[MS-PAR]:
Print System Asynchronous Remote Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.0.1	Editorial	Revised and edited the technical content.
07/20/2007	1.0.2	Editorial	Revised and edited the technical content.
08/10/2007	1.0.3	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
09/28/2007	1.1	Minor	Updated the technical content.
10/23/2007	1.2	Minor	Updated the technical content.
11/30/2007	1.2.1	Editorial	Revised and edited the technical content.
01/25/2008	1.2.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Print System Asynchronous Remote Protocol, which defines the communication of **print job** processing and **print system** management information between a **print client** and any **print server**. The Print System Asynchronous Remote Protocol is built on the **remote procedure call (RPC)** protocol and can be used as an enhanced replacement for the Print System Remote Protocol, as specified in [MS-RPRN].

The Print System Asynchronous Remote Protocol is designed for use by asynchronous clients whose implementations permit them to continue execution without waiting for an RPC method call to return.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Access Control Entry (ACE)
Authentication Level
Binary Large Object (BLOB)
Color Profile
Device
Discretionary Access Control List (DACL)
Domain
Driver Package
Driver Store
Endpoint
Enhanced Metafile Format (EMF)
Failover
GUIDString
INF file
Interface Definition Language (IDL)
Little-Endian
Network Data Representation (NDR)
Opnum
Page Description Language (PDL)
Principal Name
Print Client
Print Queue
Print Server
Print System
Registry
Remote Procedure Call (RPC)
RPC Dynamic Endpoint
RPC Endpoint
Security Identifier (SID)
Security Provider
Unicode
Universally Unique Identifier (UUID) or Globally Unique Identifier (GUID)
UTF-16LE
Well-Known Endpoint

The following terms are specific to this document:

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

Bidi: See **Bidirectional**.

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

CAB File: See **Cabinet File**.

Cabinet File: A file that has the suffix .cab and that acts as a container for other files. It serves as a compressed archive for a group of files. For more information, including the format of **.cab files**, see [MSDN-CAB].

Color Matching: The conversion of a color that is sent from its original **color space** to its visually closest color in the destination **color space**. See also: Image Color Management.

Color Space: A mapping of color components to a multidimensional coordinate system. The number of dimensions is generally two, three, or four. For example, if colors are expressed as a combination of the three components red, green, and blue, a three-dimensional space is sufficient to describe all possible colors. Grayscales, however, can be mapped to a two-dimensional **color space**. And if transparency is considered one of the components of an RGB color, four dimensions are appropriate.

Core Printer Driver: A printer driver that other printer drivers depend on. In Windows, this term includes the Unidrv and Pscript5 printer drivers.

Information Context: A special-purpose printer object that can only be used to obtain information about the fonts that are supported by a printer.

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

Multistring: An array of null-terminated, 16-bit **Unicode** strings (**UTF-16LE** encoded), with one additional null after the final **string**, which is null-terminated in it own right. Therefore, there are actually two nulls at the end of a **multistring** structure.

Multisz: A data type that defines a **multistring**.

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**.

Print Job: The graphical output data that is sent by a particular application, or by user request, in order to be processed for printing on a print **device**. The print job can contain **PDL** data or an intermediate representation of the output data, which is translated to **PDL** data in the process of printing.

Print Processor: A **plug-in** that runs on a **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 Windows-specific and not required by the protocol.

Printer Driver: A software component that provides an interface between the operating system and the print **device**. It is responsible for processing the application data into a **PDL** that can be interpreted by the print **device**.

Printer Form: A named definition of a physical sheet of paper, including dimensions and printable area.

RAW Format: **PDL** data that can be sent to the **device** without further processing.

Spool File: A representation of print job data that can be processed by a **print processor**. The term file is used in a generic fashion and does not mandate a specific implementation choice. **Print servers** may choose to keep this temporary data in another location, such as in main memory.

Spool File Format: The specific representation that is used in an instance of a spool file. Common examples for spool file formats are **enhanced metafile format (EMF)** Spool Format and XML Paper Specification. For more information, see [MSDN-SPOOL] and [MSDN-XMLP].

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

1.2 References

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. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-RPRN] Microsoft Corporation, "Print System Remote Protocol Specification", June 2007.

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2.0 Protocol Specification", July 2007.

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

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

1.2.2 Informative References

[DEVMODE] Microsoft Corporation, "DEVMODE", http://msdn2.microsoft.com/en-us/library/ms535771.aspx

[MSDN-AUTHN] Microsoft Corporation, "Authentication-Service Constants", http://msdn2.microsoft.com/en-us/library/aa373556.aspx

[MSDN-CAB] Microsoft Corporation, "Microsoft Cabinet SDK", March 1997, http://msdn2.microsoft.com/en-us/library/ms974336.aspx

[MSDN-FONTS] Microsoft Corporation, "About Fonts", http://msdn2.microsoft.com/en-us/library/ms533976.aspx

[MSDN-MUI] Microsoft Corporation, "Locale Identifier Constants and Strings", http://msdn2.microsoft.com/en-us/library/ms776260.aspx

[MSDN-SPOOL] Microsoft Corporation, "Print Spooler Components", http://msdn2.microsoft.com/en-us/library/ms802196.aspx

[MSDN-UDP] Microsoft Corporation, "Using Device Profiles with ICM2", http://msdn2.microsoft.com/en-us/library/ms536847.aspx

[MSDN-UINF] Microsoft Corporation, "Using INF Files", http://msdn2.microsoft.com/en-us/library/Aa741213.aspx

[MSDN-XMLP] Microsoft Corporation, Watson, B., "A First Look at APIs For Creating XML Paper Specification Documents", http://msdn.microsoft.com//msdnmag/issues/06/01/xmlpaperspecification/default.aspx

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

1.3 Protocol Overview (Synopsis)

The Print System Asynchronous 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 adding 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 [MS-RPRN] 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.

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 diagram illustrates this interaction using the scenario of adding a new printer:

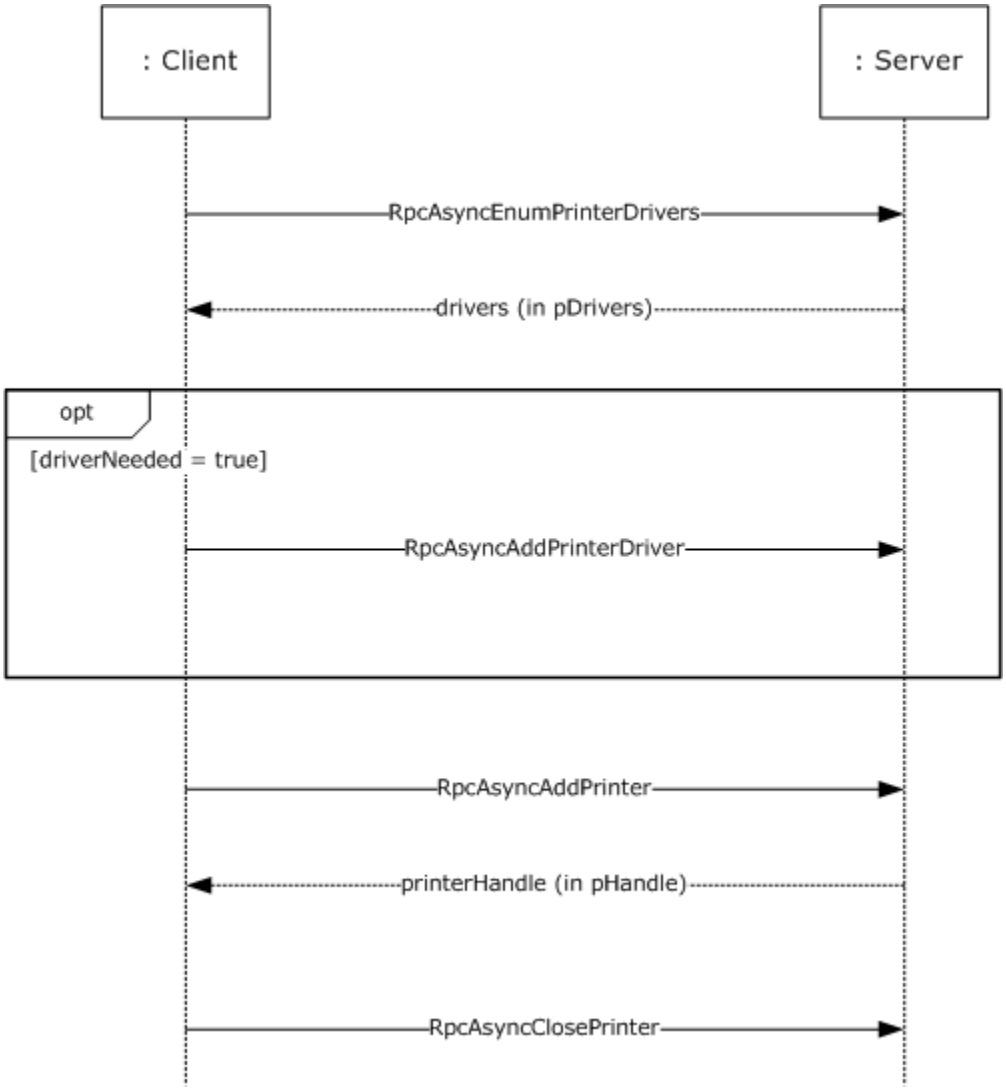

Figure 1: Adding a new printer

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 server.

In one configuration, a client uses a printer driver that is installed on the client computer in order 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 as well as 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.<1>

The following diagram illustrates this interaction.

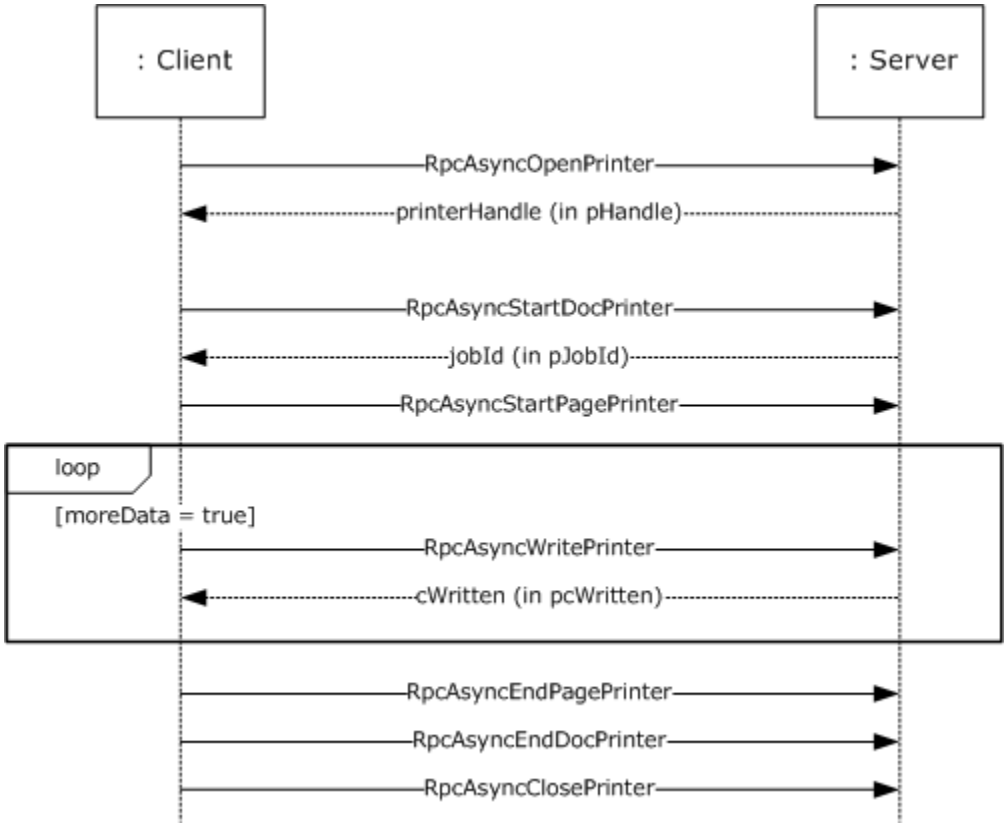

Figure 2: Communication of print job data

1.3.3 Notification of Print System Changes

This protocol also provides the methods that the 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.

A client can register with the print server to receive notifications of changes in a print queue. As long as the client is connected to the print server, it can poll the print server for the current status after it receives a notification.

The application calls **RpcAsyncGetRemoteNotifications (section 3.1.4.9.4)** to receive notification that something has changed. The server suspends the processing of this call until there are new notifications available on the print server, at which time, the server prepares a response and returns from the outstanding **RpcAsyncGetRemoteNotifications** call.

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.

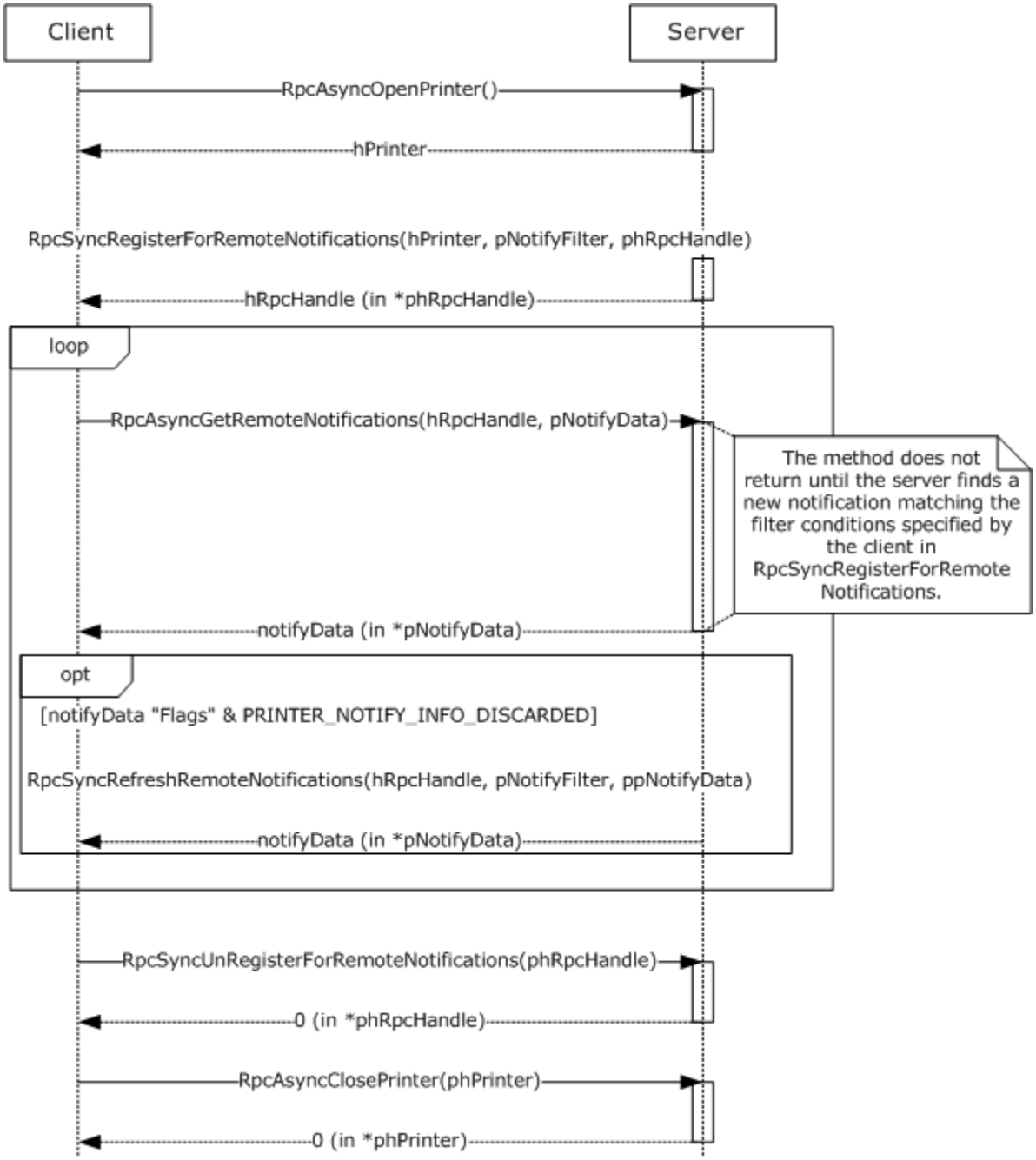

Figure 3: Notification of print system changes

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 client can query this change identifier by using the

RpcAsyncGetPrinterData method that is defined in this protocol and calling it with the **pValueName** parameter pointing to the string "ChangeID".

When a disconnected 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".

1.4 Relationship to Other Protocols

The Print System Asynchronous Remote Protocol is dependent on remote procedure call (RPC), specified in [MS-RPCE], and it references the Print System Remote Protocol, as specified in [MS-RPRN]. Many of the data structures that are used in the Print System Asynchronous Remote Protocol are specified in [MS-RPRN] sections 2.2.1 and 2.2.2.

This protocol does not specify methods for file transfer between client and server. The [MS-SMB2] protocol SHOULD be used for any file transfer between client and server, such as driver download operations.

No protocols are dependent on the Print System Asynchronous Remote Protocol.

1.5 Prerequisites/Preconditions

The Print System Asynchronous Remote Protocol is an RPC interface, and therefore, it has the prerequisites that are specified in [MS-RPCE] section 1.5, as common to RPC interfaces.

A print client MUST obtain the name of a print server that supports the Print System Asynchronous Remote Protocol before this protocol is invoked. How a client does this is not addressed in this specification.

1.6 Applicability Statement

The Print System Asynchronous 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; to print device sharing between computers; to an enterprise-use scenario that has multiple print server that are employed in a cluster configuration and client configurations that are managed by a directory access protocol.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports**: The Print System Asynchronous Remote Protocol uses RPC over TCP/IP only, as specified in section 2.1.
- **Protocol Versions**: This protocol has only one interface version. The use of these methods is specified in section 3.1.4.

  Versioning of data structures defined by the protocol is controlled by using a level value in all CONTAINER data structures. Typically, levels are 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. Levels and containers are defined in [MS-RPRN] section 2.2.1. The level value is also a parameter to some RPC methods. If a protocol method fails because the information level is unsupported, the client SHOULD try with a lower level.

- **Security and Authentication Methods**: This protocol uses **SPNEGO** and RPC packet **authentication levels** for security and authentication, as specified in section 2.1. The parameters that are sent from client to server include a "token" that defines user credentials. The print server, to verify access permissions, MUST process this token. The format of the token MUST be as specified in [MS-RPCE] section 2.2.2.12.
- **Localization**: The protocol does not contain locale-dependent information.
- **Return Values**: The methods comprising this RPC interface MUST return zero to indicate successful completion or a nonzero value to indicate failure, except where specifically described. Unless otherwise specified, a server-side implementation of this protocol SHOULD choose any nonzero Win32 error value to signify an error condition, as discussed in section 1.8. Unless otherwise specified, the client side of the Print System Asynchronous Remote Protocol MUST NOT interpret returned error codes. Unless otherwise specified, the client side of the Print System Asynchronous Remote Protocol MUST return only error codes to the invoking application without taking any protocol action.
- **Capability Negotiation**: Functional negotiation is supported through the use of container levels, as specified in [MS-RPRN] section 2.2.1. On connection to a server, the client requests a level. If the information level is a level that is supported by the server, the server MUST process the request. Otherwise, the server MUST return an error to the client, and the client SHOULD repeat the request with a lower level.

1.8 Vendor-Extensible Fields

The methods defined in the Print System Asynchronous Remote Protocol specify either the DWORD or HRESULT data type for return values.

DWORD return values are Win32 error codes that are taken from the Windows error number space, as specified in [MS-ERREF] section 4.2. Implementers MUST 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 4.1. Implementers MAY<2> choose their own HRESULT values, but the C bit (0x20000000) MUST be set, indicating that it is a customer code.

1.9 Standards Assignments

The Print System Asynchronous Remote Protocol MUST use the following private assignments:

Parameter	Value	Reference
UUID	{ 76F03F96-CDFD-44fc-A22C-64950A001209 }	Section 2.1 and [C706] Appendix A

2 Messages

The following sections specify how the Print System Asynchronous Remote Protocol messages are transported and also specify the Print System Asynchronous Remote Protocol common data types.

2.1 Transport

The Print System Asynchronous Remote Protocol specifies the following transport requirements:

- This protocol MUST use the transport RPC over TCP/IP, as specified in [MS-RPCE] section 2.1.1.1.
- This protocol MUST use **RPC dynamic endpoints**, as specified in [C706] section 4.
- This protocol MUST use the **UUID** specified in section 1.9.
- A server of this protocol MUST use:
  - A **SPNEGO** **security provider**, as specified in [MS-RPCE] section 3; and
  - The default server **principal name** for the security provider, which is the authentication-service constant **RPC_C_AUTHN_GSS_NEGOTIATE**. For information concerning Windows authentication-service constants, see [MSDN-AUTHN].
- A client of this protocol MUST use:
  - A **SPNEGO** security provider, as specified in [MS-RPCE] section 3;
  - A principal name constructed by appending the name of the print server to the string "host/"; and
  - Packet authentication level, as specified in [MS-RPCE] section 3.

For more information about **SPNEGO**, see [MS-SPNG].

2.2 Common Data Types

In addition to the RPC base types and definitions that are specified in [C706] and [MS-DTYP], additional data types are defined in this section.

