserved: The value of this field SHOULD be set to zero when sent and MUST be ignored on receipt.

wOEMExtra: The value of this field specifies the total size of private data that is used by the vendor-supplied driver plug-in. This data immediately follows this structure.

wVer: The value of this field must be 0x0022.

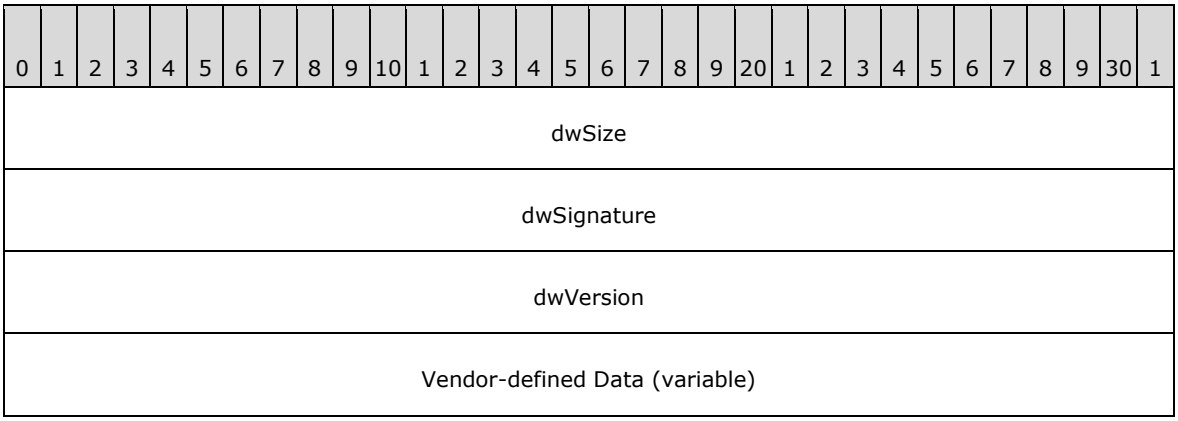
dwChecksum32: The value of this field is the checksum of the names of the vendor-defined features and feature options that are supported by the printer model, as provided by the printer driver. The

checksum is calculated using the 32-bit CRC function defined in section 3.2.9 of [IEEE802.3-2008]. For each feature, the checksum is calculated on the null-terminated ASCII string representations of the feature name and each of the feature options in order. The checksum is accumulated in this manner over all of the features supported by the printer model.

dwOptions: The value of this field is the number of entries in the **aOptions** array.

aOptions: This field is an array that is 512 bytes in length and contains the options selected by the user for each vendor-defined feature. Unused fields should be initialized to zero. The meaning of the entries differs for each supported printer model.

<159> Section 2.2.2.1.3: This is the **OEM_DMEXTRA** structure, which contains the _DEVMODE **dmDriverExtraData** defined by vendor-supplied driver plug-in modules.



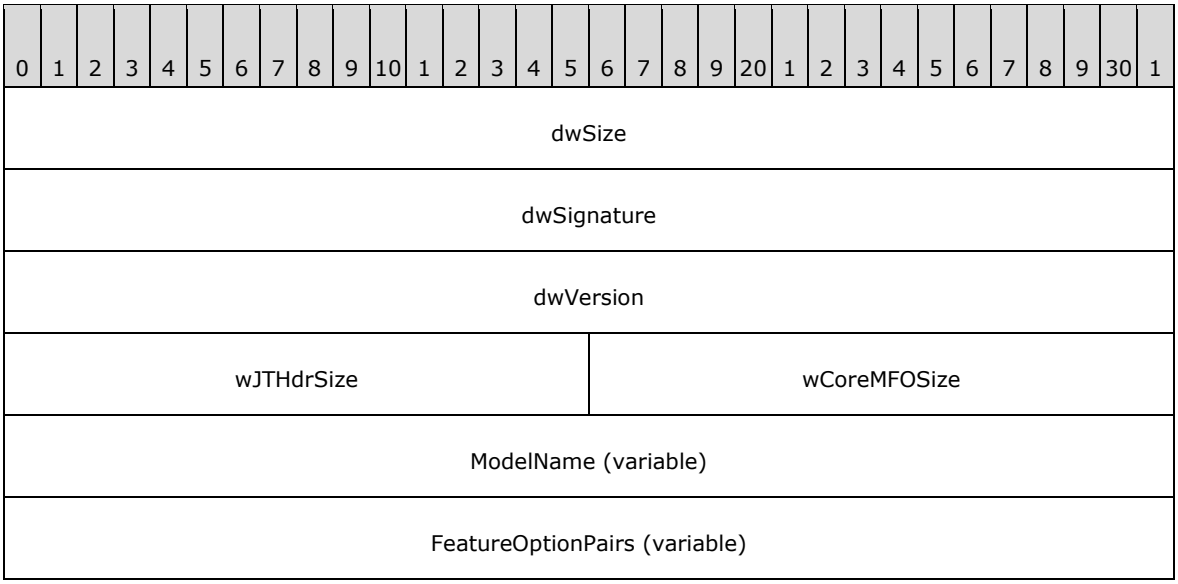
dwSize: Must be the total size of the vendor-defined data.

dwSignature: Must be a vendor-defined unique number.

dwVersion: Must be the version of the vendor-supplied plug-in.

Vendor-defined Data (variable): A variable-length field that holds vendor-defined data.

<160> Section 2.2.2.1.4: This is the **JTEXP** structure, which contains the _DEVMODE **dmDriverExtraData** defined by Windows to hold feature selection information.



dwSize: Must be the total size of the **JTEXP** structure. This size must be large enough to accommodate any possible combination of options for the vendor-specified features stored in **FeatureOptionPairs**.

dwSignature: Must be 0x534D544A.

dwVersion: Must be zero.

wJTHdrSize: Must be set to 16.

wCoreMFOSize: Must be the combined size of **ModelName** and **FeatureOptionPairs**. This must be exactly 16 bytes less than **dwSize**.

ModelName (variable): Must be a zero-terminated UTF-16LE encoded string specifying the name of the printer model.

FeatureOptionPairs (variable): Must be a concatenation of an even number of zero-terminated ASCII strings, terminated by an additional zero character. Each pair of two consecutive strings specifies a vendor-defined feature and the currently selected option for that feature. Each printer driver provides its own list of features and possible option values for each feature.

<161> Section 2.2.2.4.3: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<162> Section 2.2.2.4.4: This feature is supported on the following Windows version:

- Windows 2000

<163> Section 2.2.2.4.5: This feature is supported on the following Windows versions:

- Windows 2000

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<164> Section 2.2.2.4.6: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<165> Section 2.2.2.4.7: This feature is supported on the following Windows versions:

- Windows XP

- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<166> Section 2.2.2.4.7: Windows uses INF files for installation configuration data. For more information, see [MSDN-UINF] for more details.

<167> Section 2.2.2.4.8: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<168> Section 2.2.2.4.10: Windows print clients use this field to detect changes to printer driver files and to decide whether to update their local copies.

<169> Section 2.2.2.5.2: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<170> Section 2.2.2.6.3: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<171> Section 2.2.2.6.4: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<172> Section 2.2.2.9.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<173> Section 2.2.2.9.5: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<174> Section 2.2.2.9.6: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<175> Section 2.2.2.9.7: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP

- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<176> Section 2.2.2.9.8: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<177> Section 2.2.2.9.9: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP

- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<178> Section 2.2.2.11: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<179> Section 2.2.2.12: The Windows print server obtains this checksum by calling the **GdiQueryFonts** API method.

<180> Section 2.2.2.13: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<181> Section 2.2.2.13: In Windows implementations, the driver version is matched to the version portion of the INF file **DriverVer** member. For information about INF file syntax, see [MSDN-UINF].

<182> Section 2.2.2.14.4: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<183> Section 2.2.2.14.5: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<184> Section 2.2.2.15.3: This feature is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<185> Section 2.2.3.1: This feature is supported on the following Windows version:

- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<186> Section 2.2.3.1: This feature is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<187> Section 2.2.3.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003

- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<188> Section 2.2.3.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<189> Section 2.2.3.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<190> Section 2.2.3.2: These change notification flags are supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<191> Section 2.2.3.3: These job notification values are supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<192> Section 2.2.3.4: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<193> Section 2.2.3.6.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<194> Section 2.2.3.6.1: PRINTER_CHANGE_ALL (0x7777FFFF) is supported by the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2

<195> Section 2.2.3.6.1: PRINTER_CHANGE_ALL_2 (0x7F77FFFF) is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<196> Section 2.2.3.6.2: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<197> Section 2.2.3.6.2: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<198> Section 2.2.3.7: Windows displays this printer in its list of network-available printers.

