[MS-RDPESP]:
Remote Desktop Protocol:
Serial Port Virtual Channel Extension

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	1.0	Major	Updated and revised the technical content.
10/23/2007	1.1	Minor	Updated the technical content.
11/30/2007	1.2	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	1.2.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Remote Desktop Protocol: Serial Port Virtual Channel Extension. This protocol is used to redirect serial and parallel ports from a **terminal client** to the **terminal server**. This allows the **server** to access **client** ports as if the connected devices were local to the server.

1.1 Glossary

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

Client
Server

The following terms are specific to this document:

Device Control: Driver-specific operations that can be performed on various drivers. Each DeviceIOControl is associated with an operation code (called IoCode) and optionally input and output buffers. Device drivers depending on the IoCode take various actions on the input and output buffers

Remote Device: A device remotely attached to a remote (or **client**) machine as opposed to a device physically attached to a machine.

Terminal Client: A **client** of a **terminal server**. A terminal client program that runs on the **client** machine.

Terminal Server: A **server** that provides a graphical user interface (GUI) of a desktop to terminal server **clients** allowing **clients** to remotely run applications on the **server**. The **server** transmits the GUI of the program to the **client**, and the **client** returns keyboard and mouse clicks to be processed by the **server**.

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.

[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-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification", June 2007.

[MS-RDPEFS] Microsoft Corporation, "Remote Desktop Protocol: File System Virtual Channel Extension", September 2007.

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

[MSFT-W2KDDK] "Microsoft Windows 2000 Driver Development Reference Kit, volumes 1-3", Microsoft Press, March 2000, ISBN 0735609292

[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

1.2.2 Informative References

[MSDN-PORTS] Microsoft Corporation, "Serial and Parallel ports", http://msdn2.microsoft.com/en-us/library/bb870477.aspx

1.3 Protocol Overview (Synopsis)

The Remote Desktop Protocol: Serial Port Virtual Channel Extension specifies the communication used to enable the redirection of ports between a terminal client and a terminal server. By redirecting ports from the terminal client to the terminal server, applications running on a server machine can access the **remote devices** attached to those ports.

1.3.1 Purpose of Device Redirection Extensions

This extension enables the redirection of serial and parallel port devices attached to the terminal client. With the redirection, such devices can then be accessed by the applications running on the server.

1.3.2 Protocol Initialization

This extension can be considered as a subprotocol within the Remote Desktop Protocol: File System Virtual Channel Extension as specified in [MS-RDPEFS]. It follows the initialization of the Remote Desktop Protocol: File System Virtual Channel Extension to enable port redirection.

1.4 Relationship to Other Protocols

This extension can be considered as a subprotocol within Remote Desktop Protocol: File System Virtual Channel Extension as specified in [MS-RDPEFS]. This extension extends the Remote Desktop Protocol: File System Virtual Channel Extension to enable port redirection.

1.5 Prerequisites/Preconditions

The Remote Desktop Protocol: Serial Port Virtual Channel Extension operates only after the Remote Desktop Protocol: File System Virtual Channel Extension transport, as specified in [MS-RDPEFS], is fully established.

1.6 Applicability Statement

The Remote Desktop Protocol: Serial Port Virtual Channel Extension is designed to be run within the context of a Remote Desktop Protocol virtual channel established between a client and server. This protocol extension is applicable when applications running on the terminal server need to access the ports physically located on a client machine.

1.7 Versioning and Capability Negotiation

This extension relies on the Remote Desktop Protocol: File System Virtual Channel Extension, as specified in [MS-RDPEFS], to perform basic versioning and capability negotiation.

1.8 Vendor-Extensible Fields

The Remote Desktop Protocol: Serial Port Virtual Channel Extension contains no vendor-extensible fields.

1.9 Standards Assignments

The Remote Desktop Protocol: Serial Port Virtual Channel Extension contains no standards assignments.

2 Messages

Because this is a subprotocol of Remote Desktop Protocol: File System Virtual Channel Extension, as specified in [MS-RDPEFS], this extension shares messages and common data types already specified in [MS-RDPEFS]. This section describes the messages and data types used by Remote Desktop Protocol: Serial Port Virtual Channel Extension.

2.1 Transport

All messages MUST be transported over an established Remote Desktop Protocol device extensions channel (as specified in [MS-RDPEFS] section 2.1).

2.2 Message Syntax

The following sections contain Remote Desktop Protocol: Serial Port Virtual Channel Extension message syntax.