The Print System Asynchronous Remote Protocol MUST indicate to the RPC runtime that it is to support both the **NDR** and NDR64 transfer syntaxes and provide a negotiation mechanism for determining which transfer syntax is 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 Asynchronous Remote Protocol employs a combination of the following data representations:

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

The following statements apply to the entire specification, unless noted otherwise:

- All **strings** that are defined in this protocol MUST consist of characters encoded in **Unicode UTF-16LE**, and MUST be null terminated. Each UTF-16 code point in a **string**, including null terminators, MUST occupy 16 bits. The details of these strings are as specified in [RFC2781] section 2.1.
- A list of **strings** is referred to as a multisz in this specification. 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 method parameters or structure members specifying the number of characters in a **string** or multisz specify the number of characters including terminating null characters.
- All method parameters or structure members specifying the number of bytes in buffer containing a **string** or multisz specify the number of bytes including terminating null characters.
- All data types made up of more than a single byte MUST be treated as specified in **little-endian** byte order.
- The term "null" means "a null pointer", and "zero" means the number 0.
- All method parameters or structure members that specify the size of a buffer that is 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.

2.2.1 EPrintPropertyType

The **EPrintPropertyType** enumeration defines the data types for different printing properties.

```
typedef  enum
{
  kPropertyTypeString = 1,
  kPropertyTypeInt32,
  kPropertyTypeInt64,
  kPropertyTypeByte,
  kPropertyTypeTime,
  kPropertyTypeDevMode,
  kPropertyTypeSD,
  kPropertyTypeNotificationReply,
  kPropertyTypeNotificationOptions
} EPrintPropertyType;
```

kPropertyTypeString: The data type is **string**.

kPropertyTypeInt32: The data type is a 32-bit signed integer.

kPropertyTypeInt64: The data type is a 64-bit signed integer.

kPropertyTypeByte: The data type is a **BYTE**.

kPropertyTypeTime: The data type is **SYSTEMTIME_CONTAINER**, as specified in [MS-RPRN] section **2.2.1.2.16**.

kPropertyTypeDevMode: The data type is **DEVMODE_CONTAINER**, as specified in [MS-RPRN] section **2.2.1.2.1**.

kPropertyTypeSD: The data type is **SECURITY_CONTAINER**, as specified in [MS-RPRN] section **2.2.1.2.13**.

kPropertyTypeNotificationReply: The data type is **NOTIFY_REPLY_CONTAINER**, as specified in section 2.2.7.

kPropertyTypeNotificationOptions: The data type is **NOTIFY_OPTIONS_CONTAINER**, as specified in section 2.2.6.

2.2.2 RpcPrintPropertyValue

The **RpcPrintPropertyValue** structure specifies a data type and its value. Data types are members of the enumeration **EPrintPropertyType**, specified in section 2.2.1.

```
typedef struct {
  EPrintPropertyType ePropertyType;
  [switch_type(EPrintPropertyType), switch_is(ePropertyType)]
    union {
    [case(kPropertyTypeString)]
      wchar_t* propertyString;
    [case(kPropertyTypeInt32)]
      long propertyInt32;
    [case(kPropertyTypeInt64)]
      __int64 propertyInt64;
    [case(kPropertyTypeByte)]
      BYTE propertyByte;
    [case(kPropertyTypeDevMode)]
      DEVMODE_CONTAINER propertyDevModeContainer;
    [case(kPropertyTypeTime)]
      SYSTEMTIME_CONTAINER propertyTimeContainer;
    [case(kPropertyTypeSD)]
      SECURITY_CONTAINER propertySDContainer;
    [case(kPropertyTypeNotificationReply)]
      NOTIFY_REPLY_CONTAINER propertyReplyContainer;
    [case(kPropertyTypeNotificationOptions)]
      NOTIFY_OPTIONS_CONTAINER propertyOptionsContainer;
  } value;
} RpcPrintPropertyValue;
```

ePropertyType: This member MUST be a value from the **EPrintPropertyType** enumeration and MUST specify the data type of the variable.

value: This member MUST specify an information structure that corresponds to the type of property specified by the **ePropertyType** parameter. Information containers and structures are defined in [MS-RPRN] sections 2.2.1 and 2.2.2.

propertyString: Value is a **string**.

propertyInt32: Value is a 32-bit signed integer.

propertyInt64: Value is a 64-bit signed integer.

propertyByte: Value is a **BYTE**.

propertyDevModeContainer: Value is a **DEVMODE_CONTAINER**, specified in [MS-RPRN] section **2.2.1.2.1**.

propertyTimeContainer: Value is a **SYSTEMTIME_CONTAINER**, specified in [MS-RPRN] section **2.2.1.2.16**.

propertySDContainer: Value is a **SECURITY_CONTAINER**, specified in [MS-RPRN] section **2.2.1.2.13**.

propertyReplyContainer: Value is a **NOTIFY_REPLY_CONTAINER**, specified in section 2.2.7.

propertyOptionsContainer: Value is a **NOTIFY_OPTIONS_CONTAINER**, specified in section 2.2.6.

2.2.3 RpcPrintNamedProperty

The **RpcPrintNamedProperty** structure MUST specify a name/typed-value pair that defines a single property.

```
typedef struct {
  wchar_t* propertyName;
  RpcPrintPropertyValue propertyValue;
} RpcPrintNamedProperty;
```

propertyName: MUST point to a string that MUST specify the name of the property.

propertyValue: MUST specify the value of the property.

When used as an input parameter, the **propertyName** and the **ePropertyType** member of **propertyValue** MUST be one of the following pairs:

propertyName	**propertyValue**	
	ePropertyType	**value**
"RemoteNotifyFilter Flags"	kPropertyTypeInt32	Same as **fdwFlags**, as specified in [MS-RPRN] section **3.1.4.10.4**.
"RemoteNotifyFilter Options"	kPropertyTypeInt32	Same as **fdwOptions**, as specified in [MS-RPRN] section **3.1.4.10.4**.
"RemoteNotifyFilter NotifyOptions"	kPropertyTypeNotificationOptions	Same as **pOptions**, as specified in [MS-RPRN] section **3.1.4.10.4**.
"RemoteNotifyFilter Color"	kPropertyTypeInt32	Same as **dwColor**, as specified in [MS-RPRN] section **3.1.4.10.5**.

When used as an output parameter, the **propertyName** and the **ePropertyType** member of **propertyValue** MUST be one of the following pairs:

propertyName	propertyValue	
	ePropertyType	value
"RemoteNotifyData Flags"	kPropertyTypeInt32	Same as **fdwFlags**, as specified in [MS-RPRN] section **3.2.4.1.4**.
"RemoteNotifyData Info"	kPropertyTypeNotificationReply	Same as **Reply.pInfo**, as specified in [MS-RPRN] section **3.2.4.1.4**.
"RemoteNotifyData Color"	kPropertyTypeInt32	Same as **dwColor**, as specified in [MS-RPRN] section **3.2.4.1.4**.

2.2.4 RpcPrintPropertiesCollection

The **RpcPrintPropertiesCollection** structure MUST hold a collection of name/typed-value pairs.

```
typedef struct {
  [range(0, kMaxNumberOfJobProperties)]
    DWORD numberOfProperties;
  [size_is(numberOfProperties), unique]
    RpcPrintNamedProperty* propertiesCollection;
} RpcPrintPropertiesCollection;
```

numberOfProperties: MUST contain a value that specifies the number of properties in the collection. This number MUST NOT exceed kMaxNumberOfJobProperties, which is 50.

propertiesCollection: MUST be a pointer to an array of **RpcPrintNamedProperty** values.

When used as input to specify notification filter settings, the following properties MUST be present in the collection pointed to by the **propertiesCollection** member:

- "RemoteNotifyFilter Flags"
- "RemoteNotifyFilter Options"
- "RemoteNotifyFilter NotifyOptions"
- "RemoteNotifyFilter Color"

When used as output to return notification data, the following properties MUST be present in the collection pointed to by the **propertiesCollection** member:

- "RemoteNotifyData Flags"
- "RemoteNotifyData Info"
- "RemoteNotifyData Color"

2.2.5 RMTNTFY_HANDLE

The **RMTNTFY_HANDLE** serves as an RPC context handle for methods that take a **RMTNTFY_HANDLE** parameter. RPC context handles are as specified in [MS-RPCE].

This type is declared as follows:

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

The **RMTNTFY_HANDLE** context handle is returned by **RpcSyncRegisterForRemoteNotifications**.

2.2.6 NOTIFY_OPTIONS_CONTAINER

The **NOTIFY_OPTIONS_CONTAINER** structure encapsulates an **RPC_V2_NOTIFY_OPTIONS** structure, which specifies options for a change notification object that monitors a printer or print server for changes in state. For more information, see [MS-RPRN] section **2.2.1.13.1**.

```
typedef struct _NOTIFY_OPTIONS_CONTAINER {
  RPC_V2_NOTIFY_OPTIONS* pOptions;
} NOTIFY_OPTIONS_CONTAINER;
```

pOptions: A pointer to an **RPC_V2_NOTIFY_OPTIONS** structure, as specified in [MS-RPRN] section **2.2.1.13.1**.

2.2.7 NOTIFY_REPLY_CONTAINER

The **NOTIFY_REPLY_CONTAINER** structure encapsulates an **RPC_V2_NOTIFY_INFO** structure, which provides printer information members and current data for those members. For details, see [MS-RPRN] section **2.2.1.13.3**.

```
typedef struct _NOTIFY_REPLY_CONTAINER {
  RPC_V2_NOTIFY_INFO* pInfo;
} NOTIFY_REPLY_CONTAINER;
```

pInfo: A pointer to an **RPC_V2_NOTIFY_INFO** structure, as specified in [MS-RPRN] section **2.2.1.13.3**.

2.2.8 CORE_PRINTER_DRIVER

The **CORE_PRINTER_DRIVER** structure specifies information that identifies a specific **core printer driver**. See the **RpcAsyncGetCorePrinterDrivers (section 3.1.4.2.9)** method for an example of its use.

```
typedef struct _CORE_PRINTER_DRIVER {
  GUID CoreDriverGUID;
  FILETIME ftDriverDate;
  DWORDLONG dwlDriverVersion;
  wchar_t szPackageID[260];
} CORE_PRINTER_DRIVER;
```

CoreDriverGUID: A GUID value that MUST uniquely identify the package.

ftDriverDate: A **FILETIME** value that MUST specify the date this package was published.

dwlDriverVersion: A 32-bit value that MUST specify the version of the core printer driver that MAY <3>be used to match the driver version in the driver installation control file.

szPackageID: A **string** that MUST specify the package name. The server MUST generate a unique package name when the package is uploaded using the **RpcAsyncUploadPrinterDriverPackage (section 3.1.4.2.8)** method. The length of the string MUST not exceed the maximum path size, which is 260 characters, including a null terminator.

3 Protocol Details

The following sections specify details of the Print System Asynchronous Remote Protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 IRemoteWinspool Server Details

The Print System Asynchronous Remote Protocol server interface, IRemoteWinspool, is identified by UUID 76F03F96-CDFD-44fc-A22C-64950A001209.

3.1.1 Abstract Data Model

This section describes a conceptual model of the possible data organization that an implementation SHOULD maintain to participate in this protocol. The organization that is described in this section is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior described in this document.

This document describes the protocol in terms of an abstract data model that represents physical devices as objects and provides interfaces for communication and configuration management. The print server MUST behave as if it hosted the following objects in the hierarchy specified by the [MS-RPRN] section listed for each object below.

List of Print Server Names: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Printers: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Printer Drivers: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Ports: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Port Monitors: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Print Providers: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Print Processors: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

List of Known Printers: This object MUST adhere to the specification of the component with the same name, as specified in [MS-RPRN] section 3.1.1.

The abstract model MUST be able to relate each printer to a single driver as well as to a single **port**. Additionally each printer MUST be related to exactly one print processor.

For every object that is stored in the model, there MUST be an associated set of attributes, as specified in [MS-RPRN] section 2.2.2.

Note The previous conceptual data can be implemented by using a variety of techniques. An implementation can implement such data as needed.

3.1.2 Timers

No protocol timers are required on the server other than those that are used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE] section 3.2.3.2.

3.1.3 Initialization

The server SHOULD listen on **well-known endpoints** that are defined for this RPC interface. For more information, see section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

The Print System Asynchronous Remote Protocol MUST indicate to the RPC runtime that:

- 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 nonzero conformant value; and
- By using the strict_context_handle attribute, it is to reject use of context handles that are created by a method of an RPC interface other than this one;

as specified in [MS-RPCE] section 3.

The methods that this protocol defines are grouped into functional categories, and their syntax and behavior are specified in sections, as shown in the following table. Most methods described in these sections have functionally equivalent methods, as described in [MS-RPRN] section **3.1.4** and its subsections. This correspondence is detailed in the parent section of each functional category.

Functional category	Description	Section
Printer management	Methods used for discovering and obtaining access to supported printers.	3.1.4.1
Printer driver management	Methods for discovering and installing printer drivers.	3.1.4.2
Printer port management	Methods for discovering and communicating with printer ports.	3.1.4.3
Print-processor management	Methods for discovering and manipulating print-processor objects.	3.1.4.4
Port monitor management	Methods for discovering and installation of **port monitor modules**.	3.1.4.5
Form management	Methods for discovering and configuring printer forms.	3.1.4.6
Job management	Methods for discovering, defining and scheduling print jobs.	3.1.4.7
Job printing	Methods for adding documents, pages and data to print jobs.	3.1.4.8
Printing-related notifications	Methods for obtaining notifications of printing events.	3.1.4.9

The following table lists all the methods of the Print System Asynchronous Remote Protocol in ascending order of their **opnums**.

Methods in RPC Opnum Order

Method	Description
RpcAsyncOpenPrinter	**RpcAsyncOpenPrinter** retrieves a handle to a specified printer or print server. A client uses this method to obtain a print handle to an existing printer on a remote machine. Opnum: 0
RpcAsyncAddPrinter	**RpcAsyncAddPrinter** installs a remote printer driver. Opnum: 1
RpcAsyncSetJob	**RpcAsyncSetJob** pauses, resumes, cancels, or restarts a print job on a specified printer. This method can also set print job parameters, including the job priority and document name. Opnum: 2
RpcAsyncGetJob	**RpcAsyncGetJob** retrieves information about a specified print job on a specified printer. Opnum: 3
RpcAsyncEnumJobs	**RpcAsyncEnumJobs** retrieves information about a specified set of print jobs on a specified printer. Opnum: 4
RpcAsyncAddJob	**RpcAsyncAddJob** adds a print job to the list of jobs that a specified printer can schedule, and retrieves the name of the file used to store the job. Opnum: 5
RpcAsyncScheduleJob	**RpcAsyncScheduleJob** requests that a specified printer schedule a specified print job. Opnum: 6
RpcAsyncDeletePrinter	**RpcAsyncDeletePrinter** deletes the specified printer object. The client MUST still call **RpcAsyncClosePrinter** (see section 3.1.4.1.10) with the same **PRINTER_HANDLE** after calling **RpcAsyncDeletePrinter**. Opnum: 7
RpcAsyncSetPrinter	**RpcAsyncSetPrinter** sets the state of a specified printer. Opnum: 8
RpcAsyncGetPrinter	**RpcAsyncGetPrinter** retrieves information about a specified printer. Opnum: 9
RpcAsyncStartDocPrinter	**RpcAsyncStartDocPrinter** notifies a specified printer that a document is being spooled for printing. Opnum: 10
RpcAsyncStartPagePrinter	**RpcAsyncStartPagePrinter** notifies a specified printer that a page is about to be printed. Opnum: 11

Method	Description
RpcAsyncWritePrinter	**RpcAsyncWritePrinter** adds data to the file representing the spool file for a specified printer, if the spooling option is turned on; or it sends data to the device directly, if the printer is configured for direct printing. Opnum: 12
RpcAsyncEndPagePrinter	**RpcAsyncEndPagePrinter** notifies a specified printer that the application is at the end of a page in a print job. Opnum: 13
RpcAsyncEndDocPrinter	**RpcAsyncEndDocPrinter** signals the completion of the current print job on a specified printer. Opnum: 14
RpcAsyncAbortPrinter	The **RpcAsyncAbortPrinter** method aborts the current document on a specified printer. Opnum: 15
RpcAsyncGetPrinterData	**RpcAsyncGetPrinterData** retrieves printer data from a specified printer or print server. Opnum: 16
RpcAsyncGetPrinterDataEx	**RpcAsyncGetPrinterDataEx** retrieves configuration data for the specified printer or print server. Opnum: 17
RpcAsyncSetPrinterData	**RpcAsyncSetPrinterData** sets the configuration data for a printer or print server. Opnum: 18
RpcAsyncSetPrinterDataEx	**RpcAsyncSetPrinterDataEx** sets the configuration data for a printer or print server. Opnum: 19
RpcAsyncClosePrinter	**RpcAsyncClosePrinter** closes a handle to a printer object, server object, job object, or port object, which is opened by calling **RpcAsyncOpenPrinter** or **RpcAsyncAddPrinter**. Opnum: 20
RpcAsyncAddForm	**RpcAsyncAddForm** adds a form name to the list of supported forms. Opnum: 21
RpcAsyncDeleteForm	**RpcAsyncDeleteForm** removes a form name from the list of supported forms. Opnum: 22
RpcAsyncGetForm	**RpcAsyncGetForm** retrieves information about a specified form. Opnum: 23

Method	Description
RpcAsyncSetForm	**RpcAsyncSetForm** sets the form information for the specified printer. Opnum: 24
RpcAsyncEnumForms	**RpcAsyncEnumForms** enumerates the forms that the specified printer supports. Opnum: 25
RpcAsyncGetPrinterDriver	**RpcAsyncGetPrinterDriver** retrieves data about a specified printer driver on a specified printer. Opnum: 26
RpcAsyncEnumPrinterData	**RpcAsyncEnumPrinterData** enumerates configuration data for a specified printer. Opnum: 27
RpcAsyncEnumPrinterDataEx	**RpcAsyncEnumPrinterDataEx** enumerates all value names and data for a specified printer and key. Opnum: 28
RpcAsyncEnumPrinterKey	**RpcAsyncEnumPrinterKey** enumerates the subkeys of a specified key for a specified printer. Opnum: 29
RpcAsyncDeletePrinterData	**RpcAsyncDeletePrinterData** deletes a specified value from the configuration of a specified printer. Opnum: 30
RpcAsyncDeletePrinterDataEx	**RpcAsyncDeletePrinterDataEx** deletes a specified value from the configuration data of a specified printer. Opnum: 31
RpcAsyncDeletePrinterKey	**RpcAsyncDeletePrinterKey** deletes a specified key and all its subkeys from the configuration of a specified printer. Opnum: 32
RpcAsyncXcvData	**RpcAsyncXcvData** provides the means by which a **port monitor** client component can communicate with its server-side counterpart, the actual port monitor that is hosted by the server. Opnum: 33
RpcAsyncSendRecvBidiData	**RpcAsyncSendRecvBidiData** sends and receive **bidirectional** data. This method is used to communicate with **print monitors** that support such data. Opnum: 34
RpcAsyncCreatePrinterIC	**RpcAsyncCreatePrinterIC** creates an **information context** on a specified printer. Opnum: 35
RpcAsyncPlayGdiScriptOnPrinterIC	**RpcAsyncPlayGdiScriptOnPrinterIC** queries fonts for

Method	Description
	printer connections. Opnum: 36
RpcAsyncDeletePrinterIC	**RpcAsyncDeletePrinterIC** deletes a printer information context. Opnum: 37
RpcAsyncEnumPrinters	**RpcAsyncEnumPrinters** enumerates available local printers, printers on a specified print server, printers in a specified **domain**, or **print providers**. Opnum: 38
RpcAsyncAddPrinterDriver	**RpcAsyncAddPrinterDriver** installs a specified local or a remote printer driver on a specified print server, and it links the configuration, data, and driver files. Opnum: 39
RpcAsyncEnumPrinterDrivers	**RpcAsyncEnumPrinterDrivers** enumerates the printer drivers installed on a specified print server. Opnum: 40
RpcAsyncGetPrinterDriverDirectory	**RpcAsyncGetPrinterDriverDirectory** retrieves the path of the printer-driver directory on a specified print server. Opnum: 41
RpcAsyncDeletePrinterDriver	**RpcAsyncDeletePrinterDriver** removes the specified printer driver from the list of supported drivers for a specified print server. Opnum: 42
RpcAsyncDeletePrinterDriverEx	**RpcAsyncDeletePrinterDriverEx** removes the specified printer driver from the list of supported drivers on a specified print server, and deletes the files associated with the driver. This method also can delete specific versions of the driver. Opnum: 43
RpcAsyncAddPrintProcessor	**RpcAsyncAddPrintProcessor** installs a specified print processor on the specified server and adds its name to an internal list of supported print processors. Opnum: 44
RpcAsyncEnumPrintProcessors	**RpcAsyncEnumPrintProcessors** enumerates the print processors installed on a specified server. Opnum: 45
RpcAsyncGetPrintProcessorDirectory	**RpcAsyncGetPrintProcessorDirectory** retrieves the path for the print processor on the specified server. Opnum: 46
RpcAsyncEnumPorts	**RpcAsyncEnumPorts** enumerates the ports that are available for printing on a specified server. Opnum: 47

