[MS-RDPEPNP]:
Remote Desktop Protocol:
Plug and Play Devices Virtual Channel Extension

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	1.1	Minor	Minor technical content changes.
07/20/2007	1.1.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
08/10/2007	1.2	Minor	Updated content based on feedback.
09/28/2007	1.3	Minor	Made technical and editorial changes based on feedback.
10/23/2007	1.4	Minor	Made technical and editorial changes based on feedback.
11/30/2007	1.5	Minor	Made technical and editorial changes based on feedback.
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension to the Remote Desktop Protocol.<1> This protocol is used to redirect Plug and Play (PNP) devices from a **terminal client** to the **terminal server**. This allows the server access to devices physically connected to the client as if the device were local to the server.

1.1 Glossary

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

Device Driver
Globally Unique Identifier (GUID)
Handle
HRESULT
Unicode

The following terms are specific to this document:

Device Interface: A uniform and extensible mechanism that interacts programmatically with applications and the system. A **device driver** can expose zero, one, or more than one **device interfaces** for a particular device. A **device interface** is represented by a **GUID**.

Input/Output (I/O) Routines: A routine defined by an operating system for applications to interact with a **device driver**. Applications use these routines for tasks, such as opening a device, creating a file, reading data from a device, writing data to a device, or sending control codes to a device.

Multisz String: A null-terminated string composed of other null-terminated strings appended together. For example, a **multisz string** that contains "one", "brown", and "cow" would be represented as three null-terminated strings—"one\0", "brown\0", "cow\0"—appended together with an additional null appended, as follows: "one\0brown\0cow\0\0".

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 for the server to process.

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-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-RDPEDYC] Microsoft Corporation, "Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension", June 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

1.2.2 Informative References

None.

1.3 Protocol Overview (Synopsis)

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension specifies the communication used to enable the redirection of devices between a terminal client and a terminal server. The restrictions placed on devices that may be redirected using this protocol are specified in section 1.6. By redirecting devices from the terminal client to the terminal server, applications running on a server machine can access the **remote devices** as if they were local devices. For example, a user can attach an MP3 player device to the terminal client, and then synchronize music using a media player application running on the terminal server.

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension consists of two sub-protocols.

- Plug and Play (PNP) Device Info
- Plug and Play (PNP) Device I/O

1.3.1 PNP Device Info Subprotocol

The PNP Device Info Subprotocol specifies the communication between the terminal server client and the terminal server component that handles the creation and removal of remote devices on the server side. This subprotocol is used to create remote device instances on the server machine that correspond to the physical devices on the client machine. The following illustration shows the PNP Device Info Subprotocol message sequence. This subprotocol uses a dynamic virtual channel named PNPDR for communication between client and server.

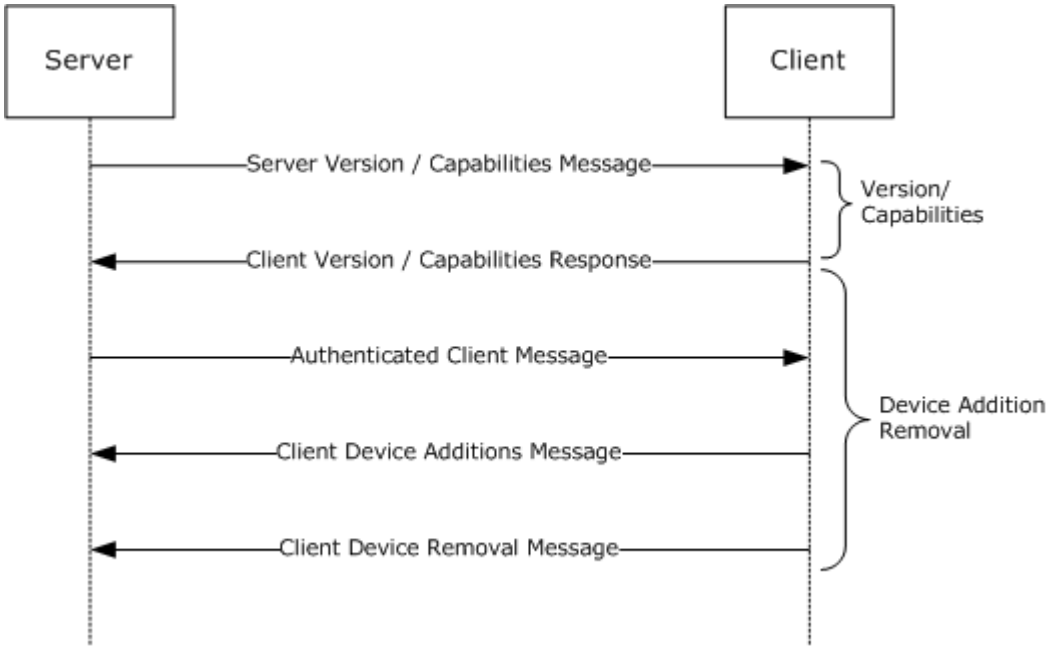


Figure 1: PNP Device Info message sequence

This subprotocol consists of a versioning and capabilities negotiation phase, in addition to a device addition and removal phase. The terminal client sends the device information to the terminal server, and the terminal server creates the remote device instances that represent the physical devices.

1.3.2 PNP Device I/O Subprotocol

The PNP Device I/O Subprotocol specifies the communication between the terminal client and the remote devices on the terminal server, for handling I/O requests. This subprotocol is used to redirect the I/O calls from applications on the terminal server side to a **device driver** on the terminal client side. The following illustration shows a typical PNP Device I/O Subprotocol message sequence. This subprotocol uses a dynamic virtual channel named **FileRedirectorChannel** for communication between client and server.