2.2.1 Common Data Types

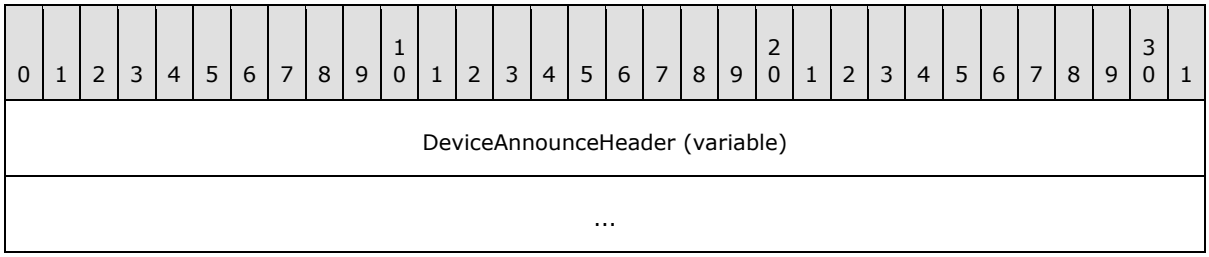
Port redirection uses common data types specified in [MS-RDPEFS] section 2.

2.2.2 Port Redirection Messages

This protocol does not define any specific messages. It uses a subset of the messages specified in [MS-RDPEFS] section 2. The messages in the following sections are used by this protocol.

2.2.2.1 Client Device List Announce Request

This message is described in [MS-RDPEFS] section 2.2.2.9. The port redirection client generates the elements of type DEVICE_ANNOUNCE (as specified in [MS-RDPEFS] section 2.2.1.3) for the port devices it wants to redirect.



DeviceAnnounceHeader (variable): For each redirected port a DEVICE_ANNOUNCE header (as specified in [MS-RDPEFS] section 2.2.1.3) is generated by the client port redirection extension.

The header fields are initialized as follows:

- **DeviceType**: Identifies the device. This value MUST be set to RDPDR_DTYP_PARALLEL for parallel ports and RDPDR_DTYP_SERIAL for serial ports.
- **DeviceId**: A unique ID that identifies the announced device. This ID can be reused only if the device is removed.

- **PreferredDosName**: The name of the port device as it appears on the client. This protocol does not enforce any semantic limitations on port naming. Client and server implementations determine the port naming convention.<1>
- **DeviceDataLength**: Number of bytes in the **DeviceData** field. For port devices, this value is set to 0.

2.2.2.2 Server Create Request (DR_PORT_CREATE_REQ)

This message is sent by the server to open an instance of the port device. The packet is specified in [MS-RDPEFS] section 2.2.1.4.1 (DR_CREATE_REQ).

Some of the parameters that are passed with this request (**DesiredAccess**, **AllocationSize**, **FileAttributes**, **SharedAccess**, **Disposition** and **CreateOptions**) are treated as opaque by this protocol. The interpretation of these parameters is determined by the client-side driver. The various possible values are specified in [MS-SMB2] section 2.2.13.

2.2.2.3 Server Close Request (DR_PORT_CLOSE_REQ)

This message is sent from the server to close the previously-opened device instance. The packet is specified in [MS-RDPEFS] section 2.2.1.4.2 (DR_CLOSE_REQ).

2.2.2.4 Server Read Request (DR_PORT_READ_REQ)

This message is sent from the server to read data from the port device instance. The packet is specified in [MS-RDPEFS] section 2.2.1.4.3 (DR_READ_REQ).

The **Offset** field in this request MUST be set to 0.

Zero-length request semantics: The protocol allows the client and server to request or to complete read/write operations with the **Length** field set to zero. The behavior of these requests and their interpretation is determined by the server application and the client driver.

2.2.2.5 Server Write Request (DR_PORT_WRITE_REQ)

This message is sent from the server to write data to the port device instance. The packet is specified in [MS-RDPEFS] section 2.2.1.4.4 (DR_WRITE_REQ).

The **Offset** field in this request MUST be set to 0.

Zero-length request semantics: The protocol allows the client and server to request or to complete read/write operations with the **Length** field set to zero. The behavior of these requests and their interpretation is determined by the server application and the client driver.

2.2.2.6 Server Device Control Request (DR_PORT_CONTROL_REQ)

This message is sent by the server to request a **device control** operation. The packet is specified in [MS-RDPEFS] section 2.2.1.4.5 (DR_CONTROL_REQ).

The possible values for the **IoControlCode** member and the corresponding Input and Output buffers applicable to parallel and serial ports are specified in [MSFT-W2KDDK] Volume 2, Chapter 13, Part II - Serial and Parallel Drivers, and in [MSDN-PORTS].

2.2.2.7 Client Create Response (DR_PORT_CREATE_RSP)

The client responds with this message to notify the server about the result of the server create request (section 2.2.2.2). This message is specified in [MS-RDPEFS] section 2.2.1.5.1 (DR_CREATE_RSP).

2.2.2.8 Client Close Response (DR_PORT_CLOSE_RSP)

The client responds with this message to notify the server about the result of the server close request (section 2.2.2.3). This message is specified in [MS-RDPEFS] section 2.2.1.5.2 (DR_CLOSE_RSP).