Method	Description
RpcAsyncEnumMonitors	**RpcAsyncEnumMonitors** retrieves information about the port monitors installed on a specified server. Opnum: 48
RpcAsyncAddPort	**RpcAsyncAddPort** adds a specified port name to the list of supported ports on a specified server. Opnum: 49
RpcAsyncSetPort	**RpcAsyncSetPort** sets the status associated with a specified port on a specified print server. Opnum: 50
RpcAsyncAddMonitor	**RpcAsyncAddMonitor** installs a specified local port monitor, and links the configuration, data, and monitor files on a specified print server. Opnum: 51
RpcAsyncDeleteMonitor	**RpcAsyncDeleteMonitor** removes a specified port monitor from a specified print server. Opnum: 52
RpcAsyncDeletePrintProcessor	**RpcAsyncDeletePrintProcessor** removes a specified print processor from a specified server. Opnum: 53
RpcAsyncEnumPrintProcessorDatatypes	**RpcAsyncEnumPrintProcessorDatatypes** enumerates the data types that a specified print processor supports. Opnum: 54
RpcAsyncAddPerMachineConnection	**RpcAsyncAddPerMachineConnection** adds an entry to the configuration store, persisting the print server name and the name of the print providers for a specified connection. Opnum: 55
RpcAsyncDeletePerMachineConnection	**RpcAsyncDeletePerMachineConnection** deletes the entry from the configuration store that corresponds to the pPrinterName parameter value. Opnum: 56
RpcAsyncEnumPerMachineConnections	**RpcAsyncEnumPerMachineConnections** enumerates each of the per-machine connections into a specified buffer. Opnum: 57
RpcSyncRegisterForRemoteNotifications	**RpcSyncRegisterForRemoteNotifications** opens a notification handle by using a printer handle, to listen for remote printer change notifications. Opnum: 58
RpcSyncUnRegisterForRemoteNotifications	**RpcSyncUnRegisterForRemoteNotifications** closes a notification handle that is opened by calling **RpcSyncRegisterForRemoteNotifications** (see

Method	Description
	section 3.1.4.9.1). Opnum: 59
RpcSyncRefreshRemoteNotifications	**RpcSyncRefreshRemoteNotifications** gets notification information for all requested members. This is called by a client if the "RemoteNotifyData Flags" key in the **RpcPrintPropertiesCollection** instance, which was returned as part of the notification from an **RpcAsyncGetRemoteNotifications** call, has the PRINTER_NOTIFY_INFO_DISCARDED bit set. PRINTER_NOTIFY_INFO_DISCARDED is defined in [MS-RPRN] section 2.2.3.7. Opnum: 60
RpcAsyncGetRemoteNotifications	A client uses **RpcAsyncGetRemoteNotifications** to poll the print server for specified change notifications. A call to this method is suspended until the server has a new change notification for the client. Subsequently, the client calls this method again to poll for additional notifications from the server. Opnum: 61
RpcAsyncInstallPrinterDriverFromPackage	**RpcAsyncInstallPrinterDriverFromPackage** installs a printer driver from a **driver package**. Opnum: 62
RpcAsyncUploadPrinterDriverPackage	**RpcAsyncUploadPrinterDriverPackage** uploads a driver package so it can be installed with RpcAsyncInstallPrinterDriverFromPackage. Opnum: 63
RpcAsyncGetCorePrinterDrivers	**RpcAsyncGetCorePrinterDrivers** retrieves the **globally unique identifier (GUID)**, the version, the date of the specified core printer drivers, and the path to their packages. Opnum: 64
RpcAsyncCorePrinterDriverInstalled	**RpcAsyncCorePrinterDriverInstalled** determines if a specific core printer driver is installed. Opnum: 65
RpcAsyncGetPrinterDriverPackagePath	**RpcAsyncGetPrinterDriverPackagePath** gets the path to the specified printer driver package. Opnum: 66
RpcAsyncDeletePrinterDriverPackage	**RpcAsyncDeletePrinterDriverPackage** deletes a specified printer driver package. Opnum: 67
RpcAsyncReadPrinter	**RpcAsyncReadPrinter** retrieves data from the specified job object. Opnum: 68
RpcAsyncResetPrinter	**RpcAsyncResetPrinter** resets the data type and device mode values to use for printing documents that

Method	Description
	are submitted by the **RpcAsyncStartDocPrinter** (section 3.1.4.8.1) method. Opnum: 69

All methods that are defined in this protocol are request/response RPC methods. Each method specifies either a DWORD or HRESULT data type for its return value. DWORD return values are Win32 error codes taken from the Windows error number space, as specified in [MS-ERREF] section 4.2. A return value of zero indicates successful completion, and a nonzero value indicates failure, with exceptions specified later in this section.

HRESULT return values are used as defined in [MS-ERREF] section 4.1. A return value of zero or a positive value indicates successful completion and a negative value indicates failure.

ERROR_MORE_DATA and ERROR_INSUFFICIENT_BUFFER are two nonzero return codes that have specific meanings in this protocol. When a method declaration in this specification has an output parameter that returns a required buffer size, the method MAY return one of the values from the following table. The caller SHOULD NOT treat these return values as errors. The caller SHOULD use the data returned by the method to learn the required buffer size and resize the buffers. The caller SHOULD call the method again by using the resized buffers. These cases are noted in the method definitions in this section.

Name/Value	Meaning
ERROR_INSUFFICIENT_BUFFER 0x0000007A	The buffer size specified in a method call is too small.
ERROR_MORE_DATA 0x000000EA	More data is available.

3.1.4.1 Printer Management Methods

The Printer Management methods support the discovery, access, and configuration of printer and print server objects. The following table presents a list of printer management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	Corresponding [MS-RPRN] method
RpcAsyncOpenPrinter	**RpcAsyncOpenPrinter** retrieves a handle to a specified printer or print server. A client uses this method to obtain a print handle to an existing printer on a remote machine. Opnum 0	**RpcOpenPrinterEx**
RpcAsyncAddPrinter	**RpcAsyncAddPrinter** installs a remote printer driver.	**RpcAddPrinterEx**

[MS-PAR] method	Description	Corresponding [MS-RPRN] method
	Opnum 1	
RpcAsyncDeletePrinter	**RpcAsyncDeletePrinter** deletes the specified printer object. The client MUST still call RpcAsyncClosePrinter (section 3.1.4.1.10) with the same **PRINTER_HANDLE** after calling **RpcAsyncDeletePrinter**. Opnum 7	RpcDeletePrinter
RpcAsyncSetPrinter	**RpcAsyncSetPrinter** sets the state of a specified printer, optionally by performing an action to change the state. Opnum 8	RpcSetPrinter
RpcAsyncGetPrinter	**RpcAsyncGetPrinter** retrieves information about a specified printer. Opnum 9	RpcGetPrinter
RpcAsyncGetPrinterData	**RpcAsyncGetPrinterData** retrieves printer data from a specified printer or print server. Opnum 16	RpcGetPrinterData
RpcAsyncGetPrinterDataEx	**RpcAsyncGetPrinterDataEx** retrieves configuration data for the specified printer or print server. Opnum 17	RpcGetPrinterDataEx
RpcAsyncSetPrinterData	**RpcAsyncSetPrinterData** sets the configuration data for a printer or print server. Opnum 18	RpcSetPrinterData
RpcAsyncSetPrinterDataEx	**RpcAsyncSetPrinterDataEx** sets the configuration data for a printer or print server. Opnum 19	RpcSetPrinterDataEx
RpcAsyncClosePrinter	**RpcAsyncClosePrinter** closes a handle to a printer object, server object, job object or port object, opened by calling RpcAsyncOpenPrinter or RpcAsyncAddPrinter. Opnum 20	RpcClosePrinter
RpcAsyncEnumPrinterData	**RpcAsyncEnumPrinterData** enumerates configuration data for a specified printer. Opnum 27	RpcEnumPrinterData

[MS-PAR] method	Description	Corresponding [MS-RPRN] method
RpcAsyncEnumPrinterDataEx	**RpcAsyncEnumPrinterDataEx** enumerates all value names and data for a specified printer and key. Opnum 28	RpcEnumPrinterDataEx
RpcAsyncEnumPrinterKey	**RpcAsyncEnumPrinterKey** enumerates the subkeys of a specified key for a specified printer. Opnum 29	RpcEnumPrinterKey
RpcAsyncDeletePrinterData	**RpcAsyncDeletePrinterData** deletes a specified value from the configuration of a specified printer. Opnum 30	RpcDeletePrinterData
RpcAsyncDeletePrinterDataEx	**RpcAsyncDeletePrinterDataEx** deletes a specified value from the configuration data of a specified printer, which consists of a set of named and typed values stored in a hierarchy of registry keys. Opnum 31	RpcDeletePrinterDataEx
RpcAsyncDeletePrinterKey	**RpcAsyncDeletePrinterKey** deletes a specified key and all of its subkeys from the configuration of a specified printer. Opnum 32	RpcDeletePrinterKey
RpcAsyncSendRecvBidiData	**RpcAsyncSendRecvBidiData** sends and receives bidirectional data. This method is used to communicate with print monitors that support such data. Opnum 34	RpcSendRecvBidiData
RpcAsyncCreatePrinterIC	**RpcAsyncCreatePrinterIC** creates an information context on a specified printer. Opnum 35	RpcCreatePrinterIC
RpcAsyncPlayGdiScriptOnPrinterIC	**RpcAsyncPlayGdiScriptOnPrinterIC** queries fonts for printer connections. Opnum 36	RpcPlayGdiScriptOnPrinterIC
RpcAsyncDeletePrinterIC	**RpcAsyncDeletePrinterIC** deletes a printer information context. Opnum 37	RpcDeletePrinterIC
RpcAsyncEnumPrinters	**RpcAsyncEnumPrinters** enumerates available local printers, printers on a specified print server, printers in a specified domain, or	RpcEnumPrinters

[MS-PAR] method	Description	Corresponding [MS-RPRN] method
	print providers. Opnum 38	
RpcAsyncAddPerMachineConnection	**RpcAsyncAddPerMachineConnection** adds an entry to the configuration store, persisting the print server name and the name of the print provider for a specified connection. Opnum 55	RpcAddPerMachineConnection
RpcAsyncDeletePerMachineConnection	**RpcAsyncDeletePerMachineConnection** deletes the entry from the configuration store that corresponds to the pPrinterName parameter value. Opnum 56	RpcDeletePerMachineConnection
RpcAsyncEnumPerMachineConnections	**RpcAsyncEnumPerMachineConnections** enumerates each of the per-machine connections into a specified buffer. Opnum 57	RpcEnumPerMachineConnections
RpcAsyncResetPrinter	**RpcAsyncResetPrinter** resets the data type and device mode values to use for printing documents submitted by the RpcAsyncStartDocPrinter (section 3.1.4.8.1) method. Opnum 69	RpcResetPrinter

3.1.4.1.1 RpcAsyncOpenPrinter (Opnum 0)

RpcAsyncOpenPrinter retrieves a handle to a specified printer or print server. This method is this protocol's counterpart of **RpcOpenPrinterEx**, as specified in [MS-RPRN] section **3.1.4.2.14**. A client uses this method to obtain a print handle to an existing printer on a remote computer.

```
DWORD RpcAsyncOpenPrinter(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pPrinterName,
  [out] PRINTER_HANDLE* pHandle,
  [in, string, unique] wchar_t* pDatatype,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] DWORD AccessRequired,
  [in] SPLCLIENT_CONTAINER* pClientInfo
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pPrinterName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

pHandle: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

pDatatype: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

pDevModeContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

AccessRequired: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

pClientInfo: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.14**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.2.14**, and apply substitutions described in section 3.1.4.1.

Processing and Response Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.2.14**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.2 RpcAsyncAddPrinter (Opnum 1)

RpcAsyncAddPrinter installs a remote printer driver. This method is this protocol's counterpart of **RpcAddPrinterEx**, as specified in [MS-RPRN] section **3.1.4.2.15**.

```
DWORD RpcAsyncAddPrinter(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in] PRINTER_CONTAINER* pPrinterContainer,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] SECURITY_CONTAINER* pSecurityContainer,
  [in] SPLCLIENT_CONTAINER* pClientInfo,
  [out] PRINTER_HANDLE* pHandle
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

pPrinterContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

pDevModeContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

pSecurityContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

pClientInfo: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

pHandle: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.15**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.2.15**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements, as specified in [MS-RPRN] section **3.1.4.2.15**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.3 RpcAsyncDeletePrinter (Opnum 7)

RpcAsyncDeletePrinter deletes the specified printer object. This method is this protocol's counterpart of **RpcDeletePrinter**, as specified in [MS-RPRN], section **3.1.4.2.4**.

The client MUST still call **RpcAsyncClosePrinter** with the same **PRINTER_HANDLE** after calling **RpcAsyncDeletePrinter**.

```
DWORD RpcAsyncDeletePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.4**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: The method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.4**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.4 RpcAsyncSetPrinter (Opnum 8)

RpcAsyncSetPrinter sets the state of a specified printer, optionally by performing an action to change the state. This method is this protocol's counterpart of **RpcSetPrinter**, as defined in [MS-RPRN] section **3.1.4.2.5**.

```
DWORD RpcAsyncSetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] PRINTER_CONTAINER* pPrinterContainer,
  [in] DEVMODE_CONTAINER* pDevModeContainer,
  [in] SECURITY_CONTAINER* pSecurityContainer,
  [in] DWORD Command
);
```

hPrinter: MUST be a handle to a printer object or server object, which MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pPrinterContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.5**.

pDevModeContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.5**.

pSecurityContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.5**.

Command: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.5**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.5**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.5 RpcAsyncGetPrinter (Opnum 9)

RpcAsyncGetPrinter retrieves information about a specified printer. This method is this protocol's counterpart of **RpcGetPrinter**, as defined in [MS-RPRN] section **3.1.4.2.6**.

```
DWORD RpcAsyncGetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPrinter,
```

```
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**).

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.6**.

pPrinter: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.6**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.6**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.6**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.2.6**, with the substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements, as specified in [MS-RPRN] section **3.1.4.2.6**, with the substitutions described in section 3.1.4.1 applied.

3.1.4.1.6 RpcAsyncGetPrinterData (Opnum 16)

RpcAsyncGetPrinterData retrieves printer data from a specified printer or print server. This method is this protocol's counterpart of **RpcGetPrinterData**, as defined in [MS-RPRN] section **3.1.4.2.7**.

```
DWORD RpcAsyncGetPrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName,
  [out] DWORD* pType,
  [out, size_is(nSize)] unsigned char* pData,
  [in] DWORD nSize,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object or server object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.7**.

pType: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.7**.

pData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.7**.

nSize: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.7**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.7**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.7**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.7**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.7 RpcAsyncGetPrinterDataEx (Opnum 17)

RpcAsyncGetPrinterDataEx retrieves configuration data for the specified printer or print server. This method is this protocol's counterpart of **RpcGetPrinterDataEx**, defined in [MS-RPRN] section **3.1.4.2.19**.

```
DWORD RpcAsyncGetPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName,
  [out] DWORD* pType,
  [out, size_is(nSize)] unsigned char* pData,
  [in] DWORD nSize,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object or server object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

pType: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

pData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

nSize: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.19**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.19**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.19**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.8 RpcAsyncSetPrinterData (Opnum 18)

RpcAsyncSetPrinterData sets the configuration data for a printer or print server. This method is this protocol's counterpart of **RpcSetPrinterData**, defined in [MS-RPRN] section **3.1.4.2.8**.

```
DWORD RpcAsyncSetPrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName,
  [in] DWORD Type,
  [in, size_is(cbData)] unsigned char* pData,
  [in] DWORD cbData
);
```

hPrinter: MUST be a handle to a printer object or server object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.8**.

Type: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.8**.

pData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.8**.

cbData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.8**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.8**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.8**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.9 RpcAsyncSetPrinterDataEx (Opnum 19)

RpcAsyncSetPrinterDataEx sets the configuration data for a printer or print server. This method is this protocol's counterpart of **RpcSetPrinterDataEx**, defined in [MS-RPRN] section **3.1.4.2.18**.

```
DWORD RpcAsyncSetPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName,
  [in] DWORD Type,
  [in, size_is(cbData)] unsigned char* pData,
  [in] DWORD cbData
);
```

hPrinter: MUST be a handle to a printer object or server object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.18**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.18**.

Type: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.18**.

pData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.18**.

cbData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.18**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.18**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.18**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.10 RpcAsyncClosePrinter (Opnum 20)

RpcAsyncClosePrinter closes a handle to a printer object, server object, job object, or port object, opened by calling **RpcAsyncOpenPrinter** or **RpcAsyncAddPrinter**. This method is this protocol's counterpart of **RpcClosePrinter**, defined in [MS-RPRN] section **3.1.4.2.9**.

```
DWORD RpcAsyncClosePrinter(
  [in, out] PRINTER_HANDLE* phPrinter
);
```

phPrinter: MUST be a pointer to a handle to a printer object, server object, job object or port object, which MUST have been opened using either **RpcAsyncOpenPrinter** (section 3.1.4.1.1) or **RpcAsyncAddPrinter** (section 3.1.4.1.2).

Return Values: MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.9**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.9**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.11 RpcAsyncEnumPrinterData (Opnum 27)

RpcAsyncEnumPrinterData enumerates configuration data for a specified printer. This method is this protocol's counterpart of **RpcEnumPrinterData**, defined in [MS-RPRN] section **3.1.4.2.16**.

```
DWORD RpcAsyncEnumPrinterData(
  [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)] unsigned char* pData,
  [in] DWORD cbData,
  [out] DWORD* pcbData
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

dwIndex: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

cbValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

pcbValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

pType: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

pData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

cbData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

pcbData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.16**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.16**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.16**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.12 RpcAsyncEnumPrinterDataEx (Opnum 28)

RpcAsyncEnumPrinterDataEx enumerates all value names and data for a specified printer and key. This method is this protocol's counterpart of **RpcEnumPrinterDataEx**, defined in [MS-RPRN] section **3.1.4.2.20**.

```
DWORD RpcAsyncEnumPrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [out, size_is(cbEnumValues)] unsigned char* pEnumValues,
  [in] DWORD cbEnumValues,
  [out] DWORD* pcbEnumValues,
  [out] DWORD* pnEnumValues
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.20**.

pEnumValues: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.20**.

cbEnumValues: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.20**.

pcbEnumValues: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.20**.

pnEnumValues: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.20**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.20**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.20**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.13 RpcAsyncEnumPrinterKey (Opnum 29)

RpcAsyncEnumPrinterKey enumerates the subkeys of a specified key for a specified printer. This method is this protocol's counterpart of **RpcEnumPrinterKey**, defined in [MS-RPRN] section **3.1.4.2.21**.

```
DWORD RpcAsyncEnumPrinterKey(
  [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: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.21**.

pSubkey: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.21**.

cbSubkey: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.21**.

pcbSubkey: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.21**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.21**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.21**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.14 RpcAsyncDeletePrinterData (Opnum 30)

RpcAsyncDeletePrinterData deletes a specified value from the configuration of a specified printer. This method is this protocol's counterpart of **RpcDeletePrinterData**, defined in [MS-RPRN] section **3.1.4.2.17**.

```
DWORD RpcAsyncDeletePrinterData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pValueName
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.17**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.17**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.17**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.15 RpcAsyncDeletePrinterDataEx (Opnum 31)

RpcAsyncDeletePrinterDataEx deletes a specified value from the configuration data of a specified printer, which consists of a set of named and typed values stored in a hierarchy of **registry** keys. This method is this protocol's counterpart of **RpcDeletePrinterDataEx**, defined in [MS-RPRN] section **3.1.4.2.22**.

```
DWORD RpcAsyncDeletePrinterDataEx(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName,
  [in, string] const wchar_t* pValueName
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.22**.

pValueName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.22**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.22** with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.22** with substitutions described in section 3.1.4.1 applied.

3.1.4.1.16 RpcAsyncDeletePrinterKey (Opnum 32)

RpcAsyncDeletePrinterKey deletes a specified key and all of its subkeys from the configuration of a specified printer. This method is this protocol's counterpart of **RpcDeletePrinterKey**, defined in [MS-RPRN] section **3.1.4.2.23**.