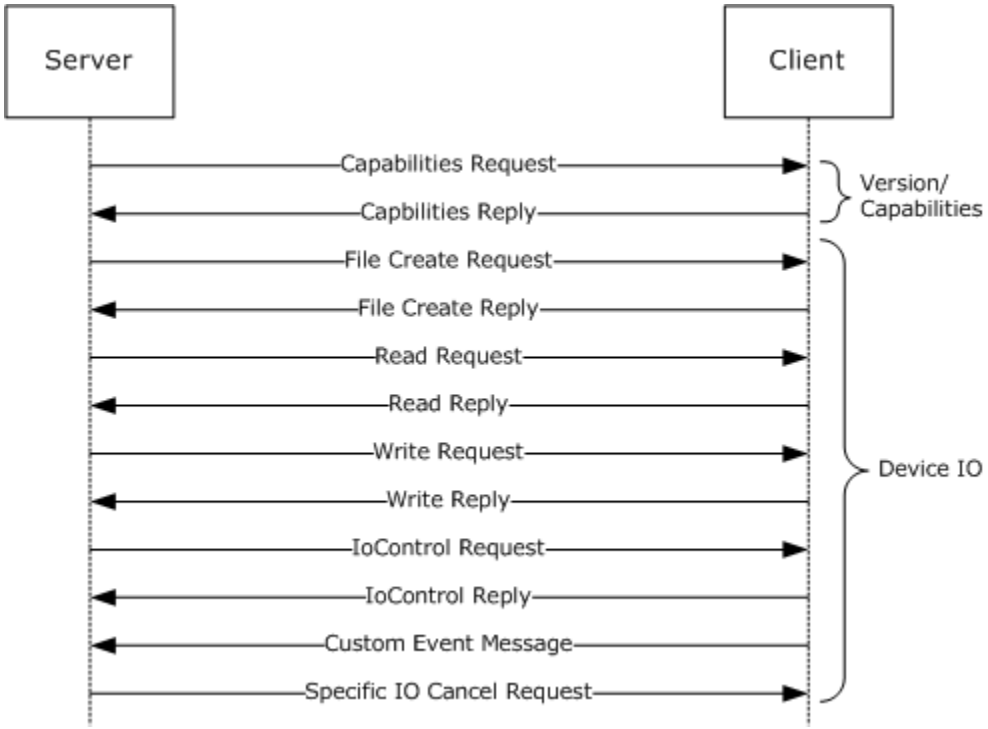


Figure 2: PNP Device I/O message sequence

For devices redirected using the PNP Device Info Subprotocol, I/O redirection takes place using the PNP Device I/O Subprotocol. The server creates a new subchannel within the **FileRedirectorChannel** main channel for each CreateFile Request. Subsequent I/O operations related to the file created are passed on this subchannel. The server sends the I/O requests to the client on behalf of applications running on the server. The client completes the I/O requests and passes the results back to the server.

1.4 Relationship to Other Protocols

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension is embedded in a dynamic virtual channel transport, as specified in [MS-RDPEDYC].

1.5 Prerequisites and Preconditions

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension operates only after the dynamic virtual channel transport is fully established. If the dynamic virtual channel transport is terminated, the Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension is also terminated.

1.6 Applicability Statement

The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension is designed to run within the context of an RDP virtual channel established between a client and server. This protocol is applicable when local client Plug and Play devices need to be accessible (redirected) in the remote session hosted on the server.

The following are requirements that device drivers and applications must meet if they need to be redirected.

- This protocol is not intended for use with devices that require quality-of-service guarantees.
- For redirection to operate properly using this protocol, all communication between devices and applications must be routed through the **I/O routines** supported by device drivers. Communication should not be routed by any other means, such as shared memory, the registry, or disk files.
- This protocol redirects operating system-specific I/O calls such as Read, Write, IOControl, and CreateFile. Communication between the custom device driver and the application cannot use anything other than these basic routines. If it does, the device cannot be redirected using this protocol.

1.7 Versioning and Capability Negotiation

This protocol defines specific messages for versioning and capability negotiations. The following messages are used for such negotiations.

- Server Version Message (section 2.2.1.2.1)
- Client Version Message (section 2.2.1.2.2)
- Server Capabilities Request Message (section 2.2.2.2.1)
- Client Capabilities Reply Message (section 2.2.2.2.2)

1.8 Vendor-Extensible Fields

This protocol uses **HRESULTs**, as specified in [MS-ERREF] section 2.1. Vendors are free to choose their own values, as long as the C bit (0x20000000) is set, indicating that it is a customer code.

1.9 Standards Assignments

This protocol contains no standards assignments.

2 Messages

2.1 Transport

This protocol is designed to operate over dynamic virtual channels, as specified in [MS-RDPEDYC], using the names PNPDR and **FileRedirectorChannel**.

One active instance of the PNPDR channel serves as a common control channel for adding and deleting devices. Multiple dynamic connections are established on the **FileRedirectorChannel** channel—one connection for each create-file request (which establishes a file **handle**) and all corresponding I/O operations made using the file handle.