2.2.2.9 Client Read Response (DR_PORT_READ_RSP)

The client responds with this message to notify the server about the result of server read request (section 2.2.2.4). This message is specified in [MS-RDPEFS] section 2.2.1.5.3 (DR_READ_RSP).

2.2.2.10 Client Write Response (DR_PORT_WRITE_RSP)

The client responds with this message to notify the server about the result of the server write request (section 2.2.2.5). This message is specified in [MS-RDPEFS] section 2.2.1.5.4 (DR_WRITE_RSP).

2.2.2.11 Client Device Control Response (DR_PORT_CONTROL_RSP)

The client responds with this message to notify the server about the result of the server device control request (2.2.2.6). This message is specified in [MS-RDPEFS] section 2.2.1.5.5 (DR_CONTROL_RSP).

3 Protocol Details

The following sections specify protocol details, including abstract data models and message processing rules.

3.1 Common Details

3.1.1 Abstract Data Model

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

The Remote Desktop Protocol: Serial Port Virtual Channel Extension follows the abstract data model specified in [MS-RDPEFS] section 3.1.1.

3.1.2 Timers

There are no common timers.

3.1.3 Initialization

The Remote Desktop Protocol: File System Virtual Channel Extension must be initialized before the ports can be redirected.

3.1.4 Higher-Layer Triggered Events

IO requests are generated in response to IO calls that the server applications make on the redirected device. Otherwise, no higher-layer triggered events are used.

3.1.5 Message Processing Events and Sequencing Rules

The common message processing events and rules that are described in [MS-RDPEFS] section 3.1.5 apply to this protocol. For client- and server-specific message processing, see sections 3.2.5 and 3.3.5.

3.1.6 Timer Events

No common timer events are used.

3.1.7 Other Local Events

There are no common local events.

3.2 Client Details

3.2.1 Abstract Data Model

The abstract data model is specified in section 3.1.1.

3.2.2 Timers

No timers are used.

3.2.3 Initialization

Initialization is specified in section 3.1.3.

3.2.4 Higher-Layer Triggered Events

No client higher-layer triggered events are used.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Port Redirection Messages

3.2.5.1.1 Sending a Client Device List Announce Request Message

After Remote Desktop Protocol: File System Virtual Channel Extension initialization messages, the client sends a DR_CORE_DEVICELIST_ANNOUNCE_REQ message to the server along with information for various devices, as specified in [MS-RDPEFS] section 2.2.2.9. The port extension prepares the port devices information that goes into this packet. The port-specific structure is specified in section 2.2.2.1.

The port redirection extension enumerates the local serial and parallel ports that need to be redirected. It MUST set the appropriate **PreferredDosName** fields, generate unique IDs for the devices, set the appropriate device types, and let the Remote Desktop Protocol: File System Virtual Channel Extension send the information over to the server.

The **DeviceId** field generated in this message is used to refer to this port in subsequent messages.

3.2.5.1.2 Processing a Server Create Request Message

After receiving the create request, the client SHOULD open and prepare the port for IO operations. The opened instance of the port is maintained by the client as a **FileId** field. This ID is used to refer to subsequent IO operations on the port instance. After the create request processing is complete, the client responds with a create response message (section 3.2.5.1.7).

3.2.5.1.3 Processing a Server Close Request Message

The **DeviceId** and **FileId** fields of DR_DEVICE_IOREQUEST (as specified in [MS-RDPEFS] section 2.2.1.4) identify the previously-opened instance of the port to operate on.

After receiving the close request, the client MUST close the previously opened port instance. It MUST also cancel pending IO operations, if any, on the port.

3.2.5.1.4 Processing a Server Read Request Message

The **DeviceId** and **FileId** fields of DR_DEVICE_IOREQUEST (as specified in [MS-RDPEFS] section 2.2.1.4) identify the previously opened instance of the port to operate on.

The **Length** and **Offset** fields of DR_READ_REQ (as specified in [MS-RDPEFS] section 2.2.1.4.3) identify the length and offset values requested by the server for this operation.

The client MUST perform the read operation on the instance of the port accordingly. After the read operation is complete, the client MUST send a response message to let the server know about the result of the operation.

The semantics of read requests are determined by the client-side driver. The protocol allows partial read results. The result of the read operations, including the data read, is passed to the server and is considered opaque to the protocol.

3.2.5.1.5 Processing a Server Write Request Message

The **DeviceId** and **FileId** fields of DR_DEVICE_IOREQUEST (as specified in [MS-RDPEFS] section 2.2.1.4) identify the previously-opened instance of the port to operate on.

The **Length**, **Offset** and **WriteData** fields of DR_WRITE_REQ (as specified in [MS-RDPEFS] section 2.2.1.4.4) identify the parameters for the write operation.

The client MUST perform the write operation on the instance of the port accordingly. After the operation is complete, the client MUST send a response message to let the server know about the result of the operation.