```
DWORD RpcAsyncDeletePrinterKey(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] const wchar_t* pKeyName
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pKeyName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.23**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.23**, with substitutions described in section 3.1.4.1 applied

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.23**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.17 RpcAsyncSendRecvBidiData (Opnum 34)

RpcAsyncSendRecvBidiData sends and receives bidirectional data. This method is used to communicate with print monitors that support such data. This method is this protocol's counterpart of **RpcSendRecvBidiData**, defined in [MS-RPRN] section **3.1.4.2.27**.

```
DWORD RpcAsyncSendRecvBidiData(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] const wchar_t* pAction,
  [in] RPC_BIDI_REQUEST_CONTAINER* pReqData,
  [out] RPC_BIDI_RESPONSE_CONTAINER** ppRespData
);
```

hPrinter: MUST be a handle to a printer object, which MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pAction: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.27**.

pReqData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.27**.

ppRespData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.27**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.27**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.27**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.18 RpcAsyncCreatePrinterIC (Opnum 35)

RpcAsyncCreatePrinterIC creates an information context for a specified printer. This method is this protocol's counterpart of **RpcCreatePrinterIC**, specified in [MS-RPRN] section 3.1.4.2.10.

```
DWORD RpcAsyncCreatePrinterIC(
  [in] PRINTER_HANDLE hPrinter,
  [out] GDI_HANDLE* pHandle,
  [in] DEVMODE_CONTAINER* pDevModeContainer
);
```

hPrinter: MUST be a handle to a printer object ([MS-RPRN] section 2.2.1.1.3), which MUST have been opened using **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pHandle: MUST adhere to the specification of the parameter with the same name in [MS-RPRN] section 3.1.4.2.10.

pDevModeContainer: MUST adhere to the specification of the parameter with the same name in [MS-RPRN] section 3.1.4.2.10.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section 3.1.4.2.10, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section 3.1.4.2.10, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.19 RpcAsyncPlayGdiScriptOnPrinterIC (Opnum 36)

RpcAsyncPlayGdiScriptOnPrinterIC returns font information for a printer connection. This method is this protocol's counterpart of **RpcPlayGdiScriptOnPrinterIC**, specified in [MS-RPRN] section 3.1.4.2.11.

```
DWORD RpcAsyncPlayGdiScriptOnPrinterIC(
  [in] GDI_HANDLE hPrinterIC,
  [in, size_is(cIn)] unsigned char* pIn,
  [in] DWORD cIn,
  [out, size_is(cOut)] unsigned char* pOut,
  [in] DWORD cOut,
  [in] DWORD* ul
);
```

hPrinterIC: MUST be a printer information context handle ([MS-RPRN] section 2.2.1.1.1), which MUST have been returned by **RpcAsyncCreatePrinterIC (section 3.1.4.1.18)**.

pIn: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.2.11.

cIn: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.2.11.

pOut: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.2.11.

cOut: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.2.11.

ul: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.2.11.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section 3.1.4.2.11, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section 3.1.4.2.11, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.20 RpcAsyncDeletePrinterIC (Opnum 37)

RpcAsyncDeletePrinterIC deletes a printer information context. This method is this protocol's counterpart of **RpcDeletePrinterIC**, specified in [MS-RPRN] section 3.1.4.2.12.

```
DWORD RpcAsyncDeletePrinterIC(
  [in, out] GDI_HANDLE* phPrinterIC
);
```

phPrinterIC: MUST be a non-null pointer to a printer information context handle ([MS-RPRN] section 2.2.1.1.1), which MUST have been returned by **RpcAsyncCreatePrinterIC (section 3.1.4.1.18)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section 3.1.4.2.12, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section 3.1.4.2.12, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.21 RpcAsyncEnumPrinters (Opnum 38)

RpcAsyncEnumPrinters enumerates available local printers, printers on a specified print server, printers in a specified domain, or print providers. This method is this protocol's counterpart of **RpcEnumPrinters**, defined in [MS-RPRN] section **3.1.4.2.1**.

```
DWORD RpcAsyncEnumPrinters(
  [in] handle_t hRemoteBinding,
  [in] DWORD Flags,
  [in, string, unique] wchar_t* Name,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPrinterEnum,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

Flags: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

Name: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

pPrinterEnum: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.1**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.1**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.22 RpcAsyncAddPerMachineConnection (Opnum 55)

RpcAsyncAddPerMachineConnection persists the print server name and the name of the print provider for a specified connection by adding an entry to the configuration store. This method is this protocol's counterpart of **RpcAddPerMachineConnection**, defined in [MS-RPRN] section **3.1.4.2.24**.

```
DWORD RpcAsyncAddPerMachineConnection(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pServer,
  [in, string] const wchar_t* pPrinterName,
  [in, string] const wchar_t* pPrintServer,
  [in, string] const wchar_t* pProvider
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pServer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.24**.

pPrinterName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.24**.

pPrintServer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.24**.

pProvider: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.24**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements:This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.24**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.24**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.23 RpcAsyncDeletePerMachineConnection (Opnum 56)

RpcAsyncDeletePerMachineConnection deletes the entry from the configuration store that corresponds to the pPrinterName parameter value. This method is this protocol's counterpart of **RpcDeletePerMachineConnection**, defined in [MS-RPRN] section **3.1.4.2.25**.

```
DWORD RpcAsyncDeletePerMachineConnection(
```

```
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pServer,
  [in, string] const wchar_t* pPrinterName
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pServer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.25**.

pPrinterName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.25**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.25**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.25**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.24 RpcAsyncEnumPerMachineConnections (Opnum 57)

RpcAsyncEnumPerMachineConnections enumerates each of the per-machine connections into a specified buffer. This method is this protocol's counterpart of **RpcEnumPerMachineConnections**, defined in [MS-RPRN] section **3.1.4.2.26**.

```
DWORD RpcAsyncEnumPerMachineConnections(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pServer,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPrinterEnum,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pServer: MUST adhere to the specification of the parameter with the same name in [MS-RPRN] section **3.1.4.2.26**.

pPrinterEnum: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.26**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.26**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.26**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.26**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.26**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.26**, with substitutions described in section 3.1.4.1 applied.

3.1.4.1.25 RpcAsyncResetPrinter (Opnum 69)

RpcAsyncResetPrinter resets the data type and device mode values to use for printing documents submitted by the **RpcAsyncStartDocPrinter (section 3.1.4.8.1)** method. This method is this protocol's counterpart of **RpcResetPrinter**, defined in [MS-RPRN] section **3.1.4.2.13**.

```
DWORD RpcAsyncResetPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] wchar_t* pDatatype,
  [in] DEVMODE_CONTAINER* pDevModeContainer
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pDatatype: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.13**.

pDevModeContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.2.13**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.2.13**, with substitutions described in section 3.1.4.1 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.2.13**, with substitutions described in section 3.1.4.1 applied.

3.1.4.2 Printer-Driver Management Methods

The Printer-Driver Management methods support the discovery, access, and installation of printer drivers. The following table presents a list of printer-driver management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncGetPrinterDriver	**RpcAsyncGetPrinterDriver** retrieves data about a specified printer driver on a specified printer. Opnum 26	**RpcGetPrinterDriver2**
RpcAsyncAddPrinterDriver	**RpcAsyncAddPrinterDriver** installs a specified local or a remote printer driver on a specified print server, and it links the configuration, data, and driver files. Opnum 39	**RpcAddPrinterDriverEx**
RpcAsyncEnumPrinterDrivers	**RpcAsyncEnumPrinterDrivers** enumerates the printer drivers installed on a specified print server. Opnum 40	**RpcEnumPrinterDrivers**
RpcAsyncGetPrinterDriverDirectory	**RpcAsyncGetPrinterDriverDirectory** retrieves the path of the printer-driver directory on a specified print server. Opnum 41	**RpcGetPrinterDriverDirectory**
RpcAsyncDeletePrinterDriver	**RpcAsyncDeletePrinterDriver** removes the specified printer driver from the list of supported drivers for a specified print server. Opnum 42	**RpcDeletePrinterDriver**
RpcAsyncDeletePrinterDriverEx	**RpcAsyncDeletePrinterDriverEx** removes the specified printer driver from the list of supported drivers on a specified print server, and deletes the files associated with the driver. This method also can delete specific versions of the driver. Opnum 43	**RpcDeletePrinterDriverEx**
RpcAsyncInstallPrinterDriverFromPackage	**RpcAsyncInstallPrinterDriverFromPackage** installs a printer driver from a driver package. Opnum 62	None.
RpcAsyncUploadPrinterDriverPack	**RpcAsyncUploadPrinterDriverPack**	None.

[MS-PAR] method	Description	[MS-RPRN] method
age	**age** uploads a driver package so that it can be installed with **RpcAsyncInstallPrinterDriverFromPackage**. Opnum 63	
RpcAsyncGetCorePrinterDrivers	**RpcAsyncGetCorePrinterDrivers** gets the GUID, version, and date of the specified core printer drivers and the path to their packages. Opnum 64	None.
RpcAsyncCorePrinterDriverInstalled	**RpcAsyncCorePrinterDriverInstalled** determines if a specific core printer driver is installed. Opnum 65	None.
RpcAsyncGetPrinterDriverPackagePath	**RpcAsyncGetPrinterDriverPackagePath** gets the path to the specified printer driver package. Opnum 66	None.
RpcAsyncDeletePrinterDriverPackage	**RpcAsyncDeletePrinterDriverPackage** deletes a specified printer driver package. Opnum 67	None.

3.1.4.2.1 RpcAsyncGetPrinterDriver (Opnum 26)

RpcAsyncGetPrinterDriver retrieves data about a specified printer driver on a specified printer. This method is this protocol's counterpart of **RpcGetPrinterDriver2**, defined in [MS-RPRN] section **3.1.4.4.6**.

```
DWORD RpcAsyncGetPrinterDriver(
  [in] PRINTER_HANDLE hPrinter,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pDriver,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [in] DWORD dwClientMajorVersion,
  [in] DWORD dwClientMinorVersion,
  [out] DWORD* pdwServerMaxVersion,
  [out] DWORD* pdwServerMinVersion
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

pDriver: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

dwClientMajorVersion: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

dwClientMinorVersion: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

pdwServerMaxVersion: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

pdwServerMinVersion: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.6**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.6** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.6** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.2 RpcAsyncAddPrinterDriver (Opnum 39)

RpcAsyncAddPrinterDriver installs a specified local or a remote printer driver on a specified print server, and it links the configuration, data, and driver files. This method is this protocol's counterpart of **RpcAddPrinterDriverEx**, defined in [MS-RPRN] section **3.1.4.4.8**.

```
DWORD RpcAsyncAddPrinterDriver(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in] DRIVER_CONTAINER* pDriverContainer,
  [in] DWORD dwFileCopyFlags
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.8**.

pDriverContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.8**.

dwFileCopyFlags: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.8**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.8** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.8** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.3 RpcAsyncEnumPrinterDrivers (Opnum 40)

RpcAsyncEnumPrinterDrivers enumerates the printer drivers installed on a specified print server. This method is this protocol's counterpart of **RpcEnumPrinterDrivers**, defined in [MS-RPRN] section **3.1.4.4.2**.

```
DWORD RpcAsyncEnumPrinterDrivers(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pDrivers,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

Level: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

pDrivers: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

cbBuf: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.2**.

Return Values: This method MUST return zero to indicate successful completion or a nonpzero Windows error code to indicate failure. The client MUST treat any nonpzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.2** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.2** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.4 RpcAsyncGetPrinterDriverDirectory (Opnum 41)

RpcAsyncGetPrinterDriverDirectory retrieves the path of the printer-driver directory on a specified print server. This method is this protocol's counterpart of **RpcGetPrinterDriverDirectory**, defined in [MS-RPRN] section **3.1.4.4.4**.

```
DWORD RpcAsyncGetPrinterDriverDirectory(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pDriverDirectory,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

pDriverDirectory: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

cbBuf: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

pcbNeeded: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.4**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.4** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.4** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.5 RpcAsyncDeletePrinterDriver (Opnum 42)

RpcAsyncDeletePrinterDriver removes the specified printer driver from the list of supported drivers for a specified print server. This method is this protocol's counterpart of **RpcDeletePrinterDriver**, defined in [MS-RPRN] section **3.1.4.4.5**.

```
DWORD RpcAsyncDeletePrinterDriver(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pDriverName
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.5**.

pEnvironment: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.5**.

pDriverName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.5** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.5** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.6 RpcAsyncDeletePrinterDriverEx (Opnum 43)

RpcAsyncDeletePrinterDriverEx removes the specified printer driver from the list of supported drivers on a specified print server, and deletes the files associated with the driver. This method also can delete specific versions of the driver. This method is this protocol's counterpart of **RpcDeletePrinterDriverEx**, defined in [MS-RPRN] section **3.1.4.4.7**.

```
DWORD RpcAsyncDeletePrinterDriverEx(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pDriverName,
  [in] DWORD dwDeleteFlag,
  [in] DWORD dwVersionNum
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.7**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.7**.

pDriverName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.7**.

dwDeleteFlag: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.7**.

dwVersionNum: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.4.7**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.4.7** with substitutions described in section 3.1.4.2 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.4.7** with substitutions described in section 3.1.4.2 applied.

3.1.4.2.7 RpcAsyncInstallPrinterDriverFromPackage (Opnum 62)

RpcAsyncInstallPrinterDriverFromPackage installs a printer driver from a driver package.

```
HRESULT RpcAsyncInstallPrinterDriverFromPackage(
  [in] handle_t hRemoteBinding,
  [in, string, unique] const wchar_t* pszServer,
```

```
  [in, string, unique] const wchar_t* pszInfPath,
  [in, string] const wchar_t* pszDriverName,
  [in, string] const wchar_t* pszEnvironment,
  [in] DWORD dwFlags
);
```

hRemoteBinding: This parameter MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server on which to install the printer driver. The **string** that is referenced by this parameter MUST NOT be null and it MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. For RPC bind handles, refer to [MS-RPCE]. For rules governing print server names, refer to [MS-RPRN] section **2.2.4.1**.

pszInfPath: MUST be a non-null pointer to a **string** that MUST specify the path to a driver-installation control file that MUST specify the printer driver, and that MAY<4> be used to install the driver on a target system. For rules governing path names, refer to [MS-RPRN] section **2.2.4.7**.

pszDriverName: MUST be a non-null pointer to a **string** that MUST specify the name of the driver.

pszEnvironment: MUST be a non-null pointer to a **string** that MUST specify the environment name for which the driver MUST be installed. For rules governing environment names, and Windows behaviors, refer to [MS-RPRN] section **2.2.4.5**.

dwFlags: This parameter MUST contain the value that MUST specify the flags that indicate the options the method MUST use to perform its function. The value of this parameter MUST be zero or the bitwise OR of one or more of the constant values listed in the following table. If the value of this parameter is zero, this method SHOULD install only the files that will not overwrite files with a newer version.

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	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Where the bits are defined as:

Value	Description
A IPDFP_COPY_ALL_FILES	This method SHOULD install all files, even if doing so would overwrite some newer-versioned files.

Return Values: This method MUST return a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The **string** pointed to by the **pszInfPath** parameter MUST contain a valid path name.
- The **string** pointed to by the **pszEnvironment** parameter MUST specify one of the supported environment names on this server for that type of driver.
- The **dwFlags** parameter MUST be zero or contain a valid bitwise OR combination of one or more of the valid constants for this parameter.

Processing and Response Requirements

If parameter validation fails, the server MUST return immediately with a failure indication and an appropriate error in its response to the client.

Otherwise, the server MUST process the method call by:

- Installing a printer driver from the driver package that is located in the print server's **driver store**.
- Returning a response that MUST contain the status of the operation.

If the operation is successful, the server MUST install the printer driver from the driver package before returning the response.

3.1.4.2.8 RpcAsyncUploadPrinterDriverPackage (Opnum 63)

RpcAsyncUploadPrinterDriverPackage uploads a driver package so it can be installed with **RpcAsyncInstallPrinterDriverFromPackage**.

```
HRESULT RpcAsyncUploadPrinterDriverPackage(
  [in] handle_t hRemoteBinding,
  [in, string, unique] const wchar_t* pszServer,
  [in, string] const wchar_t* pszInfPath,
  [in, string] const wchar_t* pszEnvironment,
  [in] DWORD dwFlags,
  [in, out, unique, size_is(*pcchDestInfPath)]
    wchar_t* pszDestInfPath,
  [in, out] DWORD* pcchDestInfPath
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server to which this method will upload the printer driver package. This **string** MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. The **string** value MUST NOT be null. For details on RPC bind handles, see [MS-RPCE]. For rules governing print server names, see [MS-RPRN] section **2.2.4.1**.

pszInfPath: MUST be a non-null pointer to a **string** that MUST specify the path to a driver-installation control file that MUST specify the printer driver, and that MAY<5> be used to install the driver on a target system. For rules governing path names, see [MS-RPRN] section **2.2.4.7**.

pszEnvironment: This parameter MUST be a non-null pointer to a **string** that MUST specify the environment name for which the driver package MUST be uploaded. For rules governing environment names, and Windows behaviors, see [MS-RPRN] section **2.2.4.5**.

dwFlags: This parameter MUST contain the value that MUST specify the flags that indicate the options that this method MUST use to perform its function. The value of this parameter MUST be zero or the bitwise OR of one or more of the constant values listed in the following table. If the value of this parameter is zero, this method MUST upload to the print server the driver package that is named by the string pointed to by the **pszInfPath** parameter to the print server, but only if that driver package is not already present on the print server.

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
0	0	0	0	0	B	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Where the bits are defined as:

Value	Description
A UPDP_UPLOAD_ALWAYS	If this flag is set, this method MUST upload the driver package files specified by the **pszInfPath** parameter even if the driver package is already present on the print server.
B UPDP_CHECK_DRIVERSTORE	If this flag is set, and before it uploads any files, this method MUST check the print server's driver store to see if the driver package is already present on the print server. If the driver package is not present on the print server, this method MUST upload the driver package that is specified by the **pszInfPath** parameter. This flag MUST be ignored if UPDP_UPLOAD_ALWAYS is set.

pszDestInfPath: MUST be a non-null pointer to a buffer that MUST receive a **string** that specifies the full path of the directory to which the driver installation control file was copied. For rules governing path names, see [MS-RPRN] section **2.2.4.7**. Note: This value SHOULD NOT<6> be used as an input parameter.

pcchDestInfPath: On input, this parameter MUST be a pointer to a variable that specifies the size, in characters, of the buffer that is referenced by the **pszDestInfPath** parameter. The specified size in characters MUST be at least 260. On output, the variable to which this parameter points MUST receive the size, in characters, of the path string, including the terminating null character, that was written into the buffer that is referenced by the **pszDestInfPath** parameter.

Return Values: This method MUST return a positive value or zero to indicate successful completion or negative values to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The **string** pointed to by the **pszInfPath** parameter MUST contain a valid path name.
- The **string** pointed to by the **pszEnvironment** parameter MUST specify one of the supported environment names on this server for that type of driver.

- The **dwFlags** parameter MUST be zero or contain a valid bitwise OR combination of one or more of the valid constants for this parameter.
- The size specified by the variable pointed to by **pcchDestInfPath** MUST be at least 260 characters.

If parameter validation fails, the server MUST return immediately with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST process the method call by:

- Uploading only the signed driver package to the driver store of the print server so that it can be installed with **RpcAsyncInstallPrinterDriverFromPackage**.
- Returning the driver store path name of the file that describes the printer driver in the buffer pointed to by the output parameter **pszDestInfPath**.<7>
- Setting the contents of the output parameter **pcchDestInfPath** to the size of the data in the buffer.
- Returning a response that MUST contain the output parameters mentioned above and the status of the operation.

If the operation is successful, the server MUST upload the driver package into the system driver store before returning the response.

3.1.4.2.9 RpcAsyncGetCorePrinterDrivers (Opnum 64)

RpcAsyncGetCorePrinterDrivers gets the GUID, versions, and publish dates of the specified core printer drivers, and the paths to their packages.

```
HRESULT RpcAsyncGetCorePrinterDrivers(
  [in] handle t hRemoteBinding,
  [in, string, unique] const wchar_t* 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
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server from which to get the core printer driver information. This **string** MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. For details on RPC bind handles, see [MS-RPCE]. For rules governing print server names, see [MS-RPRN] section **2.2.4.1**.

pszEnvironment: MUST be a non-null pointer to a **string** that MUST specify the environment name for which the core driver information will be returned. For rules governing environment names, and Windows behaviors, see [MS-RPRN] section **2.2.4.5**.

cchCoreDrivers: MUST be the size, in bytes, of the buffer that is referenced by the **pszzCoreDriverDependencies** parameter.

pszzCoreDriverDependencies: MUST be a non-null pointer to a **multisz** that MUST specify a list of the IDs<8> of the core printer drivers to be retrieved. (Use **RpcAsyncGetPrinterDriver** to retrieve a **RPC_DRIVER_INFO_8** structure that contains this list of IDs in its **pszzCoreDriverDependencies** member.)

cCorePrinterDrivers: This parameter MUST specify the count of **CORE_PRINTER_DRIVER** structures that are contained in the buffer that is pointed to by the **pCorePrinterDrivers** parameter. It MUST also equal the number of IDs that are specified in the multisz that is pointed to by the **pszzCoreDriverDependencies** parameter.

pCorePrinterDrivers: This parameter MUST be a pointer to a buffer that MUST receive an array of **CORE_PRINTER_DRIVER** structures, which are defined in section 2.2.8.

Return Values: This method MUST return a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [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.
- **cCorePrinterDrivers** MUST be equal to or greater than 1.
- **cCorePrinterDrivers** MUST be equal to the number of GUIDs present in **pszzCoreDriverDependencies**.
- The **pCorePrinterDrivers** parameter MUST NOT be null.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client

Processing and Response Requirements: Otherwise, 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 driver objects as part of processing this method call.

3.1.4.2.10 RpcAsyncCorePrinterDriverInstalled (Opnum 65)

RpcAsyncCorePrinterDriverInstalled determines if a specific core printer driver is installed.

```
HRESULT RpcAsyncCorePrinterDriverInstalled(
  [in] handle_t hRemoteBinding,
  [in, string, unique] const wchar_t* pszServer,
```

```
  [in, string] const wchar_t* pszEnvironment,
  [in] GUID CoreDriverGUID,
  [in] FILETIME ftDriverDate,
  [in] DWORDLONG dwlDriverVersion,
  [out] int* pbDriverInstalled
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server to check and determine if a core printer driver is installed. This **string** MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. For details on RPC bind handles, see [MS-RPCE]. For rules governing print server names, see [MS-RPRN] section **2.2.4.1**.

pszEnvironment: MUST be a non-null pointer to a **string** that MUST specify the environment name for which the server MUST be checked if the driver is installed. For rules governing environment names, and Windows behaviors, see [MS-RPRN] section **2.2.4.5**.

CoreDriverGUID: This parameter MUST specify the GUID of the core printer driver.

ftDriverDate: MUST specify the date of the core printer driver.<9>

dwlDriverVersion: MUST specify the version<10> of the core printer driver.

pbDriverInstalled: MUST point to a variable that MUST receive one of the following values.

Value	Meaning
0	The driver, or a newer version of the driver, is not installed.
1	The driver, or a newer version of the driver, is installed.

Return Values: This method MUST return a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [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.
- The **pbDriverInstalled** parameter MUST NOT be null.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST process the method call by:

- Searching for the core printer driver with the specified CoreDriverGUID, ftDriverDate, and dwlDriverVersion in the list of installed core printer drivers on the print server.

- Setting the value of the variable pointed to by **pbDriverInstalled** to 1 if the search succeeded or to zero if not.
- 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 driver objects as part of processing this method call.