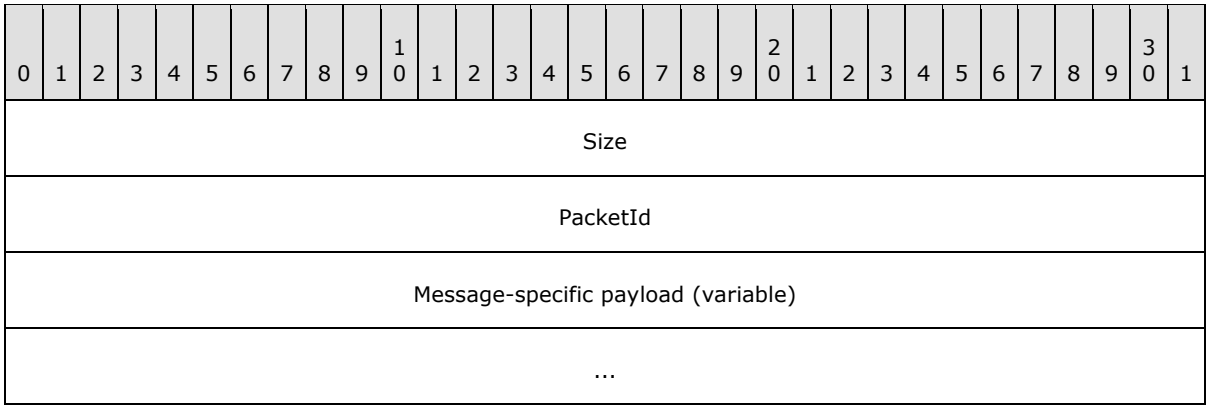
2.2 Message Syntax

2.2.1 PNP Device Info Subprotocol

The messages in the following sections specify the common header and specific messages that make up the PNP Device Info Subprotocol.

2.2.1.1 Shared Message Header (PNP_INFO_HEADER)

All messages in the PNP Device Info Subprotocol have a common header, which is followed by a message-specific payload, as described in the following sections.



Size (4 bytes): A 32-bit unsigned integer that indicates the size of the packet, including the payload.

PacketId (4 bytes): A 32-bit unsigned integer representing a unique packet ID that identifies the message. The **PacketId** field MUST be one of the following values.

Value	Meaning
IRPDR_ID_VERSION 0x00000065	Client or Server Version Message
IRPDR_ID_REDIRECT_DEVICES 0x00000066	Client Device Addition Message
IRPDR_ID_SERVER_LOGON 0x00000067	Authenticated Client Message

Value	Meaning
IRPDR_ID_UNREDIRECT_DEVICE 0x00000068	Client Device Removal Message

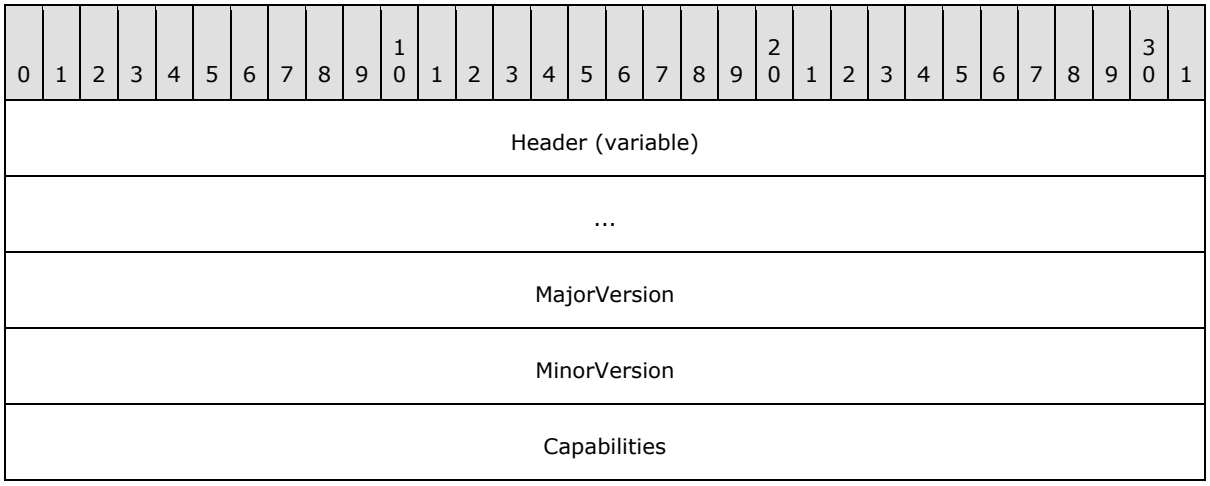
Message-specific payload (variable): The message format is identified by the **PacketId** value, as specified in the following sections.

2.2.1.2 PNP Device Info Initialization Messages

The messages in the following sections are used to initialize the PNP Device Info Subprotocol.

2.2.1.2.1 Server Version Message

The server sends this message to the client to indicate the server protocol version and server capabilities.



Header (variable): The common message header (see section 2.2.1.1). The **PacketId** field MUST be set to IRPDR_ID_VERSION (0x00000065).