3.2.5.1.6 Processing a Server Device Control Request Message

The **DeviceId** and **FileId** fields of DR_DEVICE_IOREQUEST (as specified in [MS-RDPEFS] section 2.2.1.4) identify the previously-opened instance of the port to operate on.

The **OutputBufferLength**, **InputBufferLength**, **IoControlCode** and **InputBuffer** fields of DR_CONTROL_REQ (as specified in [MS-RDPEFS] section 2.2.1.4.5) identify the parameters for the device control operation.

The client MUST perform the device control operation specified by the **IoControlCode** field on the instance of the port accordingly. After the operation is complete, the client MUST send a response message to let the server know about the result of the operation. <2>

3.2.5.1.7 Sending a Create Response Message

This message is sent in response to the server create request (section 3.2.5.1.2).

The client MUST fill out various fields of DR_CREATE_RSP (as specified in [MS-RDPEFS] section 2.2.1.5.1) as follows:

For the RDPDR_HEADER header:

- The **Component** field MUST be set to RDPDR_CTYP_CORE.
- The **PacketId** field MUST be set to PAKID_CORE_DEVICE_IOCOMPLETION.

For the DR_DEVICE_IOCOMPLETION header:

- The **DeviceId** field MUST be set to match the corresponding **DeviceId** field from the IO request.
- The **CompletionId** field MUST be set to match the **CompletionId** field from the corresponding IO request (section 3.2.5.1.2).
- The **IoStatus** field MUST be set to the NTSTATUS indicating the result of the operation.

After completing the create request, the client MUST set the **FileId** field to a unique **FileId** value to identify the instance of the port. This **FileId** field is used in subsequent IO operations to refer to the port instance.

The **Information** field MUST be set to 0.

3.2.5.1.8 Sending a Close Response Message

This message is sent in response to the server close request (3.2.5.1.3).

The client MUST fill out the various members of DR_CLOSE_RSP (as specified in [MS-RDPEFS] section 2.2.1.5.2) as follows:

For the RDPDR_HEADER:

- The **Component** field MUST be set to RDPDR_CTYP_CORE.
- The **PacketId** field MUST be set to PAKID_CORE_DEVICE_IOCOMPLETION.

For the DR_DEVICE_IOCOMPLETION:

- The **DeviceId** field MUST be set to match the corresponding **DeviceId** field from the IO request.
- The **CompletionId** field MUST be set to match the **CompletionId** field from the corresponding IO request (section 3.2.5.1.3).
- The **IoStatus** field MUST be set to the NTSTATUS value indicating the result of the operation.

3.2.5.1.9 Sending a Read Response Message

This message is sent in response to the server read request (3.2.5.1.4).

The client MUST fill out the various members of DR_READ_RSP (as specified in [MS-RDPEFS] section 2.2.1.5.3) as follows:

For the RDPDR_HEADER:

- The **Component** field MUST be set to RDPDR_CTYP_CORE.
- The **PacketId** field MUST be set to PAKID_CORE_DEVICE_IOCOMPLETION.

For the DR_DEVICE_IOCOMPLETION:

- The **DeviceId** field MUST be set to match the corresponding **DeviceId** field from the IO request.
- The **CompletionId** field MUST be set to match the **CompletionId** from the corresponding IO request (section 3.2.5.1.4).
- The **IoStatus** field MUST be set to the NTSTATUS value indicating the result of the operation.

The client prepares a reply message with the result of the read operation. The client populates the **Length** field with the number of bytes read. The actual data read follows the **Length** field. The **Length** field MAY be less than the requested length; however, the **Length** field MUST NOT be greater than requested length. These partial read requests are supported by the server.

3.2.5.1.10 Sending a Write Response Message

This message is sent in response to the server write request (section 3.2.5.1.5).

The client MUST fill out the various members of DR_WRITE_RSP (as specified in [MS-RDPEFS] section 2.2.1.5.4) as follows:

For the RDPDR_HEADER:

- The **Component** field MUST be set to RDPDR_CTYP_CORE.
- The **PacketId** field MUST be set to PAKID_CORE_DEVICE_IOCOMPLETION.

For the DR_DEVICE_IOCOMPLETION:

- The **DeviceId** field MUST be set to match the corresponding **DeviceId** field from the IO request.
- The **CompletionId** field MUST be set to match the **CompletionId** field from the corresponding IO request (section 3.2.5.1.5).
- The **IoStatus** field MUST be set to the NTSTATUS indicating the result of the operation.
- The **Padding** field is optional. This field can be included in this response by the client. This field is unused and MUST be ignored by the server.

The client prepares a reply message with the result of the write operation. The client populates the **Length** field with the number of bytes written.

3.2.5.1.11 Sending a Device Control Response Message

This message is sent in response to the server device control request (section 3.2.5.1.6).