3.1.4.2.11 RpcAsyncGetPrinterDriverPackagePath (Opnum 66)

RpcAsyncGetPrinterDriverPackagePath gets the path to the specified printer driver package.

```
HRESULT RpcAsyncGetPrinterDriverPackagePath(
  [in] handle_t hRemoteBinding,
  [in, string, unique] const wchar_t* 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
);
```

hRemoteBinding: This parameter MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server from which to get the printer driver package path. This **string** MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. For details on RPC bind handles, see [MS-RPCE]. For rules governing print server names, see [MS-RPRN] section **2.2.4.1**.

pszEnvironment: 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, and Windows behaviors, see [MS-RPRN] section **2.2.4.5**.

pszLanguage: MUST be null, or MUST be a nonnull pointer to a **string** that MUST specify the language for which the driver package path MUST<11> be returned.

pszPackageID: MUST be a non-null pointer to a **string** that MUST specify package name. The package name MUST be obtained by calling **RpcAsyncGetCorePrinterDrivers**.

pszDriverPackageCab: This parameter MUST be a pointer to a buffer that MUST<12> receive a **string** that specifies the path name of the driver package file. For rules governing path names, see [MS-RPRN] section **2.2.4.7**. **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<13> 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 a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [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.
- The value of the **pszPackageID** parameter MUST NOT be null.
- The size specified by **cchDriverPackageCab** MUST be sufficient to hold the path name of the driver package file.
- The value of the **pcchRequiredSize** parameter MUST NOT be null.

If parameter validation fails, the server MUST complete the operation immediately to indicate a failure and MUST return an appropriate error as its response to the client. If the validation fails because the size of the buffer is too small, then the server MUST calculate the required number of characters and return this value in the variable pointed to by the **pcchRequiredSize** out parameter and return ERROR_INSUFFICIENT_BUFFER.

Processing and Response Requirements: Otherwise, 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.

3.1.4.2.12 RpcAsyncDeletePrinterDriverPackage (Opnum 67)

RpcAsyncDeletePrinterDriverPackage deletes a specified printer driver package.

```
HRESULT RpcAsyncDeletePrinterDriverPackage(
  [in] handle_t hRemoteBinding,
  [in, string, unique] const wchar_t* pszServer,
  [in, string] const wchar_t* pszInfPath,
  [in, string] const wchar_t* pszEnvironment
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pszServer: MUST be a non-null pointer to a **string** that MUST specify the name of the print server from which to delete the printer driver package. This **string** MUST contain a server name that is identical to the server name that was used to create the **hRemoteBinding** parameter. For details on RPC bind handles, see [MS-RPCE]. For rules governing print server names, see [MS-RPRN] section **2.2.4.1**.

pszInfPath: MUST be a non-null pointer to a **string** that MUST specify the path name of a driver installation control file that MUST specify the printer driver and MAY<14> be used to delete the driver from the print server. For rules governing path names, see [MS-RPRN] section **2.2.4.7**.

pszEnvironment: MUST be a non-null pointer to a **string** that MUST specify the environment name for which the driver will be deleted. For rules governing environment names and Windows behavior, see [MS-RPRN] section **2.2.4.5**.

Return Values: This method MUST return a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The string pointed to by the **pszInfPath** parameter MUST contain an existing path name.
- The string pointed to by the **pszEnvironment** parameter MUST specify one of the supported environment names on the server for that type of driver.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST search for the driver package based on **pszInfPath** and delete it from driver store of the print server.

If the operation is successful, the server MUST delete the driver package from the driver store of the print server, before returning a response that contains the status of the operation.

3.1.4.3 Printer-Port Management Methods

The Printer-Port Management methods support the discovery and communication with printer ports. The following table presents a list of printer-port management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncXcvData	**RpcAsyncXcvData** provides the means by which a port monitor client component can communicate with its server-side counterpart, the actual port-monitor hosted by the	**RpcXcvData**

[MS-PAR] method	Description	[MS-RPRN] method
	server. Opnum 33	
RpcAsyncEnumPorts	**RpcAsyncEnumPorts** enumerates the ports that are available for printing on a specified server. Opnum 47	RpcEnumPorts
RpcAsyncAddPort	**RpcAsyncAddPort** adds a specified port name to the list of supported ports on a specified print server. Opnum 49	RpcAddPortEx
RpcAsyncSetPort	**RpcAsyncSetPort** sets the status associated with a specified port on a specified print server. Opnum 50	RpcSetPort

3.1.4.3.1 RpcAsyncXcvData (Opnum 33)

RpcAsyncXcvData provides the means by which a port monitor client component can communicate with its server-side counterpart, the actual port monitor hosted by the server. This method is this protocol's counterpart of **RpcXcvData**, defined in [MS-RPRN] section **3.1.4.6.5**.

```
DWORD RpcAsyncXcvData(
  [in] PRINTER_HANDLE hXcv,
  [in, string] const wchar_t* pszDataName,
  [in, size_is(cbInputData)] unsigned char* pInputData,
  [in] DWORD cbInputData,
  [out, size_is(cbOutputData)] unsigned char* pOutputData,
  [in] DWORD cbOutputData,
  [out] DWORD* pcbOutputNeeded,
  [in, out] DWORD* pdwStatus
);
```

hXcv: MUST be a handle to a port object and it MUST have been opened by using RpcAsyncOpenPrinter (section 3.1.4.1.1).

pszDataName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

pInputData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

cbInputData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

pOutputData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

cbOutputData: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

pcbOutputNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

pdwStatus: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error except as noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.6.5** with substitutions described in section 3.1.4.3 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.6.5** with substitutions described in section 3.1.4.3 applied.

3.1.4.3.2 RpcAsyncEnumPorts (Opnum 47)

RpcAsyncEnumPorts enumerates the ports that are available for printing on a specified server. This method is this protocol's counterpart of **RpcEnumPorts**, defined in [MS-RPRN] section **3.1.4.6.1**.

```
DWORD RpcAsyncEnumPorts(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPort,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

pPort: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, except as noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.6.1** with substitutions described in section 3.1.4.3 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.6.1** with substitutions described in section 3.1.4.3 applied.

3.1.4.3.3 RpcAsyncAddPort (Opnum 49)

RpcAsyncAddPort adds a specified port name to the list of supported ports on a specified print server. This method is this protocol's counterpart of **RpcAddPortEx**, defined in [MS-RPRN] section **3.1.4.6.3**.

```
DWORD RpcAsyncAddPort(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in] PORT_CONTAINER* pPortContainer,
  [in] PORT_VAR_CONTAINER* pPortVarContainer,
  [in, string] wchar_t* pMonitorName
);
```

hRemoteBinding: MUST be an RPC explicit binding handle. RPC binding handles are specified in [C706] section 2.3.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.3**.

pPortContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.3**.

pPortVarContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.3**.

pMonitorName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.6.3** with substitutions described in section 3.1.4.3 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.6.3** with substitutions described in section 3.1.4.3 applied.

3.1.4.3.4 RpcAsyncSetPort (Opnum 50)

RpcAsyncSetPort sets the status associated with a specified port on a specified print server. This method is this protocol's counterpart of **RpcSetPort**, defined in [MS-RPRN] section **3.1.4.6.4**.

```
DWORD RpcAsyncSetPort(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pPortName,
  [in] PORT_CONTAINER* pPortContainer
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.4**.

pPortName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section 3.1.4.6.4.

pPortContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.6.4**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.6.4** with substitutions described in section 3.1.4.3 applied

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.6.4** with substitutions described in section 3.1.4.3 applied.

3.1.4.4 Print-Processor Management Methods

The Print-Processor Management methods support the discovery and manipulation of print processor objects. The following table presents a list of print processor management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncAddPrintProcessor	**RpcAsyncAddPrintProcessor** installs a specified print processor on the specified server and adds its name to an internal list of supported print processors. Opnum 44	RpcAddPrintProcessor
RpcAsyncEnumPrintProcessors	**RpcAsyncEnumPrintProcessors** enumerates the print processors installed on a specified server. Opnum 45	RpcEnumPrintProcessors
RpcAsyncGetPrintProcessorDirectory	**RpcAsyncGetPrintProcessorDirectory** retrieves the path for the print processor on the specified server. Opnum 46	RpcGetPrintProcessorDirectory
RpcAsyncDeletePrintProcessor	**RpcAsyncDeletePrintProcessor** removes a specified print processor from a specified server. Opnum 53	RpcDeletePrintProcessor
RpcAsyncEnumPrintProcessorDatatypes	**RpcAsyncEnumPrintProcessorDatatypes** enumerates the data types that a specified print processor supports. Opnum 54	RpcEnumPrintProcessorDatatypes

3.1.4.4.1 RpcAsyncAddPrintProcessor (Opnum 44)

RpcAsyncAddPrintProcessor installs a specified print processor on the specified server and adds its name to an internal list of supported print processors. This method is this protocol's counterpart of **RpcAddPrintProcessor**, defined in [MS-RPRN] section **3.1.4.8.1**.

```
DWORD RpcAsyncAddPrintProcessor(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string] wchar_t* pEnvironment,
  [in, string] wchar_t* pPathName,
  [in, string] wchar_t* pPrintProcessorName
);
```

hRemoteBinding: This parameter MUST be an RPC explicit binding handle.

pName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.1**.

pEnvironment: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.1**.

pPathName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.1**.

pPrintProcessorName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.8.1** with substitutions described in section 3.1.4.4 applied

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.8.1** with substitutions described in section 3.1.4.4 applied.

3.1.4.4.2 RpcAsyncEnumPrintProcessors (Opnum 45)

RpcAsyncEnumPrintProcessors enumerates the print processors installed on a specified server. This method is this protocol's counterpart of **RpcEnumPrintProcessors**, defined in [MS-RPRN] section **3.1.4.8.2**.

```
DWORD RpcAsyncEnumPrintProcessors(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPrintProcessorInfo,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

pPrintProcessorInfo: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.2**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, except as noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.8.2** with substitutions described in section 3.1.4.4 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.8.2** with substitutions described in section 3.1.4.4 applied.

3.1.4.4.3 RpcAsyncGetPrintProcessorDirectory (Opnum 46)

RpcAsyncGetPrintProcessorDirectory retrieves the path for the print processor on the specified server. This method is this protocol's counterpart of **RpcGetPrintProcessorDirectory**, defined in [MS-RPRN] section **3.1.4.8.3**.

```
DWORD RpcAsyncGetPrintProcessorDirectory(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pEnvironment,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pPrintProcessorDirectory,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

pPrintProcessorDirectory: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.8.3** with substitutions described in section 3.1.4.4 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.8.3** with substitutions described in section 3.1.4.4 applied.

3.1.4.4.4 RpcAsyncDeletePrintProcessor (Opnum 53)

RpcAsyncDeletePrintProcessor removes a specified print processor from a specified server. This method is this protocol's counterpart of **RpcDeletePrintProcessor**, defined in [MS-RPRN] section **3.1.4.8.4**.

```
DWORD RpcAsyncDeletePrintProcessor(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* Name,
  [in, string, unique] wchar_t* pEnvironment,
  [in, string] wchar_t* pPrintProcessorName
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

Name: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.4**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.4**.

pPrintProcessorName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.4**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.8.4** with substitutions described in section 3.1.4.4 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.8.4** with substitutions described in section 3.1.4.4 applied.

3.1.4.4.5 RpcAsyncEnumPrintProcessorDatatypes (Opnum 54)

RpcAsyncEnumPrintProcessorDatatypes enumerates the data types that a specified print processor supports. This method is this protocol's counterpart of **RpcEnumPrintProcessorDatatypes**, defined in [MS-RPRN] section **3.1.4.8.5**.

```
DWORD RpcAsyncEnumPrintProcessorDatatypes(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in, string, unique] wchar_t* pPrintProcessorName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pDatatypes,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: This parameter MUST be an RPC explicit binding handle.

pName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

pPrintProcessorName: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

Level: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

pDatatypes: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

cbBuf: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

pcbNeeded: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

pcReturned: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.8.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.8.5** with substitutions described in section 3.1.4.4 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.8.5** with substitutions described in section 3.1.4.4 applied.

3.1.4.5 Port Monitor Management Methods

The Port Monitor Management methods support the discovery and installation of port monitor modules. The following table presents a list of port monitor management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements and processing and response requirements.

Print System Asynchronous Remote Protocol method	Description	Print System Remote Protocol method
RpcAsyncEnumMonitors	**RpcAsyncEnumMonitors** retrieves information about the port monitors installed on a specified server. Opnum 48	RpcEnumMonitors
RpcAsyncAddMonitor	**RpcAsyncAddMonitor** installs a specified local port monitor, and links the configuration, data, and monitor files on a specified print server. Opnum 51	RpcAddMonitor
RpcAsyncDeleteMonitor	**RpcAsyncDeleteMonitor** removes a specified port monitor from a specified print server. Opnum 52	RpcDeleteMonitor

3.1.4.5.1 RpcAsyncEnumMonitors (Opnum 48)

RpcAsyncEnumMonitors retrieves information about the port monitors installed on a specified server. This method is this protocol's counterpart of **RpcEnumMonitors**, defined in [MS-RPRN] section **3.1.4.7.1**.

```
DWORD RpcAsyncEnumMonitors(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* pName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pMonitor,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

pName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

pMonitor: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

pcReturned: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.7.1** with substitutions described in section 3.1.4.5 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.7.1** with substitutions described in section 3.1.4.5 applied.

3.1.4.5.2 RpcAsyncAddMonitor (Opnum 51)

RpcAsyncAddMonitor installs a specified local port monitor, and links the configuration, data, and monitor files on a specified print server. This method is this protocol's counterpart of **RpcAddMonitor**, defined in [MS-RPRN] section **3.1.4.7.2**.

```
DWORD RpcAsyncAddMonitor(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* Name,
  [in] MONITOR_CONTAINER* pMonitorContainer
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

Name: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.2**.

pMonitorContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.2**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.7.2** with substitutions described in section 3.1.4.5 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.7.2** with substitutions described in section 3.1.4.5 applied.

3.1.4.5.3 RpcAsyncDeleteMonitor (Opnum 52)

RpcAsyncDeleteMonitor removes a specified port monitor from a specified print server. This method is this protocol's counterpart of **RpcDeleteMonitor**, defined in [MS-RPRN] section **3.1.4.7.3**.

```
DWORD RpcAsyncDeleteMonitor(
  [in] handle_t hRemoteBinding,
  [in, string, unique] wchar_t* Name,
  [in, string, unique] wchar_t* pEnvironment,
  [in, string] wchar_t* pMonitorName
);
```

hRemoteBinding: MUST be an RPC explicit binding handle.

Name: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.3**.

pEnvironment: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.3**.

pMonitorName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.7.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.7.3** with substitutions described in section 3.1.4.5 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.7.3** with substitutions described in section 3.1.4.5 applied.

3.1.4.6 Form Management Methods

The Form Management methods support the discovery and configuration of printer forms. The following table presents a list of form management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncAddForm	**RpcAsyncAddForm** adds a form name to the list of supported forms. Opnum 21	RpcAddForm
RpcAsyncDeleteForm	**RpcAsyncDeleteForm** removes a form name from the list of supported forms. Opnum 22	RpcDeleteForm
RpcAsyncGetForm	**RpcAsyncGetForm** retrieves information about a specified form. Opnum 23	RpcGetForm
RpcAsyncSetForm	**RpcAsyncSetForm** sets the form information for the specified printer. Opnum 24	RpcSetForm
RpcAsyncEnumForms	**RpcAsyncEnumForms** enumerates the forms that the specified printer supports. Opnum 25	RpcEnumForms

3.1.4.6.1 RpcAsyncAddForm (Opnum 21)

RpcAsyncAddForm adds a form name to the list of supported forms. This method is this protocol's counterpart of **RpcAddForm**, defined in [MS-RPRN] section **3.1.4.5.1**.

```
DWORD RpcAsyncAddForm(
  [in] PRINTER_HANDLE hPrinter,
  [in] FORM_CONTAINER* pFormInfoContainer
);
```

hPrinter: MUST be a handle to a printer object or server object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pFormInfoContainer: MUST adhere to the specification of the parameter with the same name, specified in [MS-RPRN] section **3.1.4.5.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.5.1** with substitutions described in section 3.1.4.6 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.5.1** with substitutions described in section 3.1.4.6 applied.

3.1.4.6.2 RpcAsyncDeleteForm (Opnum 22)

RpcAsyncDeleteForm removes a form name from the list of supported forms. This method is this protocol's counterpart of **RpcDeleteForm**, defined in [MS-RPRN] section **3.1.4.5.2**.

```
DWORD RpcAsyncDeleteForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName
);
```

hPrinter: MUST be a handle to a printer object or server object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pFormName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.2**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.5.2** with substitutions described in section 3.1.4.6 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.5.2** with substitutions described in section 3.1.4.6 applied.

3.1.4.6.3 RpcAsyncGetForm (Opnum 23)

RpcAsyncGetForm retrieves information about a specified form. This method is this protocol's counterpart of **RpcGetForm**, defined in [MS-RPRN] section **3.1.4.5.3**.

```
DWORD RpcAsyncGetForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pForm,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object or server object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pFormName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.3**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.3**.

pForm: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.3**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.3**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.5.3** with substitutions described in section 3.1.4.6 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.5.3** with substitutions described in section 3.1.4.6 applied.

3.1.4.6.4 RpcAsyncSetForm (Opnum 24)

RpcAsyncSetForm sets the form information for the specified printer. This method is this protocol's counterpart of **RpcSetForm**, defined in [MS-RPRN] section **3.1.4.5.4**.

```
DWORD RpcAsyncSetForm(
  [in] PRINTER_HANDLE hPrinter,
  [in, string] wchar_t* pFormName,
  [in] FORM_CONTAINER* pFormInfoContainer
);
```

hPrinter: MUST be a handle to a printer object or server object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pFormName: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.4**.

pFormInfoContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.4**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.5.4** with substitutions described in section 3.1.4.6 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.5.4** with substitutions described in section 3.1.4.6 applied.

3.1.4.6.5 RpcAsyncEnumForms (Opnum 25)

RpcAsyncEnumForms enumerates the forms that the specified printer supports. This method is this protocol's counterpart of **RpcEnumForms**, defined in [MS-RPRN] section **3.1.4.5.5**.

```
DWORD RpcAsyncEnumForms(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pForm,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hPrinter: MUST be a handle to a printer object or server object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.5**.

pForm: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.5**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.5**.

pcbNeeded: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.5**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.5.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.5.5** with substitutions described in section 3.1.4.6 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.5.5** with substitutions described in section 3.1.4.6 applied.

3.1.4.7 Job Management Methods

The Job Management methods support the discovery, definition and scheduling of print jobs. The following table presents a list of job management methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncSetJob	**RpcAsyncSetJob** pauses, resumes, cancels, or restarts a print job on a specified printer. This method can also set print job parameters, including the job priority and document name. Opnum 2	**RpcSetJob**
RpcAsyncGetJob	**RpcAsyncGetJob** retrieves information about a specified print job on a specified printer. Opnum 3	**RpcGetJob**
RpcAsyncEnumJobs	**RpcAsyncEnumJobs** retrieves information about a specified set of print jobs on a specified printer. Opnum 4	**RpcEnumJobs**
RpcAsyncAddJob	**RpcAsyncAddJob** adds a print job to the list of jobs that a specified printer can schedule, and retrieves the name of the file used to store the job. Opnum 5	**RpcAddJob**
RpcAsyncScheduleJob	**RpcAsyncScheduleJob** requests that a specified printer schedule a specified print job. Opnum 6	**RpcScheduleJob**

3.1.4.7.1 RpcAsyncSetJob (Opnum 2)

RpcAsyncSetJob pauses, resumes, cancels, or restarts a print job on a specified printer. This method is this protocol's counterpart of **RpcSetJob**, defined in [MS-RPRN] section **3.1.4.3.1**. This method can also set print job parameters, including the job priority and document name.

```
DWORD RpcAsyncSetJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in, unique] JOB_CONTAINER* pJobContainer,
  [in] DWORD Command
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

JobId: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.1**.

pJobContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.1**.

Command: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.3.1** with substitutions described in section 3.1.4.7 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.3.1** with substitutions described in section 3.1.4.7 applied.

3.1.4.7.2 RpcAsyncGetJob (Opnum 3)

RpcAsyncGetJob retrieves information about a specified print job on a specified printer. This method is this protocol's counterpart of **RpcGetJob**, defined in [MS-RPRN] section **3.1.4.3.2**.

```
DWORD RpcAsyncGetJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

JobId: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.2**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.2**.

pJob: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.2**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.2**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.2**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.3.2** with substitutions described in section 3.1.4.7 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.3.2** with substitutions described in section 3.1.4.7 applied.

3.1.4.7.3 RpcAsyncEnumJobs (Opnum 4)

RpcAsyncEnumJobs retrieves information about a specified set of print jobs on a specified printer. This method is this protocol's counterpart of **RpcEnumJobs**, defined in [MS-RPRN] section **3.1.4.3.3**.

```
DWORD RpcAsyncEnumJobs(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD FirstJob,
  [in] DWORD NoJobs,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded,
  [out] DWORD* pcReturned
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

FirstJob: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

NoJobs: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

pJob: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

pcReturned: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.3.3**, with substitutions described in section 3.1.4.7 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.3.3**, with substitutions described in section 3.1.4.7 applied.