MajorVersion (4 bytes): A 32-bit unsigned integer. This field MUST indicate the server major version.<2>

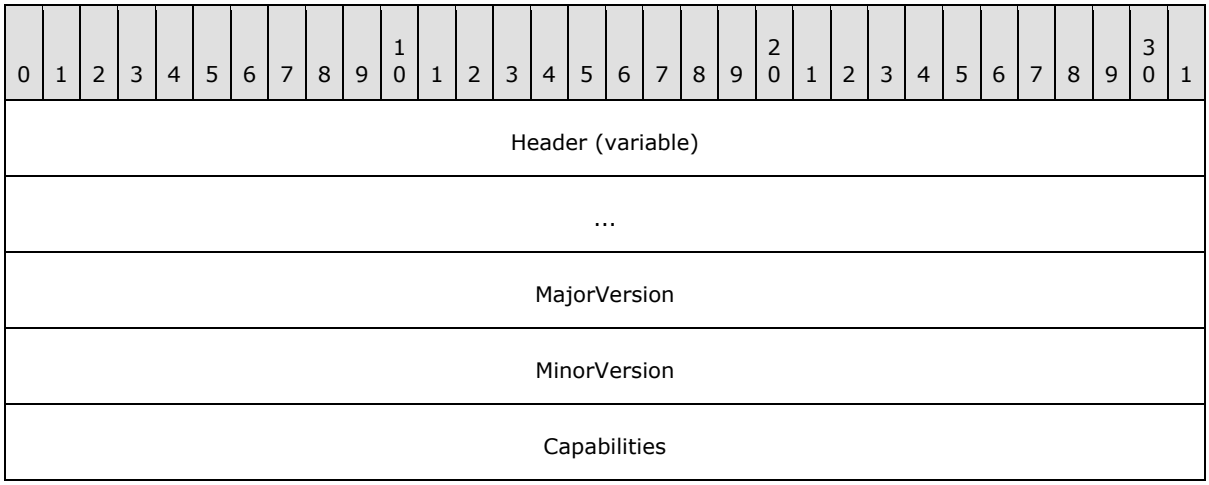
MinorVersion (4 bytes): A 32-bit unsigned integer. This field MUST indicate the server minor version.<3>

Capabilities (4 bytes): A 32-bit unsigned integer that represents a set of bit flags indicating server protocol capabilities. A bit is true (or set) if its value is equal to 1. This field MUST be composed of the bitwise OR of one or more of the following values.

Value	Meaning
0x00000001	The server supports dynamic addition of devices.

2.2.1.2.2 Client Version Message

The client sends this message to the server to indicate the client protocol version and supported capabilities in response to a Server Version Message (section 2.2.1.2.1).



Header (variable): The common message header (see section 2.2.1.1). The **PacketId** field MUST be set to IRPDR_ID_VERSION (0x00000065).

MajorVersion (4 bytes): A 32-bit unsigned integer. This field MUST indicate the client major version.<4>

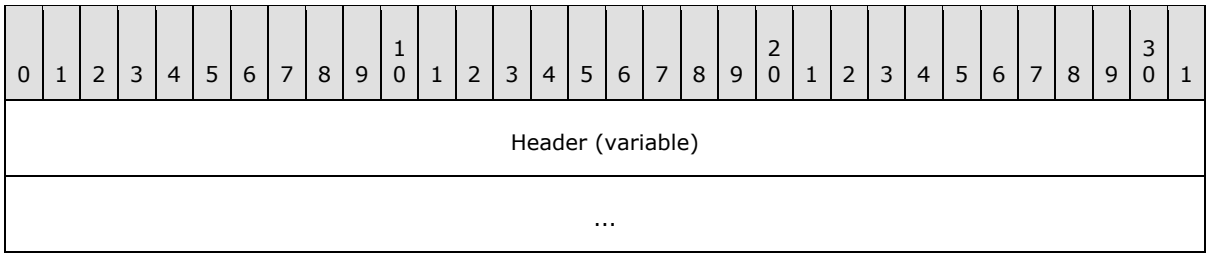
MinorVersion (4 bytes): A 32-bit unsigned integer. This field MUST indicate the client minor version.<5>

Capabilities (4 bytes): A 32-bit unsigned integer. This represents a set of bit flags that indicate client protocol capabilities. A bit is true (or set) if its value is equal to 1. This field MUST be composed of the bitwise OR of one or more of the following values.

Value	Meaning
0x00000001	The client supports dynamic addition of devices.

2.2.1.2.3 Authenticated Client Message

The server notifies the client that the user has been authenticated by sending this message. This informs the client that the server is now ready to accept any device addition or removal of PNP messages.



Header (variable): The common message header (see section 2.2.1.1). The **PacketId** field MUST be set to IRPDR_ID_SERVER_LOGON (0x00000067).

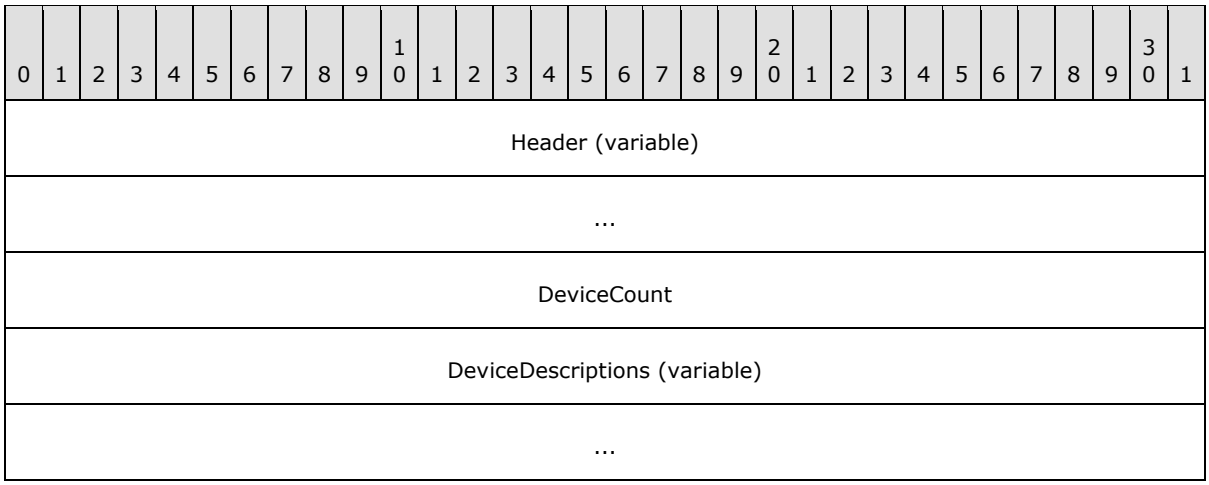
This message MUST not contain any payload.