<199> Section 2.2.3.7: Windows displays this printer in its list of network-available printers.

<200> Section 2.2.3.7: Windows displays this printer in its list of network-available printers.

<201> Section 2.2.3.7: Windows displays this printer in its list of network-available printers.

<202> Section 2.2.3.7: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<203> Section 2.2.3.8: 3D printers are not supported on the following Windows versions:

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 98
- Windows 2000
- Windows Millennium Edition
- Windows XP
- Windows Server 2003

- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012

<204> Section 2.2.3.8: This printer notification value is not supported on the following Windows versions:

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 98
- Windows 2000
- Windows Millennium Edition
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012

<205> Section 2.2.3.8: This printer notification value is not supported on the following Windows versions:

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 98
- Windows 2000
- Windows Millennium Edition
- Windows XP

- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012

<206> Section 2.2.3.8: Unless noted otherwise, these printer notification values are not supported on the following Windows versions:

- Windows NT 3.1
- Windows 98
- Windows Millennium Edition

<207> Section 2.2.3.8: This feature is not supported on the following Windows versions:

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 98
- Windows 2000
- Windows Millennium Edition
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2

<208> Section 2.2.3.8: This feature is not supported on the following Windows versions:

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 98

- Windows Millennium Edition

<209> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003

<210> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 2000
- Windows Server 2003

<211> Section 2.2.3.10: Windows does not use this key name remotely.

<212> Section 2.2.3.10: Windows does not use this key name remotely.

<213> Section 2.2.3.10: Windows does not use this key name remotely.

<214> Section 2.2.3.10: Windows does not use this key name remotely.

<215> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003

<216> Section 2.2.3.10: Windows does not use this key name remotely.

<217> Section 2.2.3.10: Windows does not use this key name remotely.

<218> Section 2.2.3.10: Windows does not use this key name remotely.

<219> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<220> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<221> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<222> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<223> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<224> Section 2.2.3.10: This feature is supported on the following Windows versions:

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<225> Section 2.2.3.10: Windows by default does not configure this value, allowing drivers with a driver version (see *cVersion* in section 2.2.1.3.1) of 0x00000004 to run printer UI applications if available. An administrator can change this behavior by creating the "HKLM\Software\Policies\Microsoft\Windows NT\Printers\V4DriverDisallowPrinterUIApp" REG_DWORD value in the registry and initializing this value with 0x00000001. This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<226> Section 2.2.3.10.1: The dwBuildNumber value for OSVERSIONINFO and OSVERSIONINFOEX for specific versions of Windows is shown in the table that follows.

Version	dwBuildNumber value
Windows 8.1 and Windows Server 2012 R2	>= 9431
Windows 8 and Windows Server 2012	>= 9200
Windows 7 and Windows Server 2008 R2	>= 7007
Windows Vista SP1 and Windows Server 2008	>= 6001
Windows Vista and Windows Server 2008	>= 6000
Windows XP SP1	>= 2196

Version	dwBuildNumber value
Windows XP and Windows Server 2003	>= 2196
Windows 2000	>= 1382
Windows NT 4.0	>= 1381

<227> Section 2.2.3.10.3: This value indicates Windows 2000 Professional operating system, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, or Windows Server 2016 Technical Preview.

<228> Section 2.2.3.10.3: This value indicates Windows 2000 Server operating system, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.

<229> Section 2.2.3.10.3: This value indicates Windows 2000 Server, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.

<230> Section 2.2.3.10.5: Windows uses the following values, which can be combined to specify multiple product suites.

Value	Meaning
VER_SUITE_SMALLBUSINESS 0x00000001	Microsoft Small Business Server was once installed on the system, but it might have been upgraded to another version of Windows.
VER_SUITE_ENTERPRISE 0x00000002	Windows NT Server 4.0 operating system, Enterprise Edition, Windows 2000 Advanced Server operating system, Windows Server 2003 Enterprise Edition operating system, or Windows Server 2008 Enterprise operating system is installed.
VER_SUITE_BACKOFFICE 0x00000004	Microsoft BackOffice components are installed.
VER_SUITE_TERMINAL 0x00000010	Terminal Services is installed. If **VER_SUITE_TERMINAL** is set but **VER_SUITE_SINGLEUSERTS** is not set, the system is running in **application server mode**.
VER_SUITE_SMALLBUSINESS_RESTRICTED 0x00000020	Microsoft Small Business Server is installed with the restrictive client license in force.
VER_SUITE_EMBEDDEDNT 0x00000040	Windows XP Embedded is installed.
VER_SUITE_DATACENTER 0x00000080	Windows 2000 Datacenter Server operating system, Windows Server 2003 Datacenter Edition operating system, or Windows Server 2008 Datacenter operating system is installed.
VER_SUITE_SINGLEUSERTS 0x00000100	Remote Desktop is supported, but only one interactive session. This value is set unless the system is running in application server mode.
VER_SUITE_PERSONAL 0x00000200	Windows XP Home Edition operating system, Windows Vista Home Basic, or Windows Vista Home Premium is installed.
VER_SUITE_BLADE 0x00000400	Windows Server 2003 Web Edition operating system is installed.

Value	Meaning
VER_SUITE_STORAGE_SERVER 0x00002000	Windows Storage Server 2003 R2 or Windows Storage Server 2003 is installed.
VER_SUITE_COMPUTE_SERVER 0x00004000	Windows Server 2003 operating system Compute Cluster Edition is installed.
VER_SUITE_WH_SERVER 0x00008000	Windows Home Server is installed.

<231> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<232> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<233> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<234> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<235> Section 2.2.3.11: Windows reports as the last driver identifier in the list the Microsoft Universal Sharing driver ({A9838643-5862-4F72-ACAF-F4CECE098759}). This value is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<236> Section 2.2.3.11: This value is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<237> Section 2.2.3.11: This value is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<238> Section 2.2.3.11: This value is supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<239> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<240> Section 2.2.3.11: This feature is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<241> Section 2.2.3.11: The default value on Windows implementations is 10 MB.

<242> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<243> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows Vista

- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<244> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<245> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<246> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<247> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<248> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008

- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<249> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<250> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista

- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<251> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<252> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<253> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<254> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<255> Section 2.2.3.12: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7

- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<256> Section 2.2.4.2: Windows supports the spool file data type formats EMFSPOOL, RAW format, XML Paper Specification (XPS), and custom data types defined by custom print processors. For more information on these formats, see [MS-EMFSPOOL], [MSDN-SPOOL], [MSDN-XMLP], and [MSFT-XMLPAPER].

Windows standard print processors support the following case-insensitive data type name strings:

Name	Description
"RAW"	RAW
"RAW [FF appended]"	RAW
"RAW [FF auto]"	RAW
"NT EMF 1.003"	EMFSPOOL
"NT EMF 1.006"	EMFSPOOL
"NT EMF 1.007"	EMFSPOOL
"NT EMF 1.008"	EMFSPOOL
"TEXT"	Plain text
"XPS_PASS"	XPS passthrough (supported on Windows Vista, Windows Server 2008, Windows Vista SP1, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview).
"XPS2GDI"	XPS data to be converted into a new Graphics Device Interface (GDI) print job to send to the device (supported on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview).

<257> Section 2.2.4.3: Windows restricts driver name strings to 260 characters, including the terminating null character.

<258> Section 2.2.4.4: The environment name strings "Windows 4.0" and "Windows NT x86" are supported on the following Windows versions:

- Windows NT 3.5

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

The environment name string "Windows IA64" is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

The environment name string "Windows x64" is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

The environment name string "Windows ARM" is supported on the following Windows versions, and only on print servers with the "Windows ARM" operating system environment:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<259> Section 2.2.4.5: Windows restricts form name strings to 260 characters, including the terminating null character.

<260> Section 2.2.4.7: Windows restricts key name strings to 260 characters, including the terminating null character.

<261> Section 2.2.4.8: Windows restricts monitor name strings to 260 characters, including the terminating null character.