3.1.4.7.4 RpcAsyncAddJob (Opnum 5)

RpcAsyncAddJob adds a print job to the list of jobs that a specified printer can schedule, and retrieves the name of the file used to store the job. This method is this protocol's counterpart of **RpcAddJob**, defined in [MS-RPRN] section **3.1.4.3.4**.

```
DWORD RpcAsyncAddJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD Level,
  [in, out, unique, size_is(cbBuf)]
    unsigned char* pAddJob,
  [in] DWORD cbBuf,
  [out] DWORD* pcbNeeded
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Level: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.4**.

pAddJob: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.4**.

cbBuf: This parameter MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.4**.

pcbNeeded: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.4**.

Return Values: This method MUST return zero to indicate successful completion or a Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error, with exceptions noted in section 3.1.4.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.3.4**, with substitutions described in section 3.1.4.7 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.3.4**, with substitutions described in section 3.1.4.7 applied.

3.1.4.7.5 RpcAsyncScheduleJob (Opnum 6)

RpcAsyncScheduleJob requests that a specified printer schedule a specified print job. This method is this protocol's counterpart of **RpcScheduleJob**, defined in [MS-RPRN] section **3.1.4.3.5**.

```
DWORD RpcAsyncScheduleJob(
  [in] PRINTER_HANDLE hPrinter,
  [in] DWORD JobId
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

JobId: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.3.5**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.3.5**, with substitutions described in section 3.1.4.7 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.3.5**, with substitutions described in section 3.1.4.7 applied.

3.1.4.8 Job Printing Methods

The Job Printing methods support the adding of documents, pages and text to print jobs. The following table presents a list of job printing methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcAsyncStartDocPrinter	**RpcAsyncStartDocPrinter** notifies a specified printer that a document is being spooled for printing. This method is this protocol's counterpart of **RpcStartDocPrinter**, defined in [MS-RPRN] section **3.1.4.9.1**. Opnum 10	**RpcStartDocPrinter**
RpcAsyncStartPagePrinter	**RpcAsyncStartPagePrinter** notifies a	**RpcStartPagePrinter**

[MS-PAR] method	Description	[MS-RPRN] method
	specified printer that a page is about to be printed. This method is this protocol's counterpart of **RpcStartPagePrinter**, defined in [MS-RPRN] section **3.1.4.9.2**. Opnum 11	
RpcAsyncWritePrinter	**RpcAsyncWritePrinter** adds data to the file representing the spool file for a specified printer, if the spooling option is turned on; or it sends data to the device directly, if the printer is configured for direct printing. This method is this protocol's counterpart of **RpcWritePrinter**, defined in [MS-RPRN] section **3.1.4.9.3**. Opnum 12	**RpcWritePrinter**
RpcAsyncEndPagePrinter	**RpcAsyncEndPagePrinter** notifies a specified printer that the application is at the end of a page in a print job. This method is this protocol's counterpart of **RpcEndPagePrinter**, defined in [MS-RPRN] section **3.1.4.9.4**. Opnum 13	**RpcEndPagePrinter**
RpcAsyncEndDocPrinter	**RpcAsyncEndDocPrinter** signals the completion of the current print job on a specified printer. This method is this protocol's counterpart of **RpcEndDocPrinter**, defined in [MS-RPRN] section **3.1.4.9.7**. Opnum 14	**RpcEndDocPrinter**
RpcAsyncAbortPrinter	The **RpcAsyncAbortPrinter** method aborts the current document on a specified printer. This method is this protocol's counterpart of **RpcAbortPrinter**, defined in [MS-RPRN] section **3.1.4.9.5**. Opnum 15	**RpcAbortPrinter**
RpcAsyncReadPrinter	**RpcAsyncReadPrinter** retrieves data from the specified job object. This method is this protocol's counterpart of **RpcReadPrinter**, as specified in [MS-RPRN] section **3.1.4.9.6**. Opnum 68	**RpcReadPrinter**

3.1.4.8.1 RpcAsyncStartDocPrinter (Opnum 10)

RpcStartDocPrinter notifies a specified printer that a document is being spooled for printing. This method is this protocol's counterpart of **RpcStartDocPrinter**, defined in [MS-RPRN] section **3.1.4.9.1**.

```
DWORD RpcAsyncStartDocPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in] DOC_INFO_CONTAINER* pDocInfoContainer,
  [out] DWORD* pJobId
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pDocInfoContainer: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.1**.

pJobId: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.1**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.9.1**, with substitutions described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.9.1**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.2 RpcAsyncStartPagePrinter (Opnum 11)

RpcAsyncStartPagePrinter notifies a specified printer that a page is about to be printed. This method is this protocol's counterpart of **RpcStartPagePrinter**, as defined in [MS-RPRN] section **3.1.4.9.2**.

```
DWORD RpcAsyncStartPagePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.9.2**, with substitutions as described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements, as specified in [MS-RPRN] section **3.1.4.9.2**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.3 RpcAsyncWritePrinter (Opnum 12)

RpcAsyncWritePrinter adds data to the file representing the spool file for a specified printer, if the spooling option is turned on; or it sends data to the device directly, if the printer is configured for direct printing. This method is this protocol's counterpart of **RpcWritePrinter**, as defined in [MS-RPRN] section **3.1.4.9.3**.

```
DWORD RpcAsyncWritePrinter(
  [in] PRINTER_HANDLE hPrinter,
  [in, size_is(cbBuf)] unsigned char* pBuf,
  [in] DWORD cbBuf,
  [out] DWORD* pcWritten
);
```

hPrinter: MUST be a handle to a printer object or port object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.3**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.3**.

pcWritten: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.3**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements, as specified in [MS-RPRN] section **3.1.4.9.3**, with substitutions described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements, as specified in [MS-RPRN] section **3.1.4.9.3**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.4 RpcAsyncEndPagePrinter (Opnum 13)

RpcAsyncEndPagePrinter notifies a specified printer that the application is at the end of a page in a print job. This method is this protocol's counterpart of **RpcEndPagePrinter**, defined in [MS-RPRN] section **3.1.4.9.4**.

```
DWORD RpcAsyncEndPagePrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.9.4**, with substitutions described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.9.4**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.5 RpcAsyncEndDocPrinter (Opnum 14)

RpcAsyncEndDocPrinter signals the completion of the current print job on a specified printer. This method is this protocol's counterpart of **RpcEndDocPrinter**, defined in [MS-RPRN] section **3.1.4.9.7**.

```
DWORD RpcAsyncEndDocPrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.9.7**, with substitutions described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.9.7**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.6 RpcAsyncAbortPrinter (Opnum 15)

RpcAsyncAbortPrinter aborts the current document on a specified printer. This method is this protocol's counterpart of **RpcAbortPrinter**, defined in [MS-RPRN] section **3.1.4.9.5**.

```
DWORD RpcAsyncAbortPrinter(
  [in] PRINTER_HANDLE hPrinter
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any nonzero return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.9.5** and apply substitutions described in section 3.1.4.8.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.9.5**, with substitutions described in section 3.1.4.8 applied.

3.1.4.8.7 RpcAsyncReadPrinter (Opnum 68)

RpcAsyncReadPrinter retrieves data from the specified job object. This method is this protocol's counterpart of **RpcReadPrinter**, as specified in [MS-RPRN] section **3.1.4.9.6**.

```
DWORD RpcAsyncReadPrinter(
  [in] PRINTER_HANDLE hPrinter,
  [out, size_is(cbBuf)] unsigned char* pBuf,
  [in] DWORD cbBuf,
  [out] DWORD* pcNoBytesRead
);
```

hPrinter: MUST be a handle to a job object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.6**.

cbBuf: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.6**.

pcNoBytesRead: MUST adhere to the specification of the parameter with the same name, as specified in [MS-RPRN] section **3.1.4.9.6**.

Return Values: This method MUST return zero to indicate successful completion or a nonzero Windows error code to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: This method MUST adhere to the parameter validation requirements specified in [MS-RPRN] section **3.1.4.9.6**, with substitutions described in section 3.1.4.8 applied.

Processing and Response Requirements: This method MUST adhere to the processing and response requirements specified in [MS-RPRN] section **3.1.4.9.6**, with substitutions described in section 3.1.4.8 applied.

3.1.4.9 Printing-Related Notification Methods

The Printing-Related Notification methods support the registration for and receipt of notification events concerning a specific print job. The following table presents a list of printing-related notification methods and their counterparts in the Print System Remote Protocol, as specified in [MS-RPRN]. All methods are specified in sections that follow.

Wherever appropriate, this protocol's method descriptions refer to the method descriptions in [MS-RPRN] for parameter details, parameter validation requirements, and processing and response requirements.

[MS-PAR] method	Description	[MS-RPRN] method
RpcSyncRegisterForRemoteNotifications	**RpcSyncRegisterForRemoteNotifications** opens a notification handle by using a printer handle, to listen for remote printer change notifications. Opnum 58	None.
RpcSyncUnRegisterForRemoteNotifications	**RpcSyncUnRegisterForRemoteNotifications** closes a notification handle opened by calling **RpcSyncRegisterForRemoteNotifications** (section 3.1.4.9.1). Opnum 59	None.
RpcSyncRefreshRemoteNotifications	**RpcSyncRefreshRemoteNotifications** gets notification information for all requested members. This is called by a client if the "RemoteNotifyData Flags" key in the **RpcPrintPropertiesCollection** instance, which was returned as part of the notification from an **RpcAsyncGetRemoteNotifications** call, has the **PRINTER_NOTIFY_INFO_DISCARDED** bit set. **PRINTER_NOTIFY_INFO_DISCARDED** is defined in [MS-RPRN] section **2.2.3.7**. Opnum 60	None.
RpcAsyncGetRemoteNotifications	A client uses **RpcAsyncGetRemoteNotifications** to poll the print server for specified change notifications. A call to this method is suspended until the server has a new change notification for the client. Subsequently, the client calls this method again to poll for additional notifications from the server. Opnum 61	None.

3.1.4.9.1 RpcSyncRegisterForRemoteNotifications (Opnum 58)

RpcSyncRegisterForRemoteNotifications opens a notification handle by using a printer handle, to listen for remote printer change notifications.

```
HRESULT RpcSyncRegisterForRemoteNotifications(
  [in] PRINTER_HANDLE hPrinter,
  [in] RpcPrintPropertiesCollection* pNotifyFilter,
  [out] RMTNTFY_HANDLE* phRpcHandle
);
```

hPrinter: MUST be a handle to a printer object and it MUST have been opened by using either **RpcAsyncOpenPrinter (section 3.1.4.1.1)** or **RpcAsyncAddPrinter (section 3.1.4.1.2)**.

pNotifyFilter: MUST be a pointer to an **RpcPrintPropertiesCollection (section 2.2.4)** instance that MUST contain the caller-specified notification filter settings.

phRpcHandle: MUST be a pointer to a variable that MUST receive the remote notification handle.

Return Values: This method MUST return a positive value or zero to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The **hPrinter** printer handle MUST NOT be null and MUST point to a printer object that can be monitored for notifications.
- The printer handle MUST be authorized to monitor printer objects for notifications.
- The **pNotifyFilter** parameter MUST point to an **RpcPrintPropertiesCollection** instance that has all the name-value pairs required to register for notifications.

If parameter validation fails, the server MUST return immediately with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST process the method call by:

- Creating a notification object that points to the printer object and contains notification filter data sent by the client.
- Associating the response with an RPC handle and returning it to the user.

If the operation is successful, the server MUST modify the list of notification listeners associated with the printer object so that any changes to the object can be sent to the new listener.

3.1.4.9.2 RpcSyncUnRegisterForRemoteNotifications (Opnum 59)

RpcSyncUnRegisterForRemoteNotifications closes a notification handle opened by calling **RpcSyncRegisterForRemoteNotifications (section 3.1.4.9.1)**.

```
HRESULT RpcSyncUnRegisterForRemoteNotifications(
  [in, out] RMTNTFY_HANDLE* phRpcHandle
);
```

phRpcHandle: MUST be a pointer to a variable that is the remote notification handle from which the user no longer wants to receive notifications.

Return Values: This method MUST return zero or a positive value to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST verify that the **phRpcHandle** parameter is not null, and that it points to a non-null **RMTNTFY_HANDLE**.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client.

Processing and Response Requirements: If the operation is successful, the server MUST make the following changes to printer storage objects before returning:

- Remove the notification listener from the list of listeners associated with the printer object.
- Delete the RPC handle and corresponding notification state.

3.1.4.9.3 RpcSyncRefreshRemoteNotifications (Opnum 60)

RpcSyncRefreshRemoteNotifications gets notification information for all requested members. This SHOULD be called by a client if the "RemoteNotifyData Flags" key in the **RpcPrintPropertiesCollection** instance, which was returned as part of the notification from an **RpcAsyncGetRemoteNotifications** call, has the **PRINTER_NOTIFY_INFO_DISCARDED** bit set. **PRINTER_NOTIFY_INFO_DISCARDED** is defined in [MS-RPRN] section **2.2.3.7**.

```
HRESULT RpcSyncRefreshRemoteNotifications(
  [in] RMTNTFY_HANDLE hRpcHandle,
  [in] RpcPrintPropertiesCollection* pNotifyFilter,
  [out] RpcPrintPropertiesCollection** ppNotifyData
);
```

hRpcHandle: This parameter MUST be a remote notification handle that MUST have been opened by using **RpcSyncRegisterForRemoteNotifications (section 3.1.4.9.1)**.

pNotifyFilter: This parameter MUST be a pointer to an **RpcPrintPropertiesCollection** (section 2.2.4) instance that MUST contain the caller-specified notification filter settings.

ppNotifyData: This parameter MUST be a pointer to a variable that MUST receive a pointer to an **RpcPrintPropertiesCollection** instance that contains the notification data.

Return Values: This method MUST return zero or a positive value to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST validate parameters as follows:

- The **hRpcHandle** parameter MUST NOT be null.
- The **pNotifyFilter** parameter MUST point to an **RpcPrintPropertiesCollection** instance that has all the name-value pairs required to get notification data.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST process the method call by:

- Storing the notification data requested by the client in the **RpcPrintPropertiesCollection** structure pointed to by **ppNotifyData**.
- Returning a response that MUST contain the status of the operation.

If the operation is successful, the server MUST make the following changes to printer object data before returning the response:

- Store the notification synchronization value in the **RpcPrintPropertiesCollection** instance pointed to by **pNotifyFilter**, which corresponds to the "RemoteNotifyFilter Color" key, so that the client can use it in **RpcAsyncGetRemoteNotifications** calls.
- Delete the notification data associated with the notification handle that has been successfully sent.

3.1.4.9.4 RpcAsyncGetRemoteNotifications (Opnum 61)

A client uses **RpcAsyncGetRemoteNotifications** to poll the print server for specified change notifications. A call to this method is suspended until the server has a new change notification for the client. Subsequently, the client calls this method again to poll for additional notifications from the server.