2.2.1.3 PNP Device Info Subprotocol Device Addition and Removal Messages

The messages in the following sections are used to start and stop device redirection.

2.2.1.3.1 Client Device Addition Message

A client sends this message to redirect one or more devices. This message MUST be sent only after an Authenticated Client message is received from the server.



Header (variable): The common message header (see section 2.2.1.1). The **PacketId** field MUST be set to IRPDR_ID_REDIRECT_DEVICES (0x00000066).

DeviceCount (4 bytes): A 32-bit unsigned integer. This field indicates the number of devices contained in the following **DeviceDescriptions** field.

DeviceDescriptions (variable): An array of PNP_DEVICE_DESCRIPTION (section 2.2.1.3.1.1) structures. The number of instances of PNP_DEVICE_DESCRIPTION is specified by the **DeviceCount** field.

2.2.1.3.1.1 PNP_DEVICE_DESCRIPTION

A client device description structure. This structure contains the required information to redirect a particular device.

ClientDeviceId (4 bytes): A 32-bit unsigned integer. This value MUST be a unique ID generated by the client for the device. The server and client use this ID to refer to the device in subsequent messages.

DataSize (4 bytes): A 32-bit unsigned integer. This value MUST indicate the size of the PNP_DEVICE_DESCRIPTION structure.

cbInterfaceLength (4 bytes): A 32-bit unsigned integer. This value MUST contain the length of the **InterfaceGUIDArray** field, in bytes. This field MAY be 0x00000000.

InterfaceGUIDArray (variable): An array of **GUID** values, each representing a **device interface** exposed by the client-side device. If the value in the **cbInterfaceLength** field is 0x00000000, the **InterfaceGUIDArray** buffer MUST not be present.

cbHardwareIdLength (4 bytes): A 32-bit unsigned integer. This field MUST specify the length of the **HardwareId** field of the client-side device. This field MAY be 0x00000000.

HardwareId (variable): An array of bytes. A variable-length field that specifies a **multisz string** representing the hardware ID of the client-side device. If the value in the **cbHardwareIdLength** field is 0x00000000, the **HardwareId** buffer MUST not be present.

cbCompatIdLength (4 bytes): A 32-bit unsigned integer that specifies the length of the **CompatibilityID** field, in bytes. This field MAY be 0x00000000.

CompatibilityID (variable): An array of bytes. A variable-length field that specifies a multisz string representing the compatibility ID of the client-side device. If the value in the **cbCompatIdLength** field is 0x00000000, the **CompatibilityID** buffer MUST not be present.

cbDeviceDescriptionLength (4 bytes): A 32-bit unsigned integer that specifies the length of the **DeviceDescription** field, in bytes. This field MAY be 0x00000000.

DeviceDescription (variable): An array of bytes. A variable-length field that contains a **Unicode** string representing the device description of the client-side device. The string is not null-terminated. If the value contained in the **cbDeviceDescriptionLength** field is 0x00000000, the **DeviceDescription** buffer MUST not be present.

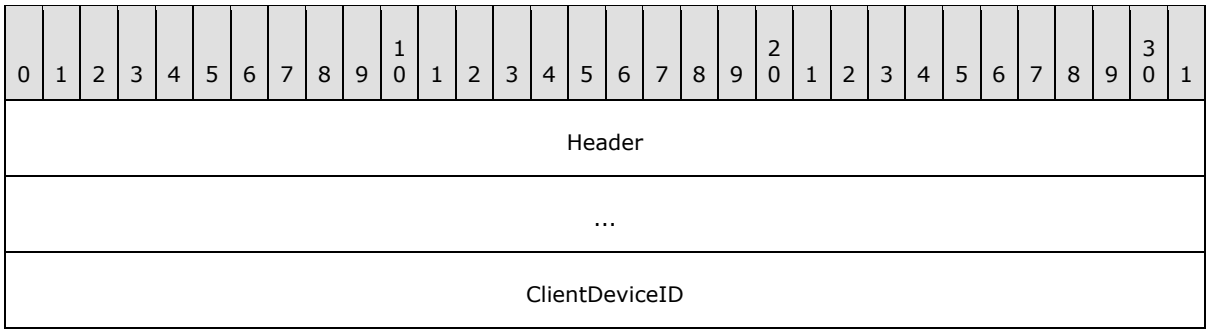
CustomFlagLength (4 bytes): A 32-bit unsigned integer. This field MUST be set to 0x00000004 because the **CustomFlag** field is hard-coded to be a 32-bit unsigned integer.

CustomFlag (4 bytes): A 32-bit unsigned integer that contains one of the following flags that indicates whether the device is an optional device. Optional devices are devices that the server MAY install; for all other devices, the server MUST install the device.

Value	Meaning
0x00000000	The device is not optional.
0x00000001	The device is optional.

2.2.1.3.2 Client Device Removal Message

A client sends this message to stop redirecting a particular device. The remote device is removed from the server's perspective and applications may no longer use it.