The client MUST fill out the various members of DR_CONTROL_RSP (as specified in [MS-RDPEFS] section 2.2.1.5.5) as follows:

For the RDPDR_HEADER:

- The **Component** field MUST be set to RDPDR_CTYP_CORE.
- The **PacketId** field MUST be set to PAKID_CORE_DEVICE_IOCOMPLETION.

For the DR_DEVICE_IOCOMPLETION:

- The **DeviceId** field MUST be set to match the corresponding **DeviceId** field from the IO request.
- The **CompletionId** field MUST be set to match the **CompletionId** field from the corresponding IO request (section 3.2.5.1.6).
- The **IoStatus** field MUST be set to the NTSTATUS indicating the result of the operation.

The client prepares a reply message with the result of the device control operation. The client populates the **OutputBufferLength** field with the number of bytes returned by the device control operation. The actual data returned, if any, follows the packet.

3.2.6 Timer Events

There are no timer events.

3.2.7 Other Local Events

There are no other local events.

3.3 Server Details

3.3.1 Abstract Data Model

The abstract data model is specified in section 3.1.1.

3.3.2 Timers

There are no timers used.

3.3.3 Initialization

Initialization is specified in section 3.1.3.

3.3.4 Higher-Layer Triggered Events

All IO requests are originated by the applications making IO calls on the redirected device. Otherwise, no other higher-layer triggered events are used.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Port Redirection Messages

3.3.5.1.1 Processing a Client Device List Announce Request Message

After receiving the port announce message, the server SHOULD create a pseudo port device that emulates the client device. The server should maintain the association of the **DeviceId** value obtained from the client with such a pseudo device.

3.3.5.1.2 Sending a Server Create Request Message

The server sends this message when any server application opens the pseudo port device. The server passes all the create parameters obtained from the application request over to the client for actual processing.

3.3.5.1.3 Sending a Server Close Request Message

The server sends this message to the client in response to the server application requesting the close operation on the pseudo port device for actual processing.

3.3.5.1.4 Sending a Server Write Request Message

The server sends this message to the client in response to the server application requesting the write operation on the pseudo port device for actual processing.

3.3.5.1.5 Sending a Server Read Request Message

The server sends this message to the client in response to the server application requesting the read operation on the pseudo port device for actual processing.

3.3.5.1.6 Sending a Server Device Control Request Message

The server sends this message to the client in response to the server application requesting the device control operation on the pseudo port device for actual processing.

3.3.5.1.7 Processing a Client Create Response Message

After receiving the create response, the server responds to the application that initiated the create operation (section 3.3.5.1.1). The server MUST maintain an association between the **FileId** returned by the client and the file handle returned to the application. For any subsequent IO operations on the file handle, the server SHOULD send the IO to the client for completion using the same **FileId** field.

3.3.5.1.8 Processing a Client Close Response Message

The server responds to the application with the result of the close response received from the client.

3.3.5.1.9 Processing a Client Write Response Message

The server SHOULD forward the result of the write response to the application that requested the write operation.

3.3.5.1.10 Processing a Client Read Response Message

The server SHOULD forward the result of the read response to the application that initiated the read operation.

3.3.5.1.11 Processing a Client Device Control Response Message

The server SHOULD forward the result of the device control response to the application that initiated the operation.

3.3.6 Timer Events

There are no timer events.

3.3.7 Other Local Events

There are no other local events.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Remote Desktop Protocol: Serial Port Virtual Channel Extension.

4.1 Port Redirection Annotations

After the Remote Desktop Protocol: Serial Port Virtual Channel Extension finishes its initialization, the client sends the client device list announce message to the server. Information about the ports to be redirected is contained within the message.

The following sequence shows a COM2 port being sent to the server for redirection.

```
Channel Name = RDPDR, 28, Client to server
00000000 72 44 41 44 01 00 00 00 01 00 00 00 01 00 00 00
00000010 43 4f 4d 32 00 00 00 00 00 00 00 00

72 44                   -> RDPDR_CTYP_CORE                = 0x4472
41 44                   -> PAKID_CORE_DEVICELIST_ANNOUNCE = 0x4441
01 00 00 00             -> DeviceCount                    = 0x00000001
01 00 00 00             -> DeviceType (RDPDR DTYP SERIAL) = 0x00000001
01 00 00 00             -> DeviceId                       = 0x00000001
43 4f 4d 32 00 00 00 00 -> PreferredDosName(8 characters) = "COM2"
00 00 00 00             -> DeviceDataLength               = 0x00000000
```