<262> Section 2.2.4.9: IPv6 names are supported only on Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<263> Section 2.2.4.9: Windows uses the following pattern for local files:

```
NAME = <any TEXT except "\">

DIRECTORY = "\" 1#NAME

FILENAME = "\" 1#NAME

PATH = [alpha ":"] #DIRECTORY FILENAME
```

<264> Section 2.2.4.9: Windows restricts path name strings to 519 characters, including the terminating null character.

<265> Section 2.2.4.10: Windows uses the following patterns for port names:

```
PARALLEL_PORT = "LPT" DIGIT ":"

SERIAL PORT = "COM" DIGIT ":"

FILE_PORT = "FILE:"

USB_PORT = "USB" 1#DIGIT ":"

UNC PORT = SERVER NAME DIRECTORY FILENAME

LOCAL_FILE_PORT = PATH

PORT_NAME = (PARALLEL_PORT | SERIAL_PORT | FILE_PORT | USB_PORT |
    UNC_PORT | LOCAL_FILE_PORT)
```

where:

- **SERVER_NAME** is specified in section 2.2.4.16.
- **DIRECTORY** is specified in section 2.2.4.9.
- **PARALLEL_PORT** is used for devices attached through a parallel port.
- **SERIAL_PORT** is used for devices attached through a serial port.
- **FILE_PORT** is used to send data to a file.
- **USB_PORT** is used for devices attached through a **universal serial bus (USB)** port.
- **UNC_PORT** is used for network printers attached directly through an IP address or a network address.

Windows supports the pooling of ports. When printing to a printer associated with a pool of ports, the first available port is picked by the print server. Port pooling allows representation of multiple identical physical printers as a single logical printer. Pooled port names are represented as a comma-separated list of port names, for example, "LPT1:,LPT2:".

Clients connecting to a Windows print server need to be prepared to handle pooled ports correctly; for example, they cannot rely on individual port names enumerated by the RpcEnumPorts method to match the string pointed to by the **pPortName** member of a PRINTER_INFO (section 2.2.1.10) structure.

<266> Section 2.2.4.11: Windows restricts print processor name strings to 260 characters, including the terminating NULL character.

<267> Section 2.2.4.12: Windows restricts print provider name strings to 260 characters, including the terminating null character.

<268> Section 2.2.4.14: Windows uses the following postfix string (PRINTER_NAME_POSTFIX) values: "LocalOnly", "LocalsplOnly", and "DrvConvert". "LocalOnly" means that the client asks the server to use only local printer settings for RpcGetPrinterData and RpcSetPrinterData, and it specifies that the client is not interested in printing to this local printer but only in accessing the printer's local settings. "LocalsplOnly" and "DrvConvert" are treated the same way and mean that the client asks the server to open only the local printer with the respective name (PRINTER_NAME_PREFIX) if such local printer exists. These postfix strings are used on Windows during the upgrade of a printer driver and in other cases where it is preferred that a printer is accessed locally.

<269> Section 2.2.4.14: Windows restricts printer name strings to 539 characters (259 + 260 + 20), including all backslashes, other separators, and the terminating null character.

<270> Section 2.2.4.16: IPv6 names are supported only on Windows Vista, Windows Server 2008, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<271> Section 2.2.4.16: Windows restricts server name strings to 259 characters, including the two leading backslash characters and trailing backslash character.

<272> Section 2.2.4.18: Windows restricts value name strings to 260 characters, including the terminating null character.

<273> Section 2.3.1: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, print servers can publish printers to the Active Directory. In Windows NT 4.0 and Windows NT 3.51, print servers do not interact with the directory.

<274> Section 2.3.3.1: Windows print servers attempt to set the following attributes based on internal print spooler state:

- uNCName
- serverName
- shortServerName
- versionNumber
- printerName
- description
- driverName
- location
- portName
- printStartTime
- printEndTime
- printKeepPrintedJobs

- printSeparatorFile
- printShareName
- printSpooling
- priority
- url
- flags
- printStatus
- printAttributes
- driverVersion

Windows print servers attempt to set the following attributes based on the results of querying the printer driver configuration module:

- printBinNames
- printCollate
- printColor
- printDuplexSupported
- printStaplingSupported
- printMaxXExtent
- printMaxYExtent
- printMinXExtent
- printMinYExtent
- printMediaSupported
- printMediaReady
- printNumberUp
- printMemory
- printOrientationsSupported
- printMaxResolutionSupported
- printLanguage
- printRate
- printRateUnits
- printPagesPerMinute

All other attributes in the print schema can be set indirectly by print clients using RpcSetPrinterDataEx (section 3.1.4.2.18), as specified in section 2.3.3.1.

<275> Section 2.3.3.1: Windows print servers will publish a value under one of the "DsSpooler", "DsDriver", and "DsUser" keys to the directory, if its name corresponds to an attribute in the schema for the print queue object class.

<276> Section 2.3.3.2: Windows print servers persist a string representing the GUID for the published object in the directory. This string corresponds to the **pszObjectGUID** member of the PRINTER_INFO_7 structure. Windows print servers then call the **IDL_DRSCrackNames** method of the DRSR protocol, documented in [MS-DRSR] section 4.1.4, on the domain controller to convert the GUID string into the fully qualified DN of the object.

<277> Section 2.3.3.2: When a Windows print server performs an LDAP update operation for a print queue, it always updates all of the LDAP attributes corresponding to the printer data values under one printer data key ("DsSpooler", "DsDriver", or "DsUser") where the server has stored attributes as described in section 2.3.3.1.

<278> Section 2.3.3.3: In Windows 2000, Windows XP, and Windows Server 2003, print clients search the GC for print queues in all naming contexts. In Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, print clients only search the current naming context by default, but users can still choose to search other naming contexts.

<279> Section 2.3.3.3: In Windows 2000, Windows XP, and Windows Server 2003, print client negotiate encryption for LDAP requests. In Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, print clients do not use encryption except when users choose to customize the printer search by specifying other NCs or more detailed filters.

<280> Section 2.3.3.4: By default, Windows servers never perform this periodic operation. An administrator can change this behavior by writing a time period, in minutes, to the "HKLM\Software\Policies\Microsoft\Windows NT\Printers\VerifyPublishedState" value (REG_DWORD type) in the registry.

<281> Section 2.3.3.4: On Windows servers that are configured as **writable** domain controllers (DC), the periodic search procedure is, by default, executed every 8 hours. This default period of 8 hours can be changed by writing the new time period, in minutes, to the "HKLM\Software\Policies\Microsoft\Windows NT\Printers\PruningInterval" value (REG_DWORD type) in the registry. The Windows servers that are not configured as writable domain controllers do not execute this periodic search.

<282> Section 3.1.1: Windows implements port pooling. A printer object can manage references to multiple port objects. A physical print device is connected to each of the port objects, but the physical print devices are substantially the same. Windows transparently distributes incoming jobs to the multiple port objects to balance workload.

<283> Section 3.1.1: Windows creates the print job SECURITY_DESCRIPTOR by inheriting the SECURITY_DESCRIPTOR of the printer (which by default grants JOB_ALL_ACCESS to members of the Administrators group and to the creator/owner client) and by adding an access allowed entry that grants JOB_READ access to the submitting client.

<284> Section 3.1.1: **Job Named Properties** are supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<285> Section 3.1.1: In Windows implementations, the printer driver version is a DWORD set to one of the following values:

Value	Meaning
0x00000001	Windows NT 3.51 user-mode printer drivers
0x00000002	Windows NT 4.0 user-mode printer drivers.
0x00000003	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 user-mode printer drivers.
0x00000004	Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview user-mode printer drivers.

<286> Section 3.1.1: Windows print servers initialize the list to contain the TCPMON port monitor, which provides support for **TCP**/IP-connected printers.

<287> Section 3.1.1: Print providers are Windows–specific and not required by this protocol.

<288> Section 3.1.1: In addition to the **list of print providers** persisted in the registry, the **list of print providers** on Windows print servers also includes a print provider named "Windows NT Local Print Provider".

<289> Section 3.1.1: The only versions of Windows servers that maintain such a list are Windows XP and Windows Server 2003. Administrators can configure this list. In all other versions of Windows, this list does not exist.

<290> Section 3.1.1: Windows print servers that maintain a **list of warned printer drivers** persist the list in an INF file called printupg.inf.