Header (8 bytes): The common message header (see section 2.2.1.1). The **PacketId** field MUST be set to IRPDR_ID_UNREDIRECT_DEVICE (0x00000068).

ClientDeviceID (4 bytes): A 32-bit unsigned integer. This value MUST specify the ID for the device to stop redirecting.

2.2.2 PNP Device I/O Subprotocol

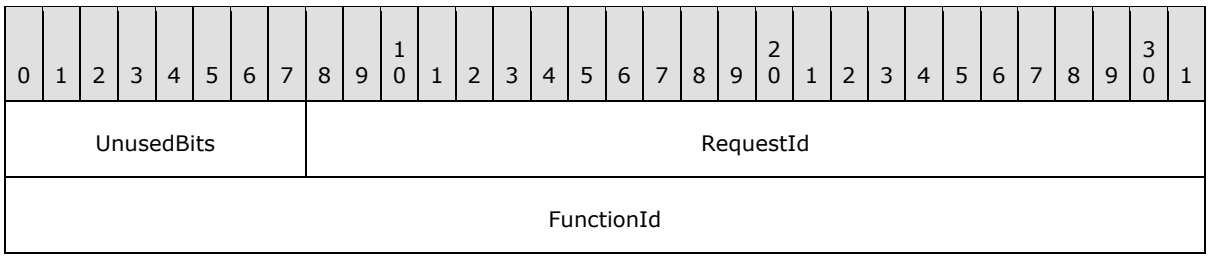
The messages in the following sections specify the common header and specific messages that make up the PNP Device I/O Subprotocol.

2.2.2.1 Shared Message Headers

All messages sent in the PNP Device I/O Subprotocol use either a Request or a Reply header, as specified in the following sections.

2.2.2.1.1 Server Message Header (SERVER_IO_HEADER)

All I/O Request messages (messages sent from the server to the client) use the following Request header.



UnusedBits (1 byte): An 8-bit reserved field. This value SHOULD be set to 0x00.

RequestId (3 bytes): A 24-bit unsigned integer. This server-generated value uniquely identifies the request. This value MUST be used to refer to the request in subsequent messages. A request ID my be reused after the reply message with that ID is received.

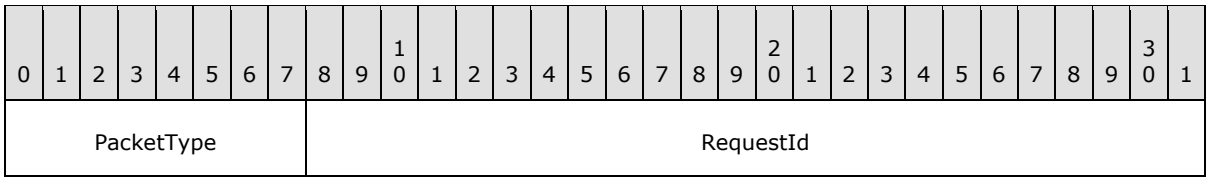
FunctionId (4 bytes): A 32-bit unsigned integer. This value identifies the function associated with the request. This value MUST be one of the following values.

Name	Value
READ_REQUEST	0x00000000

Name	Value
WRITE_REQUEST	0x00000001
IOCONTROL_REQUEST	0x00000002
GENERIC_IOCANCEL_REQUEST	0x00000003
CREATE_FILE_REQUEST	0x00000004
CAPABILITIES_REQUEST	0x00000005
SPECIFIC_IOCANCEL_REQUEST	0x00000006

2.2.2.1.2 Client Message Header (CLIENT_IO_HEADER)

All I/O Reply messages (messages from client to server) use the following Reply header.



PacketType (1 byte): An 8-bit unsigned integer that indicates the packet type. The field MUST contain one of the following values.

Value	Meaning
RESPONSE_PACKET 0x00	Indicates that the message is a response to an I/O request from the server.
CUSTOM_EVENT_PACKET 0x01	Indicates that the message is a custom event message generated by the client.

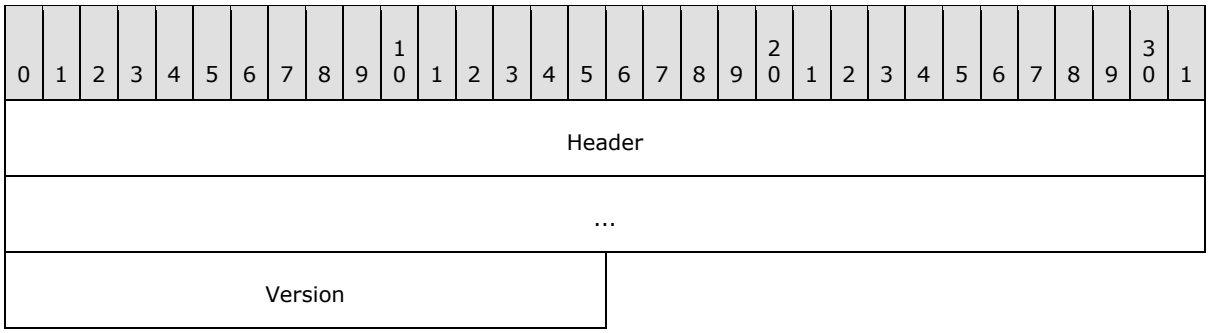
RequestId (3 bytes): A 24-bit unsigned integer. For I/O response messages, this value MUST contain the same value as the **RequestId** field in the SERVER_IO_HEADER of the corresponding request message. If the **PacketType** field contains 0x01, this is a Custom Event Message. This field is unused, MAY contain any value, and MUST be ignored.