The following sequence shows an LPT1 port being sent for redirection. In this example, the Device Announce packet contains three devices. This example only annotates the header and LPT1 Device portion from this packet.

```
Channel Name = RDPDR, 264, Client to server
00000000 72 44 41 44 03 00 00 00 04 00 00 00 04 00 00 00
00000010 50 52 4e 34 00 00 00 00 50 00 00 00 10 00 00 00
00000020 00 00 00 00 00 00 00 00 1c 00 00 00 1c 00 00 00
00000030 00 00 00 00 41 00 70 00 6f 00 6c 00 6c 00 6f 00
00000040 20 00 50 00 2d 00 31 00 32 00 30 00 30 00 00 00
00000050 41 00 70 00 6f 00 6c 00 6c 00 6f 00 20 00 50 00
00000060 2d 00 31 00 32 00 30 00 30 00 00 00 04 00 00 00
00000070 03 00 00 00 50 52 4e 33 00 00 00 00 74 00 00 00
00000080 12 00 00 00 00 00 00 00 00 00 00 00 2e 00 00 00
00000090 2e 00 00 00 00 00 00 00 43 00 61 00 6e 00 6f 00
000000a0 6e 00 20 00 42 00 75 00 62 00 62 00 6c 00 65 00
000000b0 2d 00 4a 00 65 00 74 00 20 00 42 00 4a 00 2d 00
000000c0 33 00 30 00 00 00 43 00 61 00 6e 00 6f 00 6e 00
000000d0 20 00 42 00 75 00 62 00 62 00 6c 00 65 00 2d 00
000000e0 4a 00 65 00 74 00 20 00 42 00 4a 00 2d 00 33 00
000000f0 30 00 00 00 02 00 00 00 02 00 00 00 4c 50 54 31
00000100 00 00 00 00 00 00 00 00

72 44                 -> RDPDR_CTYP_CORE               = 0x4472
41 44                 -> PAKID CORE DEVICELIST ANNOUNCE = 0x4441
03 00 00 00           -> DeviceCount                   = 0x00000003

...

02 00 00 00             -> DeviceType (RDPDR DTYP PARALLEL) = 0x00000002
02 00 00 00             -> DeviceId                         = 0x00000002
4c 50 54 31 00 00 00 00 -> PreferredDosName (8 characters)  = "LPT1"
```

```
00 00 00 00             -> DeviceDataLength                = 0x00000000
```

4.2 Server Create Request Example

The server sends a request to create an instance of the port. The following sequence captures such a request.

```
RDPDR, 56, Server to client
00000000 72 44 52 49 01 00 00 00 00 00 00 00 01 00 00 00
00000010 00 00 00 00 00 00 00 00 80 00 10 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 07 00 00 00 01 00 00 00
00000030 60 00 00 00 00 00 00 00

72 44                   -> RDPDR_CTYP_CORE              = 0x4472
52 49                   -> PAKID_CORE_DEVICE_IOREQUEST  = 0x4952
01 00 00 00             -> DeviceId                     = 0x00000001
00 00 00 00             -> FileId                       = 0x00000000
01 00 00 00             -> CompletionId                 = 0x00000001
00 00 00 00             -> MajorFunction(IRP MJ CREATE) = 0x00000000
00 00 00 00             -> MinorFunction                = 0x00000000
80 00 10 00             -> DesiredAccess                = 0x00100080
00 00 00 00 00 00 00 00 -> AllocationSize(64 bits)      = 0x0
00 00 00 00             -> FileAttributes               = 0x00000000
07 00 00 00             -> SharedAccess                 = 0x00000007
01 00 00 00             -> Disposition                  = 0x00000001
60 00 00 00             -> CreateOptions                = 0x00000060
00 00 00 00             -> PathLength                   = 0x0000000
```

4.3 Client Create Response Example

The client responds with the following response. This establishes a **FileId** value that is used subsequently for the IO operations.

```
RDPDR, 21, Client to server
00000000 72 44 43 49 02 00 00 00 00 00 00 00 00 00 00 00
00000010 01 00 00 00 00

72 44       -> RDPDR CTYP CORE              = 0x4472
43 49       -> PAKID_CORE_DEVICE_IOCOMPLETION = 0x4943
02 00 00 00 -> DeviceId                       = 0x00000001
00 00 00 00 -> CompletionId                   = 0x00000001
00 00 00 00 -> NTSTATUS                       = 0x00000000
01 00 00 00 -> FileId                         = 0x00000001
00          -> Information                    = 0x00
```

4.4 IO Operations Examples

The server on behalf of the application sends IO operations on the **FieldId**. The following message sequences illustrate the packets for read, write, and device control operations.