<291> Section 3.1.1: The **Job Named Properties** are supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<292> Section 3.1.1: Branch office print mode is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<293> Section 3.1.1: **Persistence of Abstract Data Model in the Registry**

Print system management tools such as **printmig**, **brm**, and other third-party applications read and write registry values that are used by the Windows print server to persist its abstract data model. Windows print servers also persist objects from the abstract data model and other settings in the registry, as described in the following table.

Note that the value of the registry key entry shown in the table below is wrapped to the following line at the backslash character for easier reading. The actual string value is one contiguous string.

For example: A value in the table such as

```
HKLM\
SYSTEM\
CurrentControlSet\
Control\
Print
```

represents a registry key path string that would be entered in a string variable as:

```
"HKLM\SYSTEM\CurrentControlSet\Control\Print"
```

Registry key	Value	Type	Description
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\			
	MaxRPCSize	REG_DWORD	Maximum buffer size accepted by the server in RPC calls. The default used, if this value is not present, is 50MB.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Environments\ [Env Name]\ Drivers			The **list of printer drivers** installed on the print server for the environment named [Env Name].
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Environments\ [Env Name]\ Drivers\ [Version]\ [Driver Name]			Information about the printer driver named [Driver Name] having the version identified by [Version] for the environment named [Env Name].

Registry key	Value	Type	Description
	Configuration File	REG_SZ	Persisted DRIVER_INFO member **pConfigFile**.
	Data File	REG_SZ	Persisted DRIVER_INFO member **pDataFile**.
	Driver	REG_SZ	Persisted DRIVER_INFO member **pDriverPath**.
	Help File	REG_SZ	Persisted DRIVER_INFO member **pHelpFile**.
	Monitor	REG_SZ	Persisted DRIVER_INFO member **pMonitorName**.
	Datatype	REG_SZ	Persisted DRIVER_INFO member **pDefaultDataType**.
	Dependent Files	REG_MULTI_SZ	Persisted DRIVER_INFO member **pDependentFiles**.
	Previous Names	REG_MULTI_SZ	Persisted DRIVER_INFO member **pszzPreviousNames**.
	Version	REG_DWORD	Persisted DRIVER_INFO member **cVersion**.
	Manufacturer	REG_SZ	Persisted DRIVER_INFO member **pMfgName**.
	OEM URL	REG_SZ	Persisted DRIVER_INFO member **pOEMUrl**.
	HardwareID	REG_SZ	Persisted DRIVER_INFO member **pHardwareID**.
	Provider	REG_SZ	Persisted DRIVER_INFO member **pProvider**.
	DriverDate	REG_BINARY	Persisted DRIVER_INFO member **ftDriverDate**.
	DriverVersion	REG_BINARY	Persisted DRIVER_INFO member **dwlDriverVersion**.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Environments\ [Env Name]\ Print Processors			The **list of print processors** installed on the print server for the environment named [Env Name].
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\			Information about the print processor named [Processor Name] for the environment named [Env Name].

Registry key	Value	Type	Description
Environments\ [Env Name]\ Print Processors\ [Processor Name]			
	Driver	REG_SZ	Persisted **pPathName** parameter used when print processor was added by the RpcAddPrintProcessor (section 3.1.4.8.1) method.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Forms			The **list of form objects** installed on the print server.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Monitors			The **list of language monitors** and port monitors installed on the print server.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Monitors\ [Monitor Name]			Information about the port monitor named [Monitor Name].
	Driver	REG_SZ	Persisted MONITOR_INFO_2 (section 2.2.1.8.2) member **pDLLName**.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Monitors\ LPR Port\ Ports			Information about LPR ports.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Monitors\ LPR Port\			Information about the LPR port named [Port Name].

Registry key	Value	Type	Description
Ports\ [Port name]			
	Server Name	REG_SZ	Host address of server exposing LPR port.
	Printer Name	REG_SZ	Queue name on LPR server.
HKLM\ SYSTEM\ CurrentControlSet\ Control\ Print\ Monitors\ Standard TCP/IP Port\ Ports			
HKLM\ SOFTWARE\ DigitalEquipmentCorporation \ Network Printing Software	All subkeys and values		Opaque data for DEC Networking monitors.
HKLM\ SOFTWARE\ Hewlett-Packard\ HP JetAdmin	All subkeys and values		Opaque data for HP JetAdmin monitors.
HKLM\ SOFTWARE\ Lexmark	All subkeys and values		Opaque data for Lexmark Networking monitors.
HKLM\ SOFTWARE\ Microsoft\ Windows NT operating system\ CurrentVersion\ Ports			List of all ports installed with the print server.
	[Port Name]	REG_SZ	Implementation-specific initialization parameters for the port named [Port Name]. If there are no initialization parameters, the value specifies an empty string.
HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Print\ Connections			The **list of per-machine connections** on the print server.

Registry key	Value	Type	Description
HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Print\ Printers			The **persisted list of printers** installed on the print server.
	DefaultSpoolDirectory	REG_SZ	The directory used by the print server for storing temporary print job data.
HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Print\ Printers\ [Printer Name]			Information about the printer named [Printer Name].
	Share Name	REG_SZ	Persisted PRINTER_INFO member **pShareName**.
	Print Processor	REG_SZ	Persisted PRINTER_INFO member **pPrintProcessor**.
	Datatype	REG_SZ	Persisted PRINTER_INFO member **pDatatype**.
	Parameters	REG_SZ	Persisted PRINTER_INFO member **pParameters**.
	ObjectGUID		Persisted PRINTER_INFO member **pszObjectGUID**.
	Printer Driver	REG_SZ	Persisted PRINTER_INFO member **pDriverName**.
	Default _DEVMODE	REG_SZ	Persisted PRINTER_INFO_8 member **pDevMode**.
	Priority	REG_DWORD	Persisted PRINTER_INFO member **Priority**.
	Default Priority	REG_DWORD	Persisted PRINTER_INFO member **DefaultPriority**.
	StartTime	REG_DWORD	Persisted PRINTER_INFO member **StartTime**.
	UntilTime	REG_DWORD	Persisted PRINTER_INFO member **UntilTime**.
	Separator File	REG_SZ	Persisted PRINTER_INFO member **pSepFile**.
	Location	REG_SZ	Persisted PRINTER_INFO member **pLocation**.

Registry key	Value	Type	Description
	Attributes	REG_DWORD	Persisted PRINTER_INFO member **Attributes**.
	Security	REG_BINARY	Persisted PRINTER_INFO member **pSecurityDescriptor**.
	Port	REG_SZ	Persisted PRINTER_INFO member **pPortName**.
HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Print\ Providers			The **list of print providers** installed on the print server.

Definitions for placeholders in the preceding table:

- [Env Name] is an environment name as specified in 2.2.4.4.
- [Version] is "Version-0", "Version-1", "Version-2", "Version-3", or "Version 4". The numbers 0–4 (0x00000000 – 0x00000004) are described in section 2.2.1.3.1, member cVersion.
- [Driver Name] is a driver name as specified in section 2.2.4.3.
- [Processor Name] is a print processor name as specified in section 2.2.4.11.
- [Monitor Name] is a monitor name as specified in section 2.2.4.8.
- [Port Name] is a port name as specified in section 2.2.4.10.
- [Printer Name] is a printer name as specified in section 2.2.4.14.

<294> Section 3.1.1: **Locations of Print System Components in the Server File System**

Print system management tools such as **printmig**, **brm**, and other, third-party applications remotely access files that are loaded or executed by the Windows print server. Windows print server loads or executes files as described in the following table.

File description	Location on print server file system
Printer driver files	%systemroot%\system32\spool\drivers\[env-dir]\[version-dir]
Print processor files	%systemroot%\system32\spool\prtprocs\[env-dir]
Separator files	%systemroot%\system32\spool\sepfiles

Definitions for placeholders in the preceding table:

[env-dir] specifies one of the following strings, depending on the [Env Name] of the driver:

- "W32X86" for the environment string "Windows NT x86"
- "IA64" for the environment string "Windows IA64"
- "WIN40" for the environment string "Windows 4.0"

- "W32ALPHA" for the environment string "Windows NT Alpha_AXP"
- "X64" for the environment string "Windows X64"
- "ARM" for the environment string "Windows ARM"

[version-dir] is one of the strings "0", "1", "2", "3", or "4", matching the DRIVER_INFO member **cVersion**.

The %systemroot% environment variable is shared by the print server using the share name "admin$".