2.2.2.2 Initialization Messages

The messages in the following sections are used to initialize the PNP Device I/O Subprotocol.

2.2.2.2.1 Server Capabilities Request Message

A server sends this message to indicate its version information and supported capabilities to the client.

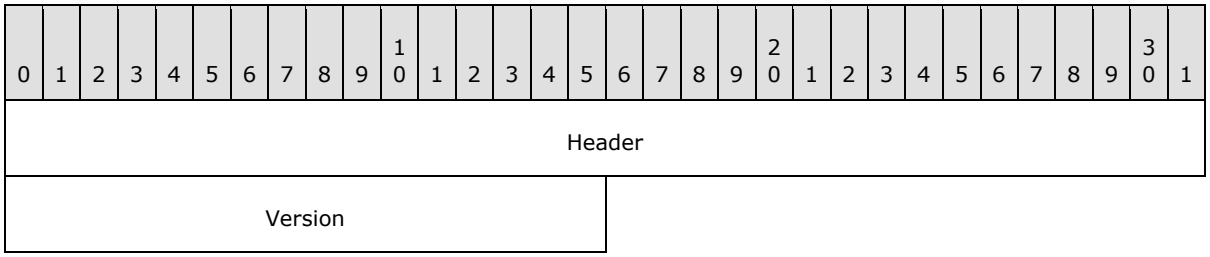


Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to CAPABILITIES_REQUEST (0x00000005).

Version (2 bytes): A 16-bit unsigned integer. This field MUST indicate the version of the server-side implementation of the PNP Device I/O Subprotocol.<6>

2.2.2.2.2 Client Capabilities Reply Message

The client replies to the server capabilities version with its own version and capabilities.



Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to RESPONSE_PACKET (0x00). The **RequestId** field MUST match the value in the **RequestId** field in the SERVER_IO_HEADER request header of the corresponding request packet.

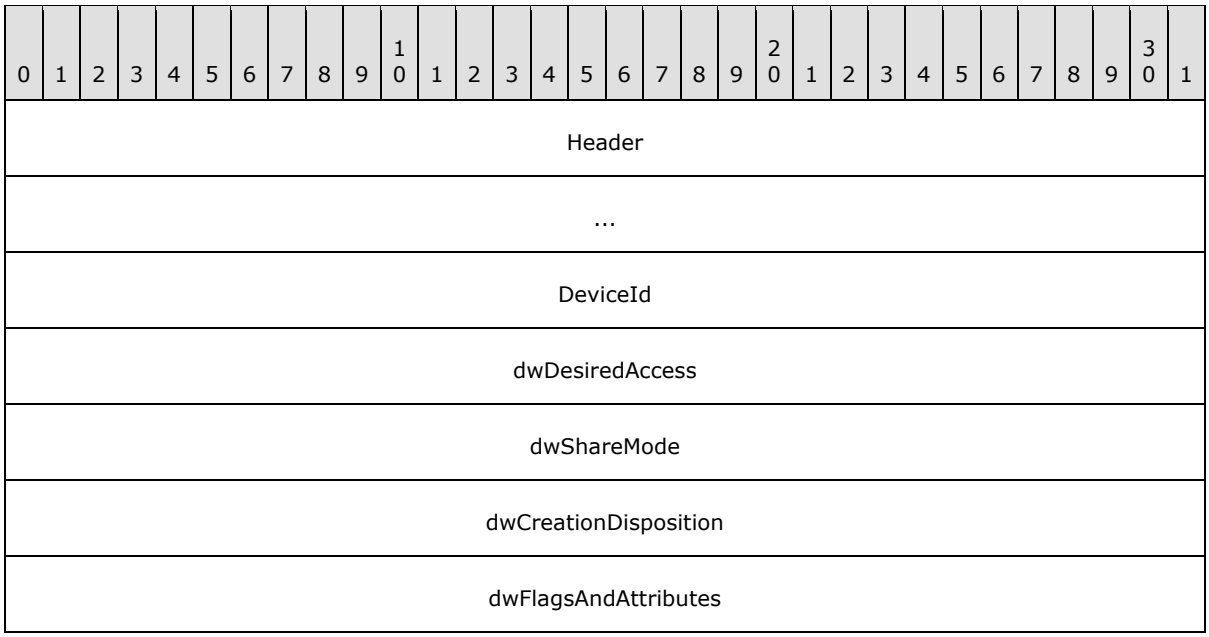
Version (2 bytes): A 16-bit unsigned integer. This field MUST indicate the version of the client-side implementation of the PNP Device I/O Subprotocol.<7>

2.2.2.3 Device I/O Messages

The messages in the following sections are used for device input and output operations in the PNP Device I/O Subprotocol.

2.2.2.3.1 CreateFile Request Message

A server sends this message to open a file handle on the client-side device. This message MUST be sent only once for a given connection within the dynamic virtual channel. A one-to-one correspondence exists between file handles opened on the client side and dynamic virtual channels used. All I/O traffic that is associated with a file handle must be done on the virtual channel used to create the file handle. As a result, to open multiple file handles, multiple dynamic virtual channels are established between client and server.



Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to CREATE_FILE_REQUEST (0x00000004).

DeviceId (4 bytes): A 32-bit unsigned integer. This field MUST identify the device redirected by the client. Device IDs are initially established as described in section 2.2.1.3.1.

dwDesiredAccess (4 bytes): A 32-bit unsigned integer. This is a flag field that indicates various access modes to use for creating and opening the file. This value SHOULD be set to 0xC0000000, meaning generic read and generic write.<8>

dwShareMode (4 bytes): A 32-bit unsigned integer that represents a set of bit flags indicating the sharing mode that the server application requested. This field SHOULD be composed of the bitwise OR of one or more of the following values.