Server read request (DR_PORT_READ_REQ)

```
RDPDR, 56, Server to client
```

```
00000000 72 44 52 49 02 00 00 00 02 00 00 00 00 00 00 00
00000010 03 00 00 00 00 00 00 00 08 02 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00


72 44         -> RDPDR CTYP CORE             = 0x4472
52 49         -> PAKID_CORE_DEVICE_IOREQUEST = 0x4952
02 00 00 00   -> DeviceId                    = 0x00000002
02 00 00 00   -> FileId                      = 0x00000002
00 00 00 00   -> CompletionId                = 0x00000000
03 00 00 00   -> MajorFunction (IRP MJ READ) = 0x00000003
00 00 00 00   -> MinorFunction               = 0x00000000
08 02 00 00   -> Length                      = 0x00000208
00 00 00 00 00 00 00 00
              -> Offset
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              -> Padding (20 bytes)
```

Server write request (DR_PORT_WRITE_REQ)

```
RDPDR, 576, Server to client
00000000 72 44 52 49 01 00 00 00 02 00 00 00 00 00 00 00
00000010 04 00 00 00 00 00 00 00 08 02 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 29 00 00 00 c4 00 7a 00  ...


72 44       -> RDPDR CTYP CORE              = 0x4472
52 49       -> PAKID CORE DEVICE IOREQUEST  = 0x4952
01 00 00 00 -> DeviceId                     = 0x00000001
02 00 00 00 -> FileId                       = 0x00000002
00 00 00 00 -> CompletionId                 = 0x00000000
04 00 00 00 -> MajorFunction (IRP_MJ_WRITE) = 0x00000004
00 00 00 00 -> MinorFunction                = 0x0000000008
08 02 00 00 -> Length                       = 0x00000208
00 00 00 00 00 00 00 00
            -> Offset
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            -> Padding (20 bytes)
00 00 ...   -> WriteData (520 bytes)
```

Server device control request (DR_PORT_CONTROL_REQ)

```
RDPDR, 56, Server to client
00000000 72 44 52 49 02 00 00 00 02 00 00 00 00 00 00 00
00000010 0E 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
00000020 50 00 1b 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00

72 44       -> RDPDR_CTYP_CORE                       = 0x4472
52 49       -> PAKID CORE DEVICE IOREQUEST           = 0x4952
02 00 00 00 -> DeviceId                              = 0x00000002
02 00 00 00 -> FileId                                = 0x00000002
00 00 00 00 -> CompletionId                          = 0x00000000
0E 00 00 00 -> MajorFunction (IRP_MJ_DEVICE_CONTROL) = 0x0000000E
00 00 00 00 -> MinorFunction                         = 0x00000000
```

```
04 00 00 00 -> OutputBufferLength                    = 0x00000004
00 00 00 00 -> InputBufferLength                     = 0x00000000
50 00 1b 00 -> IoControlCode                         = 0x001B0050
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            -> Padding (20 bytes)
```

The following packets show the typical responses from the client for these requests.

Client read response (DR_PORT_READ_RSP)

```
RDPDR, 540, Client to server
00000000 72 44 43 49 02 00 00 00 00 00 00 00 00 00 00 00
00000010 08 02 00 00 29 00 00 00 c4 00 7a 00 4c 4f 43 41 ...

72 44        -> RDPDR_CTYP_CORE                = 0x4472
43 49        -> PAKID_CORE_DEVICE_IOCOMPLETION = 0x4943
02 00 00 00  -> DeviceId                       = 0x00000001
00 00 00 00  -> CompletionId                   = 0x00000001
00 00 00 00  -> NTSTATUS                       = 0x00000000
08 02 00 00  -> Length                         = 0x00000208
29 00 ...    -> ReadData (520 bytes)
```

Client write response (DR_PORT_WRITE_RSP)

```
RDPDR,21,send
00000000 72 44 43 49 01 00 00 00 00 00 00 00 00 00 00 00
00000010 08 02 00 00 00

72 44        -> RDPDR CTYP CORE                = 0x4472
43 49        -> PAKID CORE DEVICE IOCOMPLETION = 0x4943
01 00 00 00  -> DeviceId                       = 0x00000001
00 00 00 00  -> CompletionId                   = 0x00000001
00 00 00 00  -> NTSTATUS                       = 0x00000000
08 02 00 00  -> Length
00           -> Padding
```

Client device control response (DR_PORT_CONTROL_RSP)

```
RDPDR,24,send
00000000 72 44 43 49 02 00 00 00 00 00 00 00 00 00 00 00
00000010 04 00 00 00 80 25 00 00

72 44        -> RDPDR CTYP CORE                = 0x4472
43 49        -> PAKID CORE DEVICE IOCOMPLETION = 0x4943
02 00 00 00  -> DeviceId                       = 0x00000002
00 00 00 00  -> CompletionId                   = 0x00000000
00 00 00 00  -> IoStatus                       = 0x00000000
04 00 00 00  -> OutputBufferLength             = 0x00000004
80 25 00 00  -> OutputBuffer                   = 0x00002580
```

Finally, the server calls a close request to close the port instance.