<295> Section 3.1.3: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, print servers can publish printers to the Active Directory, and print clients can search the Active Directory for printers. In Windows NT 3.51 and Windows NT 4.0, print clients and servers do not interact with the directory.

<296> Section 3.1.4: The job named property management methods are supported on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<297> Section 3.1.4: Branch office print remote logging methods are supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<298> Section 3.1.4:

Opnum	Description
37	Only used locally by Windows, never remotely.
38	Only used locally by Windows, never remotely.
43	Only used locally by Windows, never remotely.
44	Only used locally by Windows, never remotely.
45	Only used locally by Windows, never remotely.
49	Only used locally by Windows, never remotely.
50	Only used locally by Windows, never remotely.

Opnum	Description
54	Only used locally by Windows, never remotely.
55	Only used locally by Windows, never remotely.
57	Only used locally by Windows, never remotely.
63	Only used locally by Windows, never remotely.
64	Only used locally by Windows, never remotely.
68	Only used locally by Windows, never remotely.
74	Only used locally by Windows, never remotely.
75	Only used locally by Windows, never remotely.
76	Only used locally by Windows, never remotely.
83	Only used locally by Windows, never remotely.
90	Only used locally by Windows, never remotely.
91	Only used locally by Windows, never remotely.
92	Only used locally by Windows, never remotely.
93	Only used locally by Windows, never remotely.
94	Only used locally by Windows, never remotely.
95	Only used locally by Windows, never remotely.
98	Only used locally by Windows, never remotely.
99	Only used locally by Windows, never remotely.
100	Only used locally by Windows, never remotely.
101	Only used locally by Windows, never remotely.
102	Only used locally by Windows, never remotely.
103	Only used locally by Windows, never remotely.
104	Only used locally by Windows, never remotely.
105	Only used locally by Windows, never remotely.
106	Only used locally by Windows, never remotely.
107	Only used locally by Windows, never remotely.
108	Only used locally by Windows, never remotely.
109	Only used locally by Windows, never remotely.
114	Only used locally by Windows, never remotely.
115	Only used locally by Windows, never remotely.

<299> Section 3.1.4.1.4: IPv6 names are supported only on Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

<300> Section 3.1.4.1.4: All Windows client implementations derive the RPC binding directly from the STRING_HANDLE binding parameter, and all Windows server implementations perform this validation step.

<301> Section 3.1.4.1.5: All Windows client implementations derive the RPC binding directly from the STRING_HANDLE binding parameter, and all Windows server implementations perform this validation step.

<302> Section 3.1.4.1.8.1: In Windows, when the *pDevModeContainer* parameter is not declared with the "unique" IDL attribute and is set to a value of NULL, the underlying RPC protocol [MS-RPCE] implementation stops the invalid call and throws an exception before the call reaches the server.

<303> Section 3.1.4.1.8.2: In Windows, when the *pDocInfoContainer* parameter is not declared with the "unique" IDL attribute and is set to a value of NULL, the underlying RPC protocol [MS-RPCE] implementation stops the invalid call and throws an exception before the call reaches the server.

<304> Section 3.1.4.1.8.7: If the value of the **pSecurity** member in the SECURITY_CONTAINER structure is NULL, a default security descriptor is used. Security descriptors are specified in [MS-DTYP].

Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview:

The owner in the security descriptor is the local system. The discretionary access control list (DACL) contains **access control entries (ACEs)**, which grant the following permissions:

Full control to the administrator's group and to the user who added the printer.

Print permissions to "everyone".

Permissions to control (cancel, pause, resume) the job to the user who submits the job.

Windows NT, Windows 2000, Windows XP, and Windows Server 2003:

The owner in the security descriptor is the user who added the printer. The DACL contains ACEs that grant the following permissions:

Full control to administrators and power users groups.

Print permissions to "everyone".

Permissions to control (cancel, pause, resume) the job to the user who submits the job.

<305> Section 3.1.4.1.8.8: Windows does not use the following members: **pUserName**, **dwBuildNum**, **dwMajorVersion**, **dwMinorVersion**, and **wProcessorArchitecture**. **pMachineName** is used only if the server cannot determine the client machine name using remote procedure call (RPC) functions. The **pMachineName** member can be NULL.

<306> Section 3.1.4.1.8.9: This validation step can be performed on Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview.

An administrator can configure a print server to not perform this validation step by writing a nonzero value to the "HKLM\System\CurrentControlSet\Control\Print\ARMTestMode" value (REG_DWORD type) in the registry.

<307> Section 3.1.4.1.10: This verification is done by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<308> Section 3.1.4.2.1: Windows Vista, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview servers can return 0x00004001 for error conditions or if no printers are found matching the requested flags.

<309> Section 3.1.4.2.1: Windows Vista, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview servers can return 0x00004001 for error conditions or if no printers are found matching the requested flags.

<310> Section 3.1.4.2.1: The Windows server checks that the client user has the SERVER_ACCESS_ENUMERATE permission.

<311> Section 3.1.4.2.1: Windows returns the printers in alphabetical order.

<312> Section 3.1.4.2.1: Windows Vista, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview servers can return 0x00004001 for error conditions or if no printers are found matching the requested flags.

<313> Section 3.1.4.2.1: Windows uses a policy-defined period (default: 60 minutes) from the first call to RpcAddPrinter with *Level* set to 0x00000001 to determine whether ERROR_CAN_NOT_COMPLETE is returned.

<314> Section 3.1.4.2.1: In Windows, the enumeration is restricted as follows:

- Unshared printers are enumerated only if the user has the SERVER_ACCESS_ADMINISTER permission.
- Only those printers whose security descriptor grants PRINTER_ACCESS_USE to the caller are enumerated.

<315> Section 3.1.4.2.3: The Windows server checks that the client user has implementation-specific permission, such as SERVER_ACCESS_ADMINISTER, to install a printer.

<316> Section 3.1.4.2.3: Windows stores the time when the printer is added to the list. When a printer is added, Windows removes any printer on the list that was added more than 70 minutes ago. Windows stores a maximum of 256 printers in the list. If the limit is reached, no new printers are added, and ERROR_OUTOFMEMORY, as specified in [MS-ERREF], is returned.

<317> Section 3.1.4.2.4: In Windows, the server verifies that the printer object has been opened with an access value including the generic **DELETE** permission, for example, PRINTER_ALL_ACCESS.

<318> Section 3.1.4.2.4: The Windows server verifies that the client user has the PRINTER_ACCESS_ADMINISTER permission.

<319> Section 3.1.4.2.4: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<320> Section 3.1.4.2.5: The Windows server verifies for a printer object that the object has been opened with PRINTER_ACCESS_ADMINISTER permission, and for a server object that the object has been opened with SERVER_ACCESS_ADMINISTER permission.

<321> Section 3.1.4.2.5: In Windows 2000, Windows Millennium Edition, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, the Windows server can return the ERROR_IO_PENDING error code from the RpcSetPrinter call when the **Level** member of the **PRINTER_CONTAINER** pointed to by *pPrinterContainer* is 0x7 (indicating a directory service operation) and the server implementation uses an asynchronous mechanism to perform the operation.

<322> Section 3.1.4.2.5: This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<323> Section 3.1.4.2.6: In the following Windows versions, this parameter is a handle to a printer or print server object if the value of the *Level* parameter is 0x00000003.

- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<324> Section 3.1.4.2.6: The Windows server verifies for a printer object that the object has been opened with PRINTER_ACCESS_USE permission, and for a server object that the object has been opened with SERVER_ACCESS_ENUMERATE permission.

<325> Section 3.1.4.2.7: In Windows, the detection of the string "ChangeID" is case-insensitive.

<326> Section 3.1.4.2.7: In Windows, the data identified by *pValueName* for printer objects is stored in the registry under the key "PrinterDriverData"; therefore, RpcGetPrinterDataEx is used with *pKeyName* pointing to the string "PrinterDriverData" to access the identical set of values.

<327> Section 3.1.4.2.8: In Windows, the detection of the string "ChangeID" is case-insensitive.

<328> Section 3.1.4.2.8: In Windows, if *hPrinter* is a server object handle, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

If *hPrinter* is a printer object handle, the server checks that the client user has PRINTER_ACCESS_ADMINISTER permission. In Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview, the server checks that the client user has either PRINTER_ACCESS_ADMINISTER or PRINTER_ACCESS_MANAGE LIMITED permission.