```
HRESULT RpcAsyncGetRemoteNotifications(
  [in] RMTNTFY_HANDLE hRpcHandle,
  [out] RpcPrintPropertiesCollection** ppNotifyData
);
```

hRpcHandle: This parameter MUST be a remote notification handle that MUST have been opened by using **RpcSyncRegisterForRemoteNotifications (section 3.1.4.9.1)**.

ppNotifyData: This parameter MUST be a pointer to a variable that MUST receive a pointer to an **RpcPrintPropertiesCollection (section 2.2.4)** instance that contains the notification data.

Return Values: This method MUST return a positive value or zero, to indicate successful completion or a negative value to indicate failure. The client MUST treat any negative return value as a fatal error.

Exceptions Thrown: This method MUST NOT throw any exceptions other than those that are thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Parameter Validation Requirements: Upon receiving this method call, the server MUST verify that the **hRpcHandle** parameter is not null.

If parameter validation fails, the server MUST return immediately, with a failure indication in its response to the client.

Processing and Response Requirements: Otherwise, the server MUST process the method call by:

- Checking whether any change notification data is available on this notification handle.
- If change notification data is not available, MUST wait until the specified printer object changes and notification data becomes available.
- Returning a response that MUST contain the status of the operation.

If the operation is successful, the server MUST make the following changes to printer storage objects before returning the response:

- Store the notification data requested by the client that corresponds to the "RemoteNotifyData Info" key in the **RpcPrintPropertiesCollection** instance pointed to by **ppNotifyData**.
- Store the notification synchronization value in the **RpcPrintPropertiesCollection** instance pointed to by **ppNotifyData** that corresponds to the "RemoteNotifyData Color" key (see section 2.2.3 for more information). The latest synchronization value was sent by the client in a prior call to **RpcSyncRefreshRemoteNotifications**.
- Store a value specifying the members that have changed, which corresponds to the "RemoteNotifyData Flags" key in the **RpcPrintPropertiesCollection** instance pointed to by **ppNotifyData**.
- Delete the notification data associated with the notification handle that has been successfully sent.

3.1.5 Timer Events

No protocol timer events are required on the server other than the timers that are required in the underlying RPC protocol.

3.1.6 Other Local Events

No local events are maintained on the server other than the events that are maintained in the underlying RPC protocol.

3.2 IRemoteWinspool Client Details

3.2.1 Abstract Data Model

No abstract data model is required.

3.2.2 Timers

No protocol timers are required on the client—other than the timers that are required in the underlying RPC protocol.

3.2.3 Initialization

The Print System Asynchronous Remote Protocol client MUST perform the following initialization actions:

The client MUST first create an RPC binding handle to the server **RPC endpoint** in order to call an RPC method. Binding handles are either context handles that are used across multiple calls to the server, or handles that are bound to a single call to the server. Binding handles, as used with a Print System Asynchronous Remote Protocol client, are as specified in [C706] section 2.3.

The client MUST reuse a binding handle for multiple invocations when creating a print job, as in a call to **RpcAsyncOpenPrinter (Opnum 0)** followed by multiple calls to **RpcAsyncStartPagePrinter (Opnum 11)** and **RpcAsyncWritePrinter (Opnum12)**.

The client SHOULD reuse a binding handle for multiple invocations, as in a call to **RpcAsyncOpenPrinter (Opnum 0)** followed by multiple calls to **RpcAsyncGetPrinter (Opnum 9)**, **RpcAsyncGetPrinterData (Opnum 16)**, or **RpcAsyncSetPrinter (Opnum 8)**. However, for name-based calls, the client SHOULD create a separate binding handle for each method invocation.

The client SHOULD create an authenticated RPC binding handle for the best possible security. If an authenticated binding handle is established, the client SHOULD attempt to establish the strongest possible authentication. If this attempt fails but the binding remains valid—that is, the remote RPC implementation server is listening on the authenticated **endpoint** but does not support the required authentication mechanism—the client MAY<15> choose either to ignore the error for compatibility or terminate its attempt to invoke the RPC method.

The client MUST ensure that all method calls occur with causal ordering, as specified in [MS-RPCE] section 3.1.1.4.1.

3.2.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.

This protocol MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with nonzero conformant value, as specified in [MS-RPCE] section 3.

The client SHOULD NOT make any decisions based on the errors that are returned from the RPC server, but SHOULD notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client except for what is required in the underlying RPC protocol.<16>

3.2.5 Timer Events

No protocol timer events are required on the client other than the timers that are required in the underlying RPC protocol.

3.2.6 Other Local Events

No local events are maintained on the client other than the events that are maintained in the underlying RPC protocol.

4 Protocol Examples

The following sections describe several operations as they are used in common scenarios to illustrate the function of the Print System Asynchronous Remote Protocol.

Each subsection shows a sequence diagram, including the necessary steps for a common scenario.

4.1 Adding a Printer to a Server

The client SHOULD follow the steps described in [MS-RPRN] section 4.1 and apply substitutions described in [MS-RPRN] section 3.1.4.1.

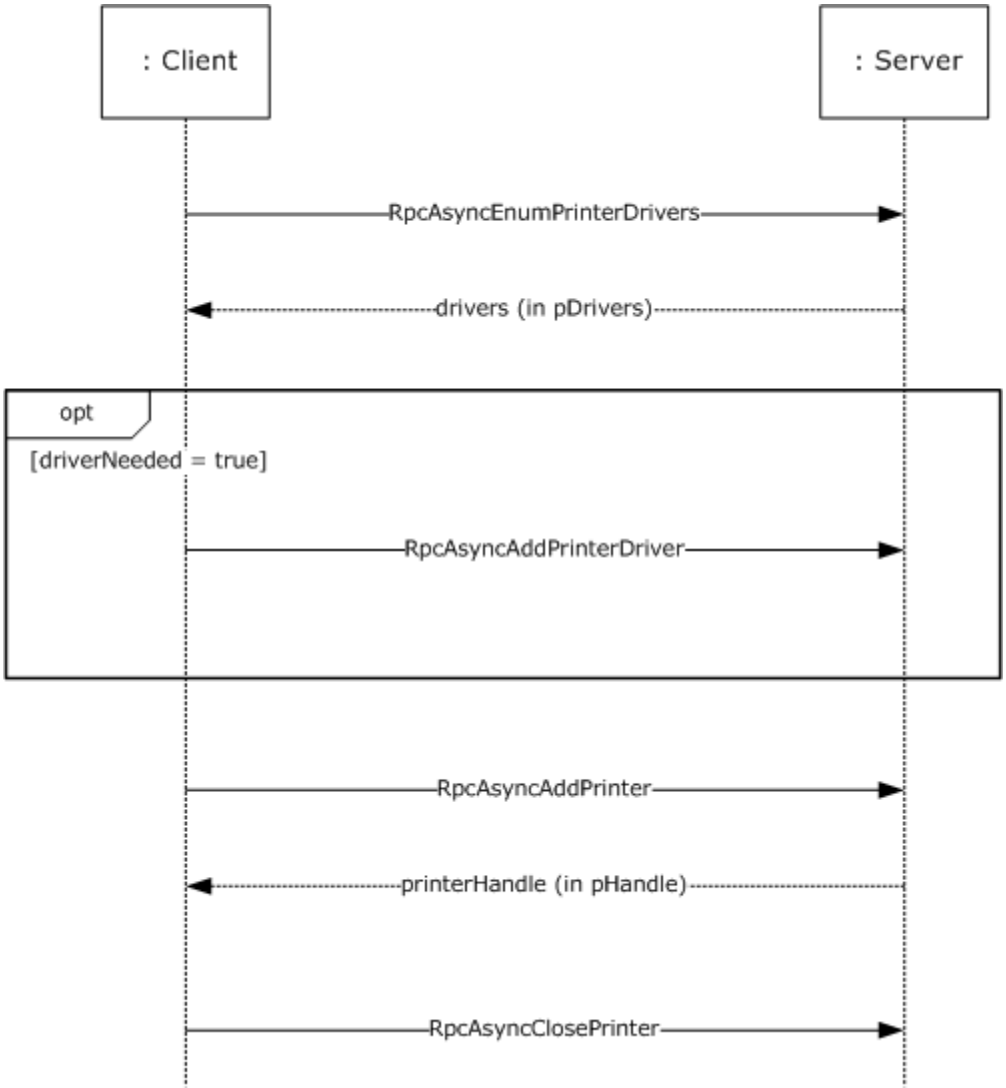

Figure 4: Adding a printer to a server

4.2 Adding a Printer Driver to a Server

The client SHOULD follow the steps described in [MS-RPRN] section 4.2 and apply substitutions described in [MS-RPRN] section 3.1.4.1.

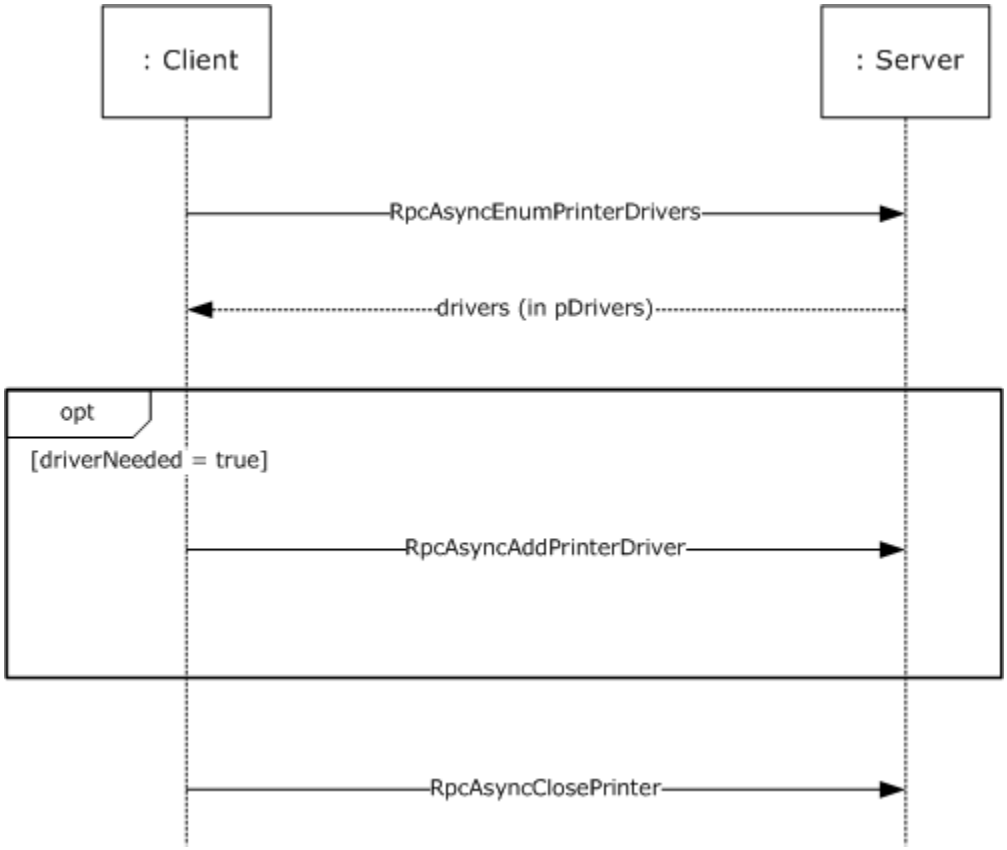

Figure 5: Adding a printer driver to a server

4.3 Enumerating and Managing Printers on a Server

The client SHOULD follow the steps described in [MS-RPRN] section 4.3 and apply substitutions described in [MS-RPRN] section 3.1.4.1.

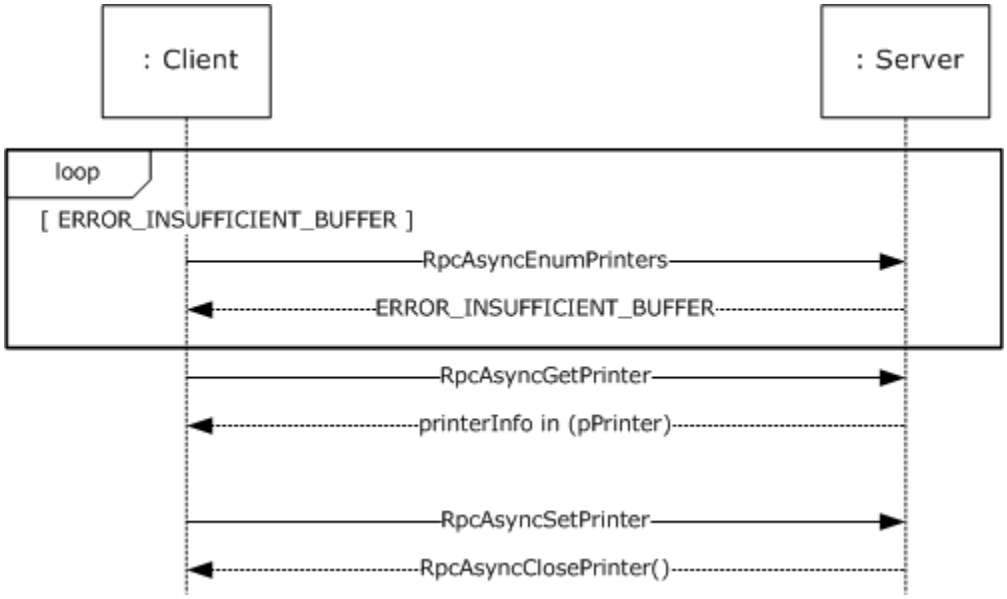

Figure 6: Enumerating printers on a server

4.4 Printing a Job on a Server by Using RpcAsyncAddJob, Job Scheduling, and Setting Job Information

The client SHOULD follow the steps described in [MS-RPRN] section 4.4 and apply substitutions described in [MS-RPRN] section 3.1.4.1.

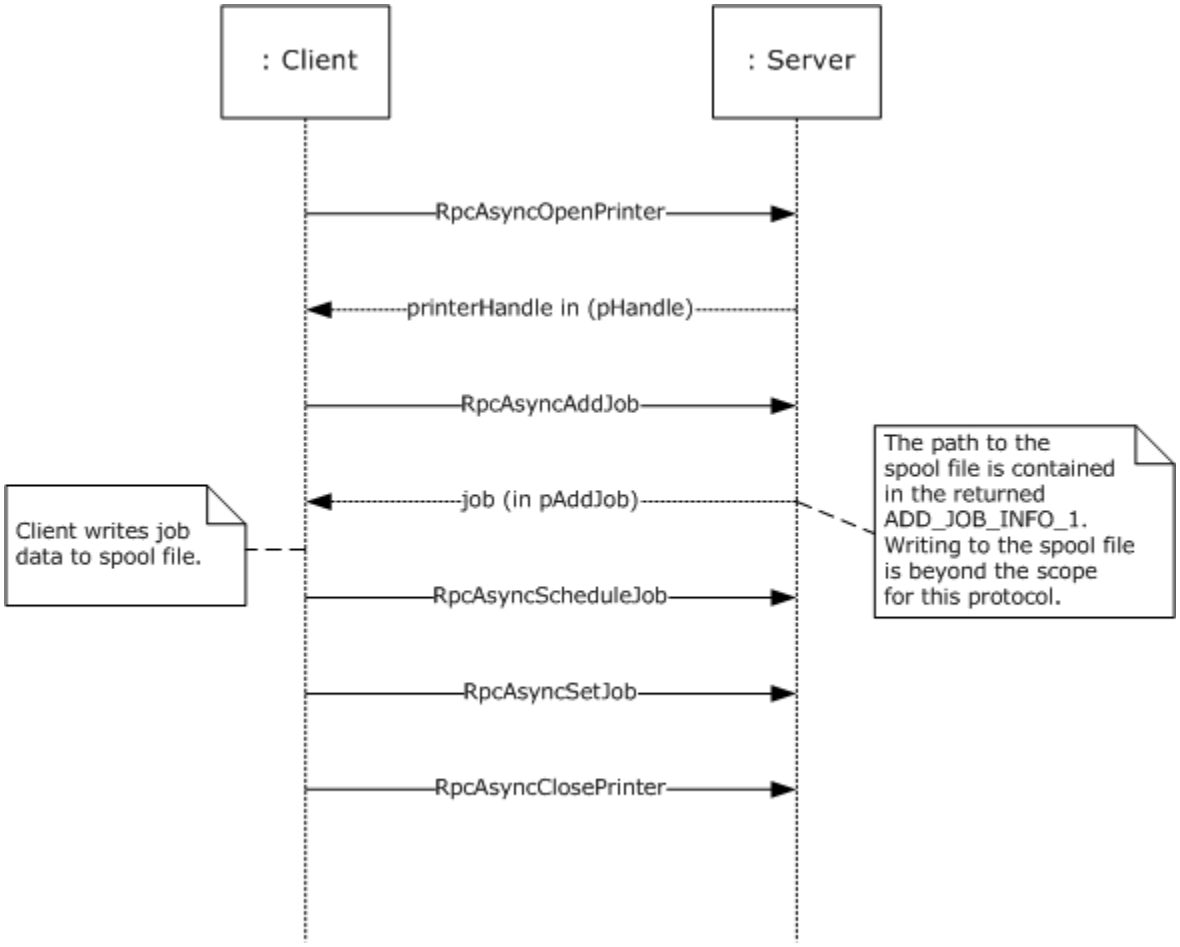

Figure 7: Printing a job on a server by using RpcAsyncAddJob, job scheduling, and setting job information

4.5 Enumerating Jobs and Modifying Job Settings on a Server

The client SHOULD follow the steps described in [MS-RPRN] section 4.5 and apply substitutions described in [MS-RPRN] section 3.1.4.1.

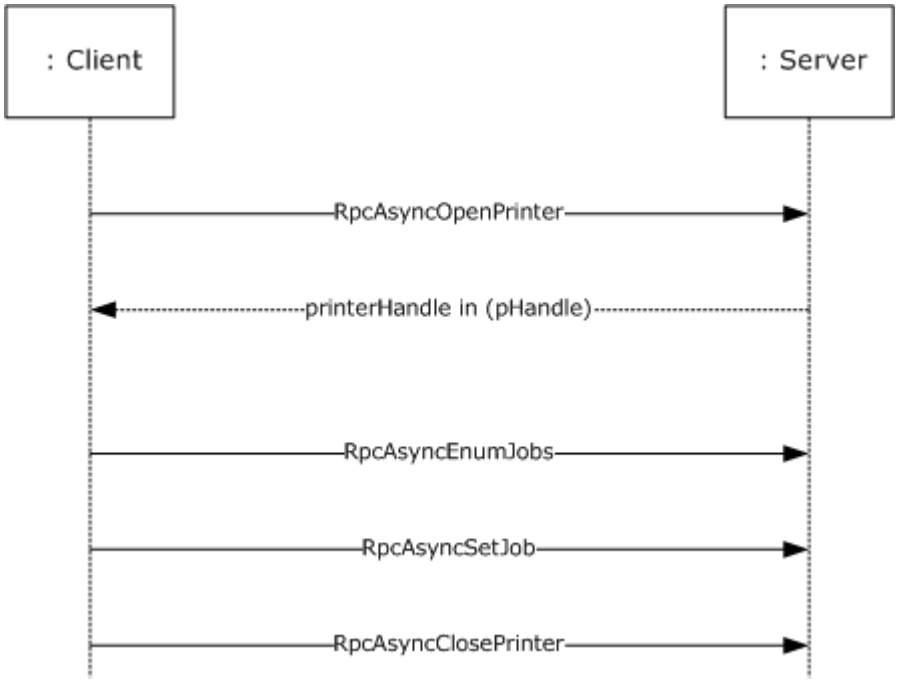

Figure 8: Enumerating jobs and modifying job settings on a server

4.6 Receiving Notifications from a Server on Different Printing Events

In order to receive notifications about state changes of print servers, printers and **print jobs**, the client SHOULD perform the following steps:

1. Open the print server or printer using **RpcAsyncOpenPrinter**.
2. Register for change notifications using **RpcSyncRegisterForRemoteNotifications**, specifying the type of notifications the client is interested in.
3. While the client is interested in state change notifications, it SHOULD call the server's **RpcAsyncGetRemoteNotifications** method. The method call will not return until there is a new state change notification.
4. If the server sets the **PRINTER_NOTIFY_INFO_DISCARDED** flag in the data returned from **RpcAsyncGetRemoteNotifications**, the client SHOULD call **RpcSyncRefreshRemoteNotifications** to obtain updated state information.
5. When the client is no longer interested in state notifications, it SHOULD cancel any outstanding **RpcAsyncOpenPrinter** calls using the underlying RPC mechanism, and then unregister from change notifications by calling **RpcSyncUnRegisterForRemoteNotifications** with the handle previously obtained from **RpcSyncRegisterForRemoteNotifications**.
6. The client SHOULD close the printer or print server handle by calling **RpcAsyncClosePrinter**.

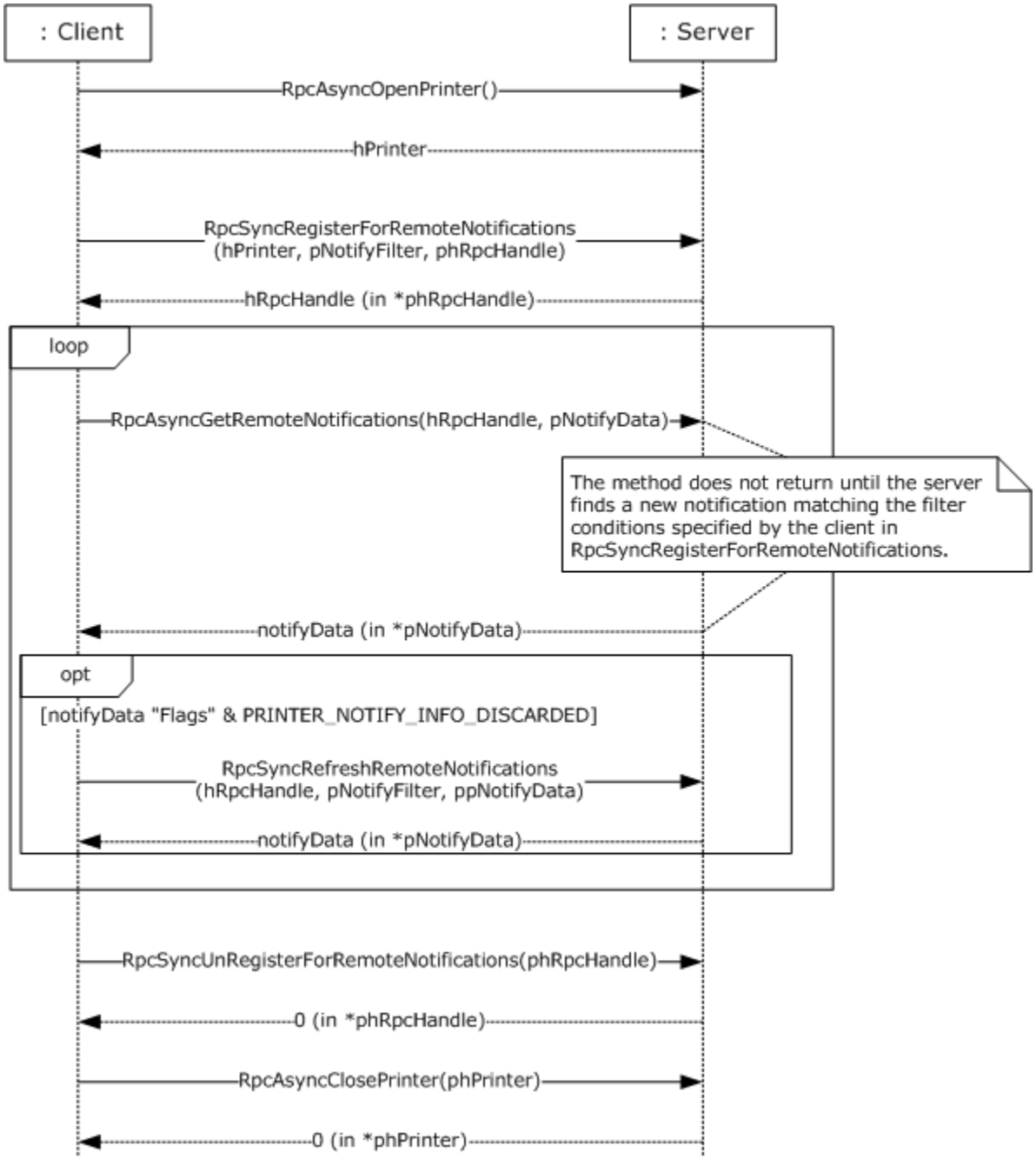

Figure 9: Receiving notifications from a server on different printing events

5 Security

The following sections specify security considerations for implementers of the Print System Asynchronous Remote Protocol.

5.1 Security Considerations for Implementers

This protocol currently has no security considerations for implementers.

5.2 Index of Security Parameters

Security considerations for both unauthenticated RPC and authenticated RPC are as specified in [C706] section 2.3 and section 5.

A Print System Asynchronous Remote Protocol client MAY<17> fail over to unauthenticated RPC by using the [MS-RPRN] protocol when authenticated RPC fails for backward compatibility. Unauthenticated RPC is not as secure as authenticated RPC; the client SHOULD either audit or support this automatic **failover** only when it is explicitly specified.

6 Appendix A: Full IDL

For ease of implementation, the full standalone IDL is provided below. Some of the data types and structures used by this protocol are defined in other documents. In order to be standalone, those types and structures, from [MS-DTYP] and [MS-RPRN], are included below.