Server close request (DR_PORT_CLOSE_REQ)

```
RDPDR, 56, Server to client
00000000 72 44 52 49 01 00 00 00 01 00 00 00 01 00 00 00
00000010 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00

72 44        -> RDPDR_CTYP_CORE              = 0x4472
52 49        -> PAKID CORE DEVICE IOREQUEST  = 0x4952
01 00 00 00  -> DeviceId                     = 0x00000001
01 00 00 00  -> FileId                       = 0x00000001
01 00 00 00  -> CompletionId                 = 0x00000001
02 00 00 00  -> MajorFunction (IRP_MJ_CLOSE) = 0x00000002
00 00 00 00  -> MinorFunction                = 0x00000000
00 00 ...    -> Padding (32 bytes)
```

After closing the local port instance, the client responds with this message.

Client close response (DR_PORT_CLOSE_RSP)

```
RDPDR, 20, Client to server
00000000 72 44 43 49 02 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00

72 44        -> RDPDR CTYP CORE                 = 0x4472
43 49        -> PAKID_CORE_DEVICE_IOCOMPLETION  = 0x4943
02 00 00 00  -> DeviceId                        = 0x00000001
00 00 00 00  -> CompletionId                    = 0x00000001
00 00 00 00  -> NTSTATUS                        = 0x00000000
00 00 00 00  -> Padding
```

5 Security

The following sections specify security considerations for implementers of the Remote Desktop Protocol: Serial Port Virtual Channel Extension.

5.1 Security Considerations for Implementers

There are no security considerations for Remote Desktop Protocol: Serial Port Virtual Channel Extension messages because all static virtual channel traffic is secured by the underlying Remote Desktop Protocol core protocol. The implemented security-related mechanisms are specified in [MS-RDPBCGR] section 5.

5.2 Index of Security Parameters

There are no security parameters in Remote Desktop Protocol: Serial Port Virtual Channel Extension.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows 2000
- Windows XP
- Windows Server 2003
- 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 2.2.2.1: The server implementation should use symbolic link names that use standard naming conventions, in order to ensure compatibility with most Windows applications and drivers that access ports.

For COM ports, a symbolic link name should be assigned that uses the standard naming convention "COM<n>", where <n> is the COM port number.

For LPT ports, a symbolic link name should be assigned that uses the standard naming convention "LPT<n>", where <n> is the LPT port number.

For more information on port naming conventions, see [MSDN-PORTS].

<2> Section 3.2.5.1.6: Windows Implementations use IOCTL constants for **IoControlCode** values. The content and values of the IOCTLs are opaque to the protocol. On the server side, the data contained in an IOCTL is simply packaged and sent to the client side. For maximum compatibility between the different versions of the Windows operating system, the client implementation only singles out critical IOCTLs and invokes the applicable Win32 port API. The other IOCTLS are passed directly to the client-side driver, and the processing of this value depends on the drivers installed on the client side. The values and parameters for these IOCTLS can be found in [MSFT-W2KDDK] Volume 2, Chapter 13, Part II - Serial and Parallel Drivers, and in [MSDN-PORTS].

- IOCTL_SERIAL_SET_BAUD_RATE
- IOCTL_SERIAL_GET_BAUD_RATE
- IOCTL_SERIAL_SET_LINE_CONTROL
- IOCTL_SERIAL_GET_LINE_CONTROL
- IOCTL_SERIAL_SET_TIMEOUTS
- IOCTL_SERIAL_GET_TIMEOUTS
- IOCTL_SERIAL_SET_CHARS
- IOCTL_SERIAL_GET_CHARS
- IOCTL_SERIAL_SET_DTR

- IOCTL_SERIAL_CLR_DTR
- IOCTL_SERIAL_RESET_DEVICE
- IOCTL_SERIAL_SET_RTS
- IOCTL_SERIAL_CLR_RTS
- IOCTL_SERIAL_SET_XOFF
- IOCTL_SERIAL_SET_XON
- IOCTL_SERIAL_SET_BREAK_ON
- IOCTL_SERIAL_SET_BREAK_OFF
- IOCTL_SERIAL_SET_QUEUE_SIZE
- IOCTL_SERIAL_GET_WAIT_MASK
- IOCTL_SERIAL_SET_WAIT_MASK
- IOCTL_SERIAL_WAIT_ON_MASK
- IOCTL_SERIAL_IMMEDIATE_CHAR
- IOCTL_SERIAL_PURGE
- IOCTL_SERIAL_GET_HANDFLOW
- IOCTL_SERIAL_SET_HANDFLOW
- IOCTL_SERIAL_GET_MODEMSTATUS
- IOCTL_SERIAL_GET_DTRRTS
- IOCTL_SERIAL_GET_COMMSTATUS
- IOCTL_SERIAL_GET_PROPERTIES
- IOCTL_SERIAL_XOFF_COUNTER
- IOCTL_SERIAL_LSRMST_INSERT
- IOCTL_SERIAL_CONFIG_SIZE
- IOCTL_SERIAL_GET_COMMCONFIG
- IOCTL_SERIAL_GET_STATS
- IOCTL_SERIAL_CLEAR_STATS

7 Index

P

R

S

T

V

W