<329> Section 3.1.4.2.8: In Windows the data identified by *pValueName* for printer objects is stored in the registry under the key "PrinterDriverData"; therefore, RpcSetPrinterDataEx is used with *pKeyName* pointing to the string "PrinterDriverData" to access the identical set of values.

For server objects, there is no aliasing of value and key because the server ignores the *pKeyName* parameter of the RpcSetPrinterDataEx method when called with a handle to a server object.

<330> Section 3.1.4.2.11: In Windows, the underlying RPC protocol [MS-RPCE] implementation stops the invalid call and throws an exception before the call reaches the server.

<331> Section 3.1.4.2.11: In Windows, the underlying RPC protocol [MS-RPCE] implementation stops the invalid call and throws an exception before the call reaches the server.

<332> Section 3.1.4.2.11: When creating print jobs in EMFSPOOL format [MS-EMFSPOOL], Windows print clients use the UNIVERSAL_FONT_ID structures returned by RpcPlayGdiScriptOnPrinterIC to determine if the server has copies of specific fonts, and if so, the client does not embed those fonts in the EMFSPOOL data sent to the print server.

<333> Section 3.1.4.2.14: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<334> Section 3.1.4.2.15: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<335> Section 3.1.4.2.15: In Windows, the server checks that the client user has implementation-specific permissions to install a printer, typically SERVER_ACCESS_ADMINISTER.

<336> Section 3.1.4.2.15: Windows stores the time when the printer is added to the list. When a printer is added, Windows removes any printer on the list that was added more than a policy-defined period (default 70 minutes) ago. Windows stores a maximum of 256 printers in the list. If the limit is reached, no new printers are added, and ERROR_OUTOFMEMORY is returned, as specified in [MS-ERREF].

<337> Section 3.1.4.2.15: Windows fails this call if, at the time of this call, the server does not have installed all the following: the printer driver, the port, and the print processor.

<338> Section 3.1.4.2.16: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<339> Section 3.1.4.2.16: In Windows, for printer objects, the data identified by *pValueName* is stored in the registry under the key named "PrinterDriverData"; therefore, RpcEnumPrinterDataEx is used with *pKeyName* pointing to the string "PrinterDriverData" to access the identical set of values.

<340> Section 3.1.4.2.17: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<341> Section 3.1.4.2.17: In Windows, the detection of the string "ChangeID" is case-insensitive.

<342> Section 3.1.4.2.17: In Windows, the server checks that the client user has PRINTER_ACCESS_ADMINISTER permission.

<343> Section 3.1.4.2.17: In Windows, the data identified by *pValueName* for printer objects is stored in the registry under the key named "PrinterDriverData"; therefore, RpcDeletePrinterDataEx is used with *pKeyName* pointing to the string "PrinterDriverData" to access the identical set of values.

<344> Section 3.1.4.2.18: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<345> Section 3.1.4.2.18: In Windows, the detection of the string "ChangeID" is case-insensitive.

<346> Section 3.1.4.2.18: In Windows, if *hPrinter* is a server object handle, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission. If *hPrinter* is a printer object handle, the server checks that the client user has PRINTER_ACCESS_ADMINISTER permission.

<347> Section 3.1.4.2.19: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<348> Section 3.1.4.2.19: This verification is done by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<349> Section 3.1.4.2.19: This verification is done by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<350> Section 3.1.4.2.20: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<351> Section 3.1.4.2.21: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<352> Section 3.1.4.2.22: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<353> Section 3.1.4.2.22: In Windows, the detection of the string "ChangeID" is case-insensitive.

<354> Section 3.1.4.2.22: In Windows, the server checks that the client user has the PRINTER_ACCESS_ADMINISTER permission.

<355> Section 3.1.4.2.23: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008

- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<356> Section 3.1.4.2.23: In Windows, the server checks that the client user has the PRINTER_ACCESS_ADMINISTER permission.

<357> Section 3.1.4.2.24: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<358> Section 3.1.4.2.24: In Windows, the name of the print provider file is used and is stored in the Windows registry. If *pProvider* is NULL, the Windows operating system uses Win32spl.dll as the name of the executable object.

Print providers are a Windows implementation detail and are not required by this protocol. Windows print clients do not use a non-NULL *pProvider* parameter remotely, but third-party software can do so. As there is no protocol method to enumerate print providers remotely, a client would need specific knowledge about the internal implementation of the server to specify a meaningful print provider name.

<359> Section 3.1.4.2.24: In Windows, the server checks that the client user has the SERVER_ACCESS_ADMINISTER permission.

<360> Section 3.1.4.2.25: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<361> Section 3.1.4.2.25: The Windows server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<362> Section 3.1.4.2.26: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<363> Section 3.1.4.2.27: This feature is supported on the following Windows versions:

- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<364> Section 3.1.4.2.27: Port monitors shipping with Windows support the following actions only for TCPMON and WSDMON, and only if connected to a printer capable of bidirectional communication: BIDI_ACTION_ENUM_SCHEMA, BIDI_ACTION_GET, and BIDI_ACTION_GET_ALL. All other Port Monitors shipping with Windows do not support RpcSendRecvBidiData.

<365> Section 3.1.4.2.27: Port monitors shipping with Windows 10 and Windows Server 2016 Technical Preview support the BIDI_ACTION_GET_WITH_ARGUMENT action for WSDMON if connected to a printer capable of bidirectional communication.

<366> Section 3.1.4.2.27: The Windows system port monitors do not support any additional action strings.

<367> Section 3.1.4.2.27: TCPMON and WSDMON check that the port or printer object has been opened with an access value that includes the PRINTER_ACCESS_USE bit.

<368> Section 3.1.4.3.1: In Windows, the command codes JOB_CONTROL_CANCEL (0x00000003) and JOB_CONTROL_DELETE (0x00000005) are equivalent. The redundancy is caused by historical reasons, but both must be supported.

<369> Section 3.1.4.3.1: In Windows, the command codes JOB_CONTROL_CANCEL (0x00000003) and JOB_CONTROL_DELETE (0x00000005) are equivalent. The redundancy is caused by historical reasons, but both must be supported.

<370> Section 3.1.4.3.1: For JOB_CONTROL_CANCEL and JOB_CONTROL_DELETE requests, the server verifies that the client has generic DELETE permission. For all other requests, the server verifies that the client has JOB_ACCESS_ADMINISTER permission for the job. For a job position change

request, the server additionally verifies that the handle is a printer or server object and has been opened with PRINTER_ACCESS_ADMINISTER requested.

<371> Section 3.1.4.4.1: The Windows server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<372> Section 3.1.4.4.1: These validation steps are performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<373> Section 3.1.4.4.1: These validation steps are performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<374> Section 3.1.4.4.1: When a print client adds a printer driver to a Windows print server using RpcAddPrinterDriver, the print server sets the Boolean values representing each of the printer driver's attributes to a Boolean value of FALSE, indicating that the printer driver does not have any of these attributes.

<375> Section 3.1.4.4.2: In Windows, if the string contains "All" or "AllCluster", all printer drivers for all environments must be enumerated.

<376> Section 3.1.4.4.2: The Windows server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<377> Section 3.1.4.4.2: This validation step is performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<378> Section 3.1.4.4.2: This validation step is performed on the following Windows versions:

- Windows 8

- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<379> Section 3.1.4.4.3: Windows servers complete the request only if the specified environment is "Windows 4.0" and **pszzPreviousNames** contains the name of a printer driver that is installed on the server for the "Windows 4.0" environment; otherwise, the server returns ERROR_UNKNOWN_PRINTER_DRIVER. For other environments, the clients can use RpcGetPrinterDriver2 when the server supports this method.

<380> Section 3.1.4.4.4: The Windows server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<381> Section 3.1.4.4.5: In Windows, the server checks that the client user has the SERVER_ACCESS_ADMINISTER permission.

<382> Section 3.1.4.4.6: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<383> Section 3.1.4.4.6: The following table shows the unsigned 32-bit major operating system version number that is used by Windows clients and servers.

Major version	Meaning
0x00000004	The operating system is Windows 95, Windows NT 4.0, Windows 98, or Windows Millennium Edition.
0x00000005	The operating system is Windows 2000, Windows XP, Windows Server 2003, or Windows Server 2003 R2 operating system.
0x00000006	The operating system is Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.

<384> Section 3.1.4.4.6: The following table shows the unsigned 32-bit minor operating system version number that is used by Windows Server 2008 clients and servers.