```
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 CCHDEVICENAME            32
#define CCHFORMNAME              32

// [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;

// [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[CCHDEVICENAME];
```

```
    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[CCHFORMNAME];

    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;
    DEVMODE* pDevMode;
    [string] wchar_t* pStatus;
    SECURITY_DESCRIPTOR* 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;
    DEVMODE* pDevMode;
    [string] wchar_t* pStatus;
    SECURITY_DESCRIPTOR* 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* pName;
} 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 {
    wchar_t* pName;
    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;
    DEVMODE* pDevMode;
    [string] wchar_t* pSepFile;
    [string] wchar_t* pPrintProcessor;
    [string] wchar_t* pDatatype;
    [string] wchar_t* pParameters;
    SECURITY_DESCRIPTOR* 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 {
    SECURITY_DESCRIPTOR* 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 {
    DEVMODE* pDevMode;
} PRINTER_INFO_8;

typedef struct _PRINTER_INFO_9 {
    DEVMODE* pDevMode;
} PRINTER_INFO_9;

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

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* Level1;
      [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)]
        unsigned __int64 pClientInfo2;
      [case(3)]
        SPLCLIENT_INFO_3* pClientInfo3;
    } 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;

// [MS-DTYP] common data types
typedef DWORD HRESULT;

// [MS-DTYP] common structures
typedef struct {
    unsigned long Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char Data4[8];
} GUID,
 *PGUID;

[
    uuid(76F03F96-CDFD-44fc-A22C-64950A001209),
    version(1.0),
#ifdef __midl
        ms_union,
#endif // __midl
    pointer_default(unique)
]
interface IRemoteWinspool {

// [MS-PAR] enumerations
typedef enum {
    kPropertyTypeString = 1,
    kPropertyTypeInt32,
    kPropertyTypeInt64,
    kPropertyTypeByte,
    kPropertyTypeTime,
    kPropertyTypeDevMode,
    kPropertyTypeSD,
    kPropertyTypeNotificationReply,
    kPropertyTypeNotificationOptions,
} EPrintPropertyType;

// [MS-PAR] data types
typedef [context_handle] void *RMTNTFY_HANDLE;

// [MS-PAR] structures
typedef struct _NOTIFY_REPLY_CONTAINER {
    RPC_V2_NOTIFY_INFO* pInfo;
} NOTIFY_REPLY_CONTAINER;

typedef struct _NOTIFY_OPTIONS_CONTAINER {
    RPC_V2_NOTIFY_OPTIONS* pOptions;
} NOTIFY_OPTIONS_CONTAINER;

typedef struct {
    EPrintPropertyType   ePropertyType;

    [switch_type(EPrintPropertyType), switch_is(ePropertyType)]
    union {
```

```
        [case(kPropertyTypeString)]
             [string] wchar_t*        propertyString;
        [case(kPropertyTypeInt32)]
             long                     propertyInt32;
        [case(kPropertyTypeInt64)]
             __int64                  propertyInt64;
        [case(kPropertyTypeByte)]
             BYTE                     propertyByte;
        [case(kPropertyTypeTime)]
             SYSTEMTIME_CONTAINER     propertyTimeContainer;
        [case(kPropertyTypeDevMode)]
             DEVMODE_CONTAINER        propertyDevModeContainer;
        [case(kPropertyTypeSD)]
             SECURITY_CONTAINER       propertySDContainer;
        [case(kPropertyTypeNotificationReply)]
             NOTIFY_REPLY_CONTAINER   propertyReplyContainer;
        [case(kPropertyTypeNotificationOptions)]
             NOTIFY_OPTIONS_CONTAINER propertyOptionsContainer;
    } value;
} RpcPrintPropertyValue;

typedef struct {
    [string] wchar_t*  propertyName;
    RpcPrintPropertyValue propertyValue;
} RpcPrintNamedProperty;

typedef struct {
    [range(0, 50)]
    unsigned long numberOfProperties;

    [size_is(numberOfProperties),unique]
    RpcPrintNamedProperty*  propertiesCollection;
}RpcPrintPropertiesCollection;

typedef struct _CORE_PRINTER_DRIVER {
    GUID         CoreDriverGUID;
    FILETIME     ftDriverDate;
    DWORDLONG    dwlDriverVersion;
    wchar_t      szPackageID[260];
} CORE_PRINTER_DRIVER;

// [MS-PAR] methods
DWORD
RpcAsyncOpenPrinter(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pPrinterName,
   [out] PRINTER_HANDLE*pHandle,
   [in, string, unique] wchar_t* pDatatype,
   [in] DEVMODE_CONTAINER* pDevModeContainer,
   [in] DWORD AccessRequired,
   [in] SPLCLIENT_CONTAINER* pClientInfo
);

DWORD
RpcAsyncAddPrinter(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in] PRINTER_CONTAINER* pPrinterContainer,
```

```
    [in] DEVMODE_CONTAINER* pDevModeContainer,
    [in] SECURITY_CONTAINER* pSecurityContainer,
    [in] SPLCLIENT_CONTAINER* pClientInfo,
    [out] PRINTER_HANDLE*pHandle
);

DWORD
RpcAsyncSetJob(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD JobId,
    [in, unique] JOB_CONTAINER* pJobContainer,
    [in] DWORD Command
);

DWORD
RpcAsyncGetJob(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD JobId,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf)] unsigned char* pJob,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncEnumJobs(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD FirstJob,
    [in] DWORD NoJobs,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf)] unsigned char* pJob,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded,
    [out] DWORD* pcReturned
);

DWORD
RpcAsyncAddJob(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf)] unsigned char* pAddJob,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncScheduleJob(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD JobId
);

DWORD
RpcAsyncDeletePrinter(
    [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAsyncSetPrinter(
```

```
    [in] PRINTER_HANDLE hPrinter,
    [in] PRINTER_CONTAINER* pPrinterContainer,
    [in] DEVMODE_CONTAINER* pDevModeContainer,
    [in] SECURITY_CONTAINER* pSecurityContainer,
    [in] DWORD Command
);

DWORD
RpcAsyncGetPrinter(
    [in] PRINTER_HANDLE hPrinter,
    [in] DWORD Level,
    [in, out, unique, size_is(cbBuf)] unsigned char* pPrinter,
    [in] DWORD cbBuf,
    [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncStartDocPrinter(
    [in] PRINTER_HANDLE hPrinter,
    [in] DOC_INFO_CONTAINER* pDocInfoContainer,
    [out] DWORD* pJobId
);

DWORD
RpcAsyncStartPagePrinter(
    [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAsyncWritePrinter(
    [in] PRINTER_HANDLE hPrinter,
    [in, size_is(cbBuf)] unsigned char* pBuf,
    [in] DWORD cbBuf,
    [out] DWORD* pcWritten
);

DWORD
RpcAsyncEndPagePrinter(
    [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAsyncEndDocPrinter(
    [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAsyncAbortPrinter(
    [in] PRINTER_HANDLE hPrinter
);

DWORD
RpcAsyncGetPrinterData(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar_t* pValueName,
    [out] DWORD* pType,
    [out, size_is(nSize)] unsigned char* pData,
    [in] DWORD nSize,
```

```
   [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncGetPrinterDataEx(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] const wchar_t* pKeyName,
   [in, string] const wchar_t* pValueName,
   [out] DWORD* pType,
   [out, size_is(nSize)] unsigned char* pData,
   [in] DWORD nSize,
   [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncSetPrinterData(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] wchar_t* pValueName,
   [in] DWORD Type,
   [in, size_is(cbData)] unsigned char* pData,
   [in] DWORD cbData
);

DWORD
RpcAsyncSetPrinterDataEx(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] const wchar_t* pKeyName,
   [in, string] const wchar_t* pValueName,
   [in] DWORD Type,
   [in, size_is(cbData)] unsigned char* pData,
   [in] DWORD cbData
);

DWORD
RpcAsyncClosePrinter(
   [in, out] PRINTER_HANDLE* phPrinter
);

DWORD
RpcAsyncAddForm(
   [in] PRINTER_HANDLE hPrinter,
   [in] FORM_CONTAINER* pFormInfoContainer
);

DWORD
RpcAsyncDeleteForm(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] wchar_t* pFormName
);

DWORD
RpcAsyncGetForm(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] wchar_t* pFormName,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pForm,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded
```

```
);

DWORD
RpcAsyncSetForm(
   [in] PRINTER_HANDLE hPrinter,
   [in, string] wchar_t* pFormName,
   [in] FORM_CONTAINER* pFormInfoContainer
);

DWORD
RpcAsyncEnumForms(
   [in] PRINTER_HANDLE hPrinter,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pForm,
   [in]  DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAsyncGetPrinterDriver(
   [in] PRINTER_HANDLE hPrinter,
   [in, unique, string] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pDriver,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [in] DWORD dwClientMajorVersion,
   [in] DWORD dwClientMinorVersion,
   [out] DWORD* pdwServerMaxVersion,
   [out] DWORD* pdwServerMinVersion
);

DWORD
RpcAsyncEnumPrinterData(
   [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)] unsigned char* pData,
   [in]  DWORD cbData,
   [out] DWORD* pcbData
);

DWORD
RpcAsyncEnumPrinterDataEx(
   [in]  PRINTER_HANDLE hPrinter,
   [in, string] const wchar_t* pKeyName,
   [out, size_is(cbEnumValues)] unsigned char* pEnumValues,
   [in]  DWORD cbEnumValues,
   [out] DWORD* pcbEnumValues,
   [out] DWORD* pnEnumValues
);

DWORD
RpcAsyncEnumPrinterKey(
```

```
    [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
RpcAsyncDeletePrinterData(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] wchar_t* pValueName
);

DWORD
RpcAsyncDeletePrinterDataEx(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName,
    [in, string] const wchar_t* pValueName
);

DWORD
RpcAsyncDeletePrinterKey(
    [in] PRINTER_HANDLE hPrinter,
    [in, string] const wchar_t* pKeyName
);

DWORD
RpcAsyncXcvData(
    [in] PRINTER_HANDLE hXcv,
    [in, string] const wchar_t* pszDataName,
    [in, size_is(cbInputData)] unsigned char* pInputData,
    [in] DWORD cbInputData,
    [out, size_is(cbOutputData)] unsigned char* pOutputData,
    [in] DWORD cbOutputData,
    [out] DWORD* pcbOutputNeeded,
    [in, out] DWORD* pdwStatus
);

DWORD
RpcAsyncSendRecvBidiData (
    [in] PRINTER_HANDLE hPrinter,
    [in,string,unique] const wchar_t* pAction,
    [in] RPC_BIDI_REQUEST_CONTAINER* pReqData,
    [out] RPC_BIDI_RESPONSE_CONTAINER** ppRespData);

DWORD
RpcAsyncCreatePrinterIC(
    [in] PRINTER_HANDLE hPrinter,
    [out] GDI_HANDLE *pHandle,
    [in] DEVMODE_CONTAINER* pDevModeContainer
);

DWORD
RpcAsyncPlayGdiScriptOnPrinterIC(
    [in] GDI_HANDLE hPrinterIC,
    [in, size_is(cIn)] unsigned char* pIn,
    [in] DWORD cIn,
    [out, size_is(cOut)] unsigned char* pOut,
```

```
   [in] DWORD cOut,
   [in] DWORD ul
);

DWORD
RpcAsyncDeletePrinterIC(
   [in, out] GDI_HANDLE* phPrinterIC
);

DWORD
RpcAsyncEnumPrinters(
   [in] handle_t hRemoteBinding,
   [in] DWORD Flags,
   [in, string, unique] wchar_t* Name,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pPrinterEnum,
   [in]  DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAsyncAddPrinterDriver(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in] DRIVER_CONTAINER* pDriverContainer,
   [in] DWORD dwFileCopyFlags
);

DWORD
RpcAsyncEnumPrinterDrivers(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, unique, string] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pDrivers,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAsyncGetPrinterDriverDirectory(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, unique, string] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pDriverDirectory,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncDeletePrinterDriver(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, string] wchar_t* pEnvironment,
   [in, string] wchar_t* pDriverName
```

```
);

DWORD
RpcAsyncDeletePrinterDriverEx(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, string] wchar_t* pEnvironment,
   [in, string] wchar_t* pDriverName,
   [in] DWORD dwDeleteFlag,
   [in] DWORD dwVersionNum
);

DWORD
RpcAsyncAddPrintProcessor(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, string] wchar_t* pEnvironment,
   [in, string] wchar_t* pPathName,
   [in, string] wchar_t* pPrintProcessorName
);

DWORD
RpcAsyncEnumPrintProcessors(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, unique, string] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char*
       pPrintProcessorInfo,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAsyncGetPrintProcessorDirectory(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, unique, string] wchar_t* pEnvironment,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char*
       pPrintProcessorDirectory,
   [in] DWORD cbBuf,
   [out] DWORD* pcbNeeded
);

DWORD
RpcAsyncEnumPorts(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pPort,
   [in]  DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
```

```
RpcAsyncEnumMonitors(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pMonitor,
   [in]  DWORD cbBuf,
   [out] DWORD* pcbNeeded,
   [out] DWORD* pcReturned
);

DWORD
RpcAsyncAddPort(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in] PORT_CONTAINER* pPortContainer,
   [in] PORT_VAR_CONTAINER* pPortVarContainer,
   [in, string] wchar_t* pMonitorName
);

DWORD
RpcAsyncSetPort(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, string, unique] wchar_t* pPortName,
   [in] PORT_CONTAINER* pPortContainer
);

DWORD
RpcAsyncAddMonitor(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* Name,
   [in] MONITOR_CONTAINER* pMonitorContainer
);

DWORD
RpcAsyncDeleteMonitor(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* Name,
   [in, unique, string] wchar_t* pEnvironment,
   [in, string] wchar_t*pMonitorName
);

DWORD
RpcAsyncDeletePrintProcessor(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* Name,
   [in, unique, string] wchar_t* pEnvironment,
   [in, string] wchar_t*pPrintProcessorName
);

DWORD
RpcAsyncEnumPrintProcessorDatatypes(
   [in] handle_t hRemoteBinding,
   [in, string, unique] wchar_t* pName,
   [in, unique, string] wchar_t* pPrintProcessorName,
   [in] DWORD Level,
   [in, out, unique, size_is(cbBuf)] unsigned char* pDatatypes,
   [in]  DWORD cbBuf,
```

```
    [out] DWORD* pcbNeeded,
    [out] DWORD* pcReturned
);

DWORD
RpcAsyncAddPerMachineConnection(
    [in] handle_t hRemoteBinding,
    [in, string, unique] wchar_t* pServer,
    [in, string] const wchar_t* pPrinterName,
    [in, string] const wchar_t* pPrintServer,
    [in, string] const wchar_t* pProvider
);

DWORD
RpcAsyncDeletePerMachineConnection(
    [in] handle_t hRemoteBinding,
    [in, string, unique] wchar_t* pServer,
    [in, string] const wchar_t* pPrinterName
);

DWORD
RpcAsyncEnumPerMachineConnections(
    [in] handle_t hRemoteBinding,
    [in, string, unique] wchar_t* pServer,
    [in, out, unique, size_is(cbBuf)] unsigned char* pPrinterEnum,
    [in]  DWORD cbBuf,
    [out] DWORD* pcbNeeded,
    [out] DWORD* pcReturned
);

HRESULT
RpcSyncRegisterForRemoteNotifications(
    [in]    PRINTER_HANDLE                  hPrinter,
    [in]    RpcPrintPropertiesCollection*   pNotifyFilter,
    [out]   RMTNTFY_HANDLE*                 phRpcHandle
);

HRESULT
RpcSyncUnRegisterForRemoteNotifications(
    [in, out]   RMTNTFY_HANDLE*             phRpcHandle
);

HRESULT
RpcSyncRefreshRemoteNotifications(
    [in]    RMTNTFY_HANDLE                  hRpcHandle,
    [in]    RpcPrintPropertiesCollection*   pNotifyFilter,
    [out]   RpcPrintPropertiesCollection**  ppNotifyData
);

HRESULT
RpcAsyncGetRemoteNotifications(
    [in]    RMTNTFY_HANDLE                  hRpcHandle,
    [out]   RpcPrintPropertiesCollection**  ppNotifyData
);

HRESULT
RpcAsyncInstallPrinterDriverFromPackage(
    [in]                        handle_t  hRemoteBinding,
```

```
    [in, string, unique] const wchar_t*  pszServer,
    [in, string, unique] const wchar_t*  pszInfPath,
    [in, string]         const wchar_t*  pszDriverName,
    [in, string]         const wchar_t*  pszEnvironment,
    [in]                       DWORD     dwFlags
);

HRESULT
RpcAsyncUploadPrinterDriverPackage(
    [in]                       handle_t  hRemoteBinding,
    [in, string, unique] const wchar_t*  pszServer,
    [in, string]         const wchar_t*  pszInfPath,
    [in, string]         const wchar_t*  pszEnvironment,
    [in]                       DWORD     dwFlags,
    [in, out, unique, size_is(*pcchDestInfPath)]
                               wchar_t*  pszDestInfPath,
    [in, out]                  DWORD*    pcchDestInfPath
);

HRESULT
RpcAsyncGetCorePrinterDrivers(
    [in]                       handle_t  hRemoteBinding,
    [in, string, unique] const wchar_t*  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
);

HRESULT
RpcAsyncCorePrinterDriverInstalled(
    [in]                       handle_t  hRemoteBinding,
    [in, string, unique] const wchar_t*  pszServer,
    [in, string]         const wchar_t*  pszEnvironment,
    [in]                       GUID      CoreDriverGUID,
    [in]                       FILETIME  ftDriverDate,
    [in]                       DWORDLONG dwlDriverVersion,
    [out]       int*pbDriverInstalled
);

HRESULT
RpcAsyncGetPrinterDriverPackagePath(
    [in]                       handle_t  hRemoteBinding,
    [in, string, unique] const wchar_t*  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
);

HRESULT
RpcAsyncDeletePrinterDriverPackage(
```

```
    [in]                        handle_t  hRemoteBinding,
    [in, string, unique] const wchar_t*  pszServer,
    [in, string]         const wchar_t*  pszInfPath,
    [in, string]         const wchar_t*  pszEnvironment
);

DWORD
RpcAsyncReadPrinter(
    [in]                    PRINTER_HANDLE  hPrinter,
    [out, size_is(cbBuf)]   unsigned char*  pBuf,
    [in]                    DWORD           cbBuf,
    [out]                   DWORD*          pcNoBytesRead
);

DWORD
RpcAsyncResetPrinter(
    [in]                    PRINTER_HANDLE     hPrinter,
    [in, string, unique]    wchar_t*           pDatatype,
    [in]                    DEVMODE_CONTAINER* pDevModeContainer
);
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3.2: All Windows versions: Windows uses various spool file formats, such as **Enhanced Metafile (EMF) Spool Format** or **RAW format**. In Windows Vista and Windows Server 2008, the **XML Paper Specification** format may be used as well. For more information about these formats, see [MSDN-SPOOL] and [MSDN-XMLP], respectively.

<2> Section 1.8: All Windows versions: Windows implementations of this protocol use only the values that are specified in [MS-ERREF] section 4.2.

<3> Section 2.2.8: All Windows versions: The driver version is matched to the version portion of the **INF file** DriverVer member. For information on INF file syntax, see [MSDN-UINF].

<4> Section 3.1.4.2.7: All Windows versions: **pszInfPath** points to an INF file. For more information on INF file structure, see [MSDN-UINF].

<5> Section 3.1.4.2.8: All Windows versions: pszInfPath points to an INF file. For more information on INF file structure, see [MSDN-UINF].

<6> Section 3.1.4.2.8: All Windows versions: The IDL specifies the "in" attribute for this parameter by mistake, but it was not removed, for interoperability with Windows servers. The value is never used as an input parameter.

<7> Section 3.1.4.2.8: All Windows versions: Printer drivers are described by INF files. For more information, see [MSDN-UINF].

<8> Section 3.1.4.2.9: All Windows versions: The IDs used are the GUID string representations of 128-bit GUIDs in the form of a **GUIDString**, as defined in [MS-GLOS].

<9> Section 3.1.4.2.10: All Windows versions: The driver date is matched to the date portion of the INF DriverVer member. For information on INF file syntax, see [MSDN-UINF].

<10> Section 3.1.4.2.10: All Windows versions: The driver version is matched to the version portion of the INF DriverVer member. For information on INF file syntax, see [MSDN-UINF].

<11> Section 3.1.4.2.11: All Windows versions: The Language string is specified using the identifiers specified for the 'Locale Name' in [MSDN-MUI].

<12> Section 3.1.4.2.11: 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].

<13> Section 3.1.4.2.11: All Windows versions: If the parameter is zero, Windows fills in the variable pointed to by **pcchRequiredSize** with the valid size.

<14> Section 3.1.4.2.12: All Windows versions: **pszInfPath** points to a string containing the path of an INF file. For more information on INF file structure, see [MSDN-UINF].

<15> Section 3.2.3: All Windows versions: Determining whether an authenticated RPC binding is available is done by method invocation. The client creates the binding handle, verifies the security capability of the remote server, and invokes the method.

For authenticated RPC, the client invokes the rpc_mgmt_inq_princ_name method of the Remote Management Interface (as specified in [C706] appendix Q and [MS-RPCE] section **2.2.1.3.4**, to retrieve the principal name "princ_name" for the SPNEGO authentication service. This invocation is done prior to every Print System Asynchronous Remote Protocol method call. If this invocation succeeds, authentication with the remote peer is deemed possible, and the RPC runtime is configured to use the SPNEGO security provider with the **RPC_C_AUTHN_GSS_NEGOTIATE** and **RPC_C_AUTHN_LEVEL_PKT_PRIVACY** flags, and the retrieved principal name, for the subsequent RPC method calls to the server.

Because this protocol is only supported on Windows Vista and Windows Server 2008 print servers, Windows Vista clients attempt to connect by using this protocol first. If the connection fails, Windows Vista clients revert to using the Print System Remote Protocol, as specified in [MS-RPRN].

<16> Section 3.2.4: All Windows versions: Clients ignore errors and pass them back to the invoker.

<17> Section 5.2: All Windows versions: The Windows print server follows a security model where the print server, print queue, and print job are securable resources. Each of the previously mentioned resources has an associated **SECURITY_DESCRIPTOR** structure, as specified in [MS-DTYP] section **2.4.6**, which contains the security information that is associated with a resource on the print server. The print server checks the RPC client's access to resources by comparing the security information that is associated with the caller against the security information that is represented by the resource's security descriptor.

Each RPC client has an associated access token containing the **security identifier** of the user making the RPC call. The security descriptor identifies the printing resource's owner and contains a **discretionary access control list (DACL)**. The DACL contains **access control entries (ACEs)** that specify the security identifier (SID) that identifies a user or a group of users and the access rights allowed, denied, or audited. For resources on a print server, the ACEs specify operations such as print, manage printers, and manage documents in a print queue.

The security descriptor that is associated with the print server or print queue controls the creation of the context handle that represents a PRINTER_HANDLE. It also controls the outcome of operations that use the PRINTER_HANDLE, from printing management to listening for notifications.

The security descriptor of a Windows print server is used to control the creation and deletion of print queues on the server and the installation of print system components, such as the printer driver, print processors, port monitors, or resources on the print server. The Windows print server security descriptor is not accessible to be modified by callers. In addition to being used to control the caller's access to resources, the Windows print server security descriptor is also used as "parent" in the creation of the print queue's security descriptor.

Note: The security descriptor of a Windows print server is different from the security descriptor that is applied on the **spoolss** named pipe. The **spoolss** named pipe security descriptor controls the RPC client's access to make RPC calls to the print server. The Windows print server security descriptor is used to control the caller's permissions to perform various operations on the print server.

The print queue's security descriptor controls the setting of properties for the print queue, such as the port and driver that are used for printing, device settings, sharing, and security. The user is

allowed to manage, print, and so on. The printer security descriptor allows auditing operations, such as print, manage printers and documents, read and change permissions, and take ownership.

Each print job has an associated security descriptor, which is created by using the print queue's security descriptor as parent. The user who submitted the document for printing is the owner for the print job and has permissions to manage the print job during its lifetime.

When the caller opens a **PRINTER_HANDLE** structure for a specific printing resource, the caller must specify the access that is needed for the subsequent operations for which the handle is being opened, such as "administrate printer or server"; "use printer or print server for printing"; or "read, write, or administrate job". If the caller has the requested permissions, the print handle is created and can be used in subsequent calls.

Besides handle-based operations, the security descriptor is used for access checks when enumerations, driver package installation, or other non-handle-based operations are performed. The access checks are primarily about testing whether the initiator of the operation has enough use or administer privileges on the resource that is being targeted by that operation. For example, an access check might be whether the initiator of the operation has the privilege to pause a printer.

8 Index