Minor version	Meaning
0x00000000	The operating system is Windows 95, Windows NT 4.0, Windows 2000, Windows Vista, or Windows Server 2008.
0x00000001	The operating system is Windows XP, Windows 7, or Windows Server 2008 R2.
0x00000002	The operating system is Windows XP Professional x64 Edition, Windows Server 2003, Windows Server 2003 R2, Windows 8, or Windows Server 2012.
0x00000003	The operating system is Windows 8.1, Windows Server 2012 R2, Windows 10, or Windows Server 2016 Technical Preview.
0x0000000A	The operating system is Windows 98.
0x0000005A	The operating system is Windows Millennium Edition.

<385> Section 3.1.4.4.6: *pdwServerMaxVersion* and *pdwServerMinVersion* are ignored by the Windows print server, and therefore no values are returned. However, the caller is expected to submit valid pointer values for *pdwServerMaxVersion* and *pdwServerMinVersion* in order to ensure that the call can be correctly received by the server.

Note If both *dwClientMajorVersion* and *dwClientMinorVersion* are set to 0xFFFFFFFF, the print server will return printer driver information for the printer driver version matching the operating system version on which the print server is running.

<386> Section 3.1.4.4.6: This validation step is performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<387> Section 3.1.4.4.7: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<388> Section 3.1.4.4.7: Windows sets this parameter to one of the following values.

Value	Meaning
0x00000001	Windows NT 3.51 user-mode printer drivers
0x00000002	Windows NT 4.0 user-mode printer drivers
0x00000003	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 user-mode printer drivers.
0x00000004	Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, and Windows Server 2016 Technical Preview user-mode printer drivers.

<389> Section 3.1.4.4.7: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER (section 2.2.3.1) permission.

<390> Section 3.1.4.4.8: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008

- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<391> Section 3.1.4.4.8: Only in Windows XP and Windows Server 2003, as described in 3.1.1, the Windows server maintains a warned list of printer drivers. Drivers in this list can only be added to the server if the **APD_INSTALL_WARNED_DRIVER** bit is set. If a client attempts to add a printer driver that is in this list and the bit is not set, the server returns the ERROR_PRINTER_DRIVER_WARNED error code. If a client attempts to add a printer driver that is in this list and the bit is set, the server attempts to add the printer driver as described in 3.1.4.4.8. On all other versions of Windows, this list does not exist, and the flag is ignored.

On Windows XP and Windows Server 2003, the Windows client detects the ERROR_PRINTER_DRIVER_WARNED error code returned from the RpcAddPrinterDriverEx method and asks the user whether to continue adding the printer driver. If the user continues, the client sets the **APD_INSTALL_WARNED_DRIVER** bit in the **dwFileCopyFlags** field and calls the method again. On Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the Windows client detects the ERROR_PRINTER_DRIVER_WARNED error code and informs the user that the driver could not be installed. Clients with these versions of the operating system do not use the **APD_INSTALL_WARNED_DRIVER** flag.

<392> Section 3.1.4.4.8: In Windows, if this bit is set, ERROR_PRINTER_DRIVER_BLOCKED is returned. if this bit is not set, ERROR_UNKNOWN_PRINTER_DRIVER is returned.

<393> Section 3.1.4.4.8: In Windows, the server checks that the client user has the SERVER_ACCESS_ADMINISTER permission.

<394> Section 3.1.4.4.8: These validation steps are performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<395> Section 3.1.4.4.8: These validation steps are performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1

- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<396> Section 3.1.4.4.8: When a print client adds a printer driver to a

<397> Section 3.1.4.4.9: This method is used remotely only if the client is at least Windows 7 operating system and the server is at least Windows Server 2008 R2 operating system.

<398> Section 3.1.4.4.9: All Windows versions: The IDs are the GUIDString representations of 128-bit GUIDs.

<399> Section 3.1.4.4.10: This method is used remotely only if the client is at least Windows 7 operating system and the server is at least Windows Server 2008 R2 operating system.

<400> Section 3.1.4.4.10: All Windows versions: The *pszLanguage* string is specified using the identifiers specified for the Locale Name in [MSDN-MUI].

<401> Section 3.1.4.4.10: All Windows versions: *pszDriverPackageCab* points to a string containing the path name of a cabinet file for the driver package. For more information, see [MSDN-CAB].

<402> Section 3.1.4.4.10: All Windows versions: If the parameter is zero, Windows fills in the variable pointed to by *pcchRequiredSize* with the valid size.

<403> Section 3.1.4.5.1: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<404> Section 3.1.4.5.2: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<405> Section 3.1.4.5.4: The Windows server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<406> Section 3.1.4.6.1: The Windows server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<407> Section 3.1.4.6.2: The RpcDeletePort method is only called over the wire by Windows NT 4.0 operating system clients.

<408> Section 3.1.4.6.2: Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, and Windows Server 2012 R2 check that the client user has SERVER_ACCESS_ENUMERATE permission. All other Windows versions check that the client user has SERVER_ACCESS_ADMINISTER permission.

<409> Section 3.1.4.6.3: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<410> Section 3.1.4.6.3: In Windows, the server checks that the client user has the SERVER_ACCESS_ADMINISTER permission.

<411> Section 3.1.4.6.4: This feature is supported on the following Windows versions:

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<412> Section 3.1.4.6.4: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<413> Section 3.1.4.6.5: In Windows, the actions and data retrieval supported by RpcXcvData are performed by executing the XcvDataPort method on the port monitor module referenced by the input handle. For more information about monitor module methods, see section 3.1.4.11. This feature is supported on the following Windows versions:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<414> Section 3.1.4.6.5: For Windows–specific behavior, see section 3.1.4.11.

<415> Section 3.1.4.6.5: These validation steps are performed on the following Windows versions:

- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<416> Section 3.1.4.7.1: In Windows, the server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<417> Section 3.1.4.7.1: On Windows servers, the monitors supporting these methods are LOCALMON, LPRMON, TCPMON, **USBMON**, and WSDMON.

<418> Section 3.1.4.7.2: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<419> Section 3.1.4.7.3: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<420> Section 3.1.4.8.1: This validation step is performed on the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<421> Section 3.1.4.8.1: In Windows, the server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<422> Section 3.1.4.8.2: The Windows server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<423> Section 3.1.4.8.3: In Windows, the server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<424> Section 3.1.4.8.4: The Windows server checks that the client user has SERVER_ACCESS_ADMINISTER permission.

<425> Section 3.1.4.8.5: In Windows, the server checks that the client user has SERVER_ACCESS_ENUMERATE permission.

<426> Section 3.1.4.9.6: For a job object, the Windows server verifies that the calling client has been granted an access value that includes the **JOB_READ** permission (section 2.2.3.1). For a port object, the Windows server verifies that the calling client has, for the currently printing job on that port, an access value that includes the **JOB_READ** permission.

<427> Section 3.1.4.9.6: The Local Port monitor in Windows supports reading from COM ports.

<428> Section 3.1.4.9.8: This feature is supported on the following Windows version:

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista

- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<429> Section 3.1.4.10.1: Windows waits for up to 600 seconds

<430> Section 3.1.4.10.2: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<431> Section 3.1.4.10.3: This method is called only by Windows NT 3.5 clients.

<432> Section 3.1.4.10.3: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<433> Section 3.1.4.10.3: Windows clients generate an arbitrary unique value for each known printer.

<434> Section 3.1.4.10.4: This method is called only by Windows NT 3.5 clients.

<435> Section 3.1.4.10.4: This feature is supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7

- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<436> Section 3.1.4.10.4: Windows clients generate an arbitrary unique value for each known printer.

<437> Section 3.1.4.10.5: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<438> Section 3.1.4.10.5: Windows clients pass in monotonically increasing values for the *dwColor* parameter in calls to this method. Windows servers use these values in calls to RpcRouterReplyPrinterEx (section 3.2.4.1.4). Clients determine the most recent notification data returned by the server by comparing the values of this parameter. Clients discard the notification data received in a call to RpcRouterReplyPrinterEx if the *dwColor* value is different from the value that was received most recently from RpcRouterRefreshPrinterChangeNotification.

Windows uses this mechanism because of the possible ordering reversal of calls caused by network latency.

<439> Section 3.1.4.11: The Windows port monitors support these functions in the following cases:

- In Windows NT 3.5 and Windows NT 4.0, LOCALMON supports AddPortEx, AddPort, ConfigurePort, and DeletePort.
- In the following Windows versions, LOCALMON supports only AddPortEx, and TCPMON supports AddPortEx, AddPort, ConfigurePort, and DeletePort.
  - Windows 2000
  - Windows XP
  - Windows Server 2003
  - Windows Vista
  - Windows Server 2008
  - Windows 7
  - Windows Server 2008 R2
  - Windows 8
  - Windows Server 2012
  - Windows 8.1
  - Windows Server 2012 R2
  - Windows 10
  - Windows Server 2016 Technical Preview

<440> Section 3.1.4.11: The Windows operating system provides port monitor modules in the form of executable dynamic-link libraries (DLLs). Examples are:

- **localmon.dll**, which manages local serial ("COM") and parallel ("LPT") ports on a Windows machine (section 3.1.4.11.1).
- **lprmon.dll**, which allows a Windows print server to send print jobs to machines that support **UNIX** print-server functions (section 3.1.4.11.2).
- **tcpmon.dll**, which manages standard TCP/IP ports on a Windows machine (section 3.1.4.11.3).
- **usbmon.dll**, which manages local USB ports on a Windows machine.
- **wsdmon.dll**, which manages Web Services for Devices (WSD) ports (section 3.1.4.11.4), except on Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, Windows 98, Windows 2000, Windows Millennium Edition, Windows XP, and Windows Server 2003.

All the functions that LOCALMON exports have been incorporated into **localspl.dll** in Windows, except for the following versions.

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51

- Windows NT 4.0
- Windows 98

Also in Windows, except for the preceding versions, port monitors are divided into two DLLs, a port monitor server DLL and a port monitor user interface DLL.

<441> Section 3.1.4.11.1: For the following versions of Windows, LOCALMON is packaged inside another executable component called **localspl.dll**; the presentation is captured in a separate monitor module. The user interface module corresponding to LOCALMON is called localui.

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<442> Section 3.1.4.11.2: The LPRMON monitor module is supported in the following versions of Windows. The user interface module corresponding to LPRMON is called **lprmonui**.

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<443> Section 3.1.4.11.3: The TCPMON monitor module is supported for the following versions of Windows. The user interface module corresponding to TCPMON is called **tcpmonui**.

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<444> Section 3.1.4.11.3: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<445> Section 3.1.4.11.3: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1

- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<446> Section 3.1.4.11.3: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<447> Section 3.1.4.11.3: This feature is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<448> Section 3.1.4.11.4: This monitor module is supported on the following Windows versions:

- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<449> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<450> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<451> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<452> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<453> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<454> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<455> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows Vista SP1
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<456> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<457> Section 3.1.4.11.4: This command is only supported on stand-alone servers. The required value for the *pInputData* parameter is a non-NULL pointer to a string specifying the **GlobalID** of the WSD endpoint. The required value for the *pOutputData* parameter is a non-NULL pointer to a buffer that is required to receive a string identifying the new port name if a WSD Printer Service is found; otherwise, ERROR_PRINTER_NOT_FOUND is returned, as specified in [MS-ERREF].

<458> Section 3.1.4.11.4: This feature is supported on the following Windows versions:

- Windows 10
- Windows Server 2016 Technical Preview

<459> Section 3.1.4.12.1: The **RpcGetJobNamedPropertyValue** method is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<460> Section 3.1.4.12.2: The **RpcSetJobNamedProperty** method is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<461> Section 3.1.4.12.3: The **RpcDeleteJobNamedProperty** method is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<462> Section 3.1.4.12.4: The **RpcEnumJobNamedProperties** method is supported by the following Windows versions:

- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10

- Windows Server 2016 Technical Preview

<463> Section 3.1.4.13.1: The RpcLogJobInfoForBranchOffice (section 3.1.4.13.1) method is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<464> Section 3.2.1: Branch office print remote logging is supported on the following Windows versions:

- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<465> Section 3.2.1: In Windows, this data element is stored in the print spooler.

<466> Section 3.2.4: The Windows implementation ignores errors and passes them back to the invoker.

<467> Section 3.2.4.1.1: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2

- Windows 10
- Windows Server 2016 Technical Preview

<468> Section 3.2.4.1.1: The Windows server calls RpcReplyOpenPrinter during the processing of a call to RpcRemoteFindFirstPrinterChangeNotificationEx. The return value indicates the success of that processing.

<469> Section 3.2.4.1.2: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<470> Section 3.2.4.1.3: This feature is supported on the following Windows versions:

- Windows NT 3.5
- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista

- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<471> Section 3.2.4.1.4: This feature is supported on the following Windows versions:

- Windows NT 3.51
- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows XP SP1
- Windows Vista
- Windows Server 2008
- Windows Vista SP1
- Windows 7
- Windows Server 2008 R2
- Windows 8
- Windows Server 2012
- Windows 8.1
- Windows Server 2012 R2
- Windows 10
- Windows Server 2016 Technical Preview

<472> Section 3.2.4.2.3: The Windows system will select a number of jobs to be 0xFFFFFFFF to obtain the full list in a single operation.

<473> Section 3.2.4.2.5: Windows clients only enumerate their shared printers if no directory services are available in the domain.

<474> Section 3.2.4.2.5: Windows clients enumerate their shared printers up to a default of three print servers and a default of two workstations and one additional server per 32 print servers found. The values can be changed via policy settings.

<475> Section 3.2.4.2.6: A Windows client uses RpcOpenPrinterEx to open a handle for the server object. Then the Windows client uses RpcGetPrinterData with the value names "Architecture", "MajorVersion", and "OSVersion" to obtain information about the server's environment. This information is used to select an appropriate printer driver. The client subsequently closes the server object handle using RpcClosePrinter. The Windows-based client also uses RpcEnumPorts to populate a list with ports available on the server from which the end-user can select the port for the new printer. The Windows-based client uses RpcEnumMonitors to determine a port monitor for the new printer. The Windows-based client uses RpcGetPrinterDriverDirectory to determine the destination directory in case the new printer requires that a printer driver be installed.

<476> Section 5.2: Windows print server follows a security model in which the print server, print queues, and print job are securable resources. Each of these resources has a security descriptor that is associated with it. The **SD** contains the security information that is associated with a resource on the print server. The print server checks the client access to resources by comparing the security information that is associated with the caller against the SD of the resource. SDs are specified by the SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6).

Each RPC client has associated with it an access token, which contains the **security identifier (SID)** of the user making the RPC call. The SD identifies the owner of the printing resource and contains a discretionary access control list (DACL). The DACL contains access control entries (ACEs), which specify the SID of a user or group of users, and whether access rights are to be allowed, denied, or audited. For resources on a print server, the ACEs specify operations including: print, manage printers, and manage documents in a print queue.

The SD that is associated with the print server or print queue controls the creation of the context handle (PRINTER_HANDLE section 2.2.1.1.4) and the outcome of operations that use the handle. These operations range from printing and job management to listening for notifications.

The SDs for a Windows-based print server are used to control the creation and deletion of print queues on the server; the installation of print system components, including printer drivers, print processors, and port monitors; and the enumeration of resources on the server. A server SD is not write-accessible to callers. In addition to being used to control callers' access to resources, the server SD is also used as the "parent" in the creation of the print queue's SD.

Note: The SD of the Windows-based print server is different from the SD that is applied on the named pipe of the spool. The SD of the spool's named pipe controls the RPC client's access to make RPC calls to the print server. The SD of the Windows-based print server is used to control the caller's permissions to perform various operations on the print server.

The SD of the print queue controls the setting of print queue properties, such as the port and printer driver that are used for printing; the rendering and device settings; or the sharing or security parameters. The printer SD allows auditing operations such as printing; managing printers and documents; reading and changing permissions; and taking ownership.

Each print job has an associated SD, which is created by using the SD of the print queue as parent. The user who submitted the document for printing is the owner of the print job and has permission to manage the print job during its lifetime.

When a PRINTER_HANDLE is opened for a specific printing resource, the caller must specify the access that is needed for subsequent operations, including printer or server administration; printing on a printer or print server; and reading, writing, or administrating a print job. If the caller has the specified permissions, the print handle is created and can be used for subsequent calls.

In addition to handle-based operations, the SD is used for access checks when enumerations, printer driver package installation, or other non-handle-based operations are performed.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W