Name	Value
FILE_SHARE_READ	0x00000001
FILE_SHARE_WRITE	0x00000002

dwCreationDisposition (4 bytes): A 32-bit unsigned integer that specifies the mode for creating or opening the file. This field SHOULD be one of the following values.<9>

Name	Value
CREATE_NEW	0x00000001
CREATE_ALWAYS	0x00000002
OPEN_EXISTING	0x00000003
OPEN_ALWAYS	0x00000004

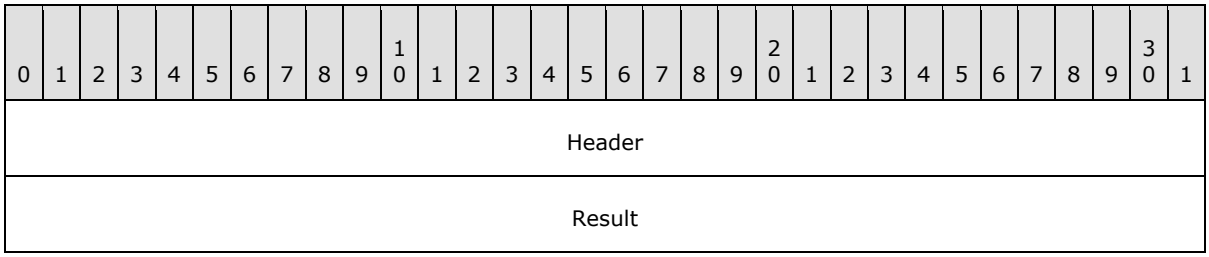
Name	Value
TRUNCATE_EXISTING	0x00000005

dwFlagsAndAttributes (4 bytes): A 32-bit unsigned integer that represents a set of bit flags specifying other flags and attributes associated with the request. This value MUST be composed of the bitwise OR of one or more of the following values.

Name	Value
FILE_ATTRIBUTE_DIRECTORY	0x00000010
FILE_ATTRIBUTE_ARCHIVE	0x00000020
FILE_ATTRIBUTE_DEVICE	0x00000040
FILE_ATTRIBUTE_NORMAL	0x00000080
FILE_FLAG_FIRST_PIPE_INSTANCE	0x00080000
FILE_FLAG_OPEN_NO_RECALL	0x00100000
FILE_FLAG_OPEN_REPARSE_POINT	0x00200000
FILE_FLAG_POSIX_SEMANTICS	0x01000000
FILE_FLAG_BACKUP_SEMANTICS	0x02000000
FILE_FLAG_DELETE_ON_CLOSE	0x04000000
FILE_FLAG_SEQUENTIAL_SCAN	0x08000000
FILE_FLAG_RANDOM_ACCESS	0x10000000
FILE_FLAG_NO_BUFFERING	0x20000000
FILE_FLAG_OVERLAPPED	0x40000000
FILE_FLAG_WRITE_THROUGH	0x80000000

2.2.2.3.2 CreateFile Reply Message

The client responds to the server create-file request with this message.

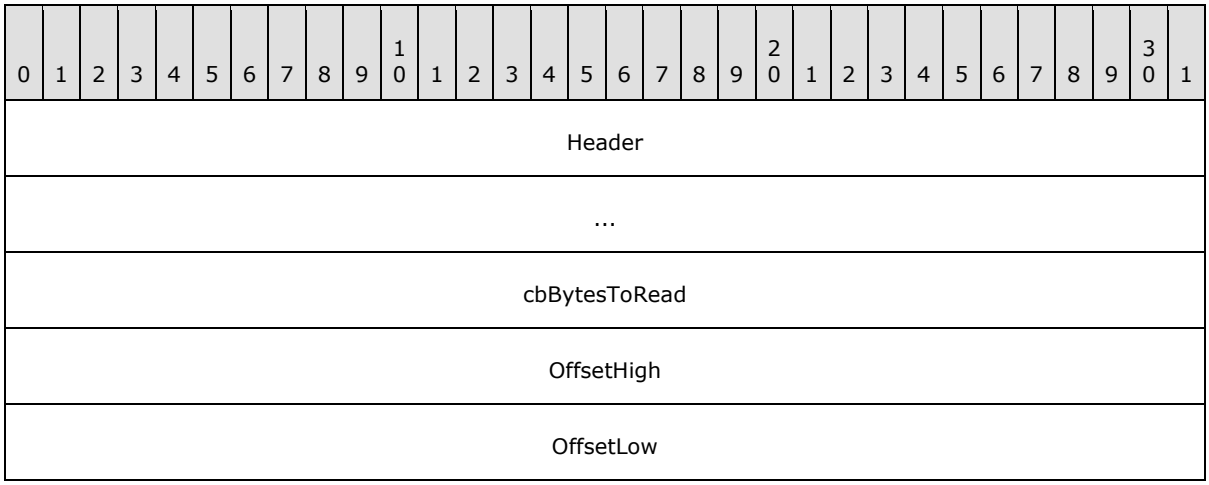


Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to RESPONSE_PACKET (0x00). The **RequestId** field MUST match the value in the **RequestId** field in the corresponding CreateFile Request Message.

Result (4 bytes): An HRESULT value that describes the result of the read operation.

2.2.2.3.3 Read Request Message

The server sends this message to request a read operation from the specified redirected client device.



Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to READ_REQUEST (0x00000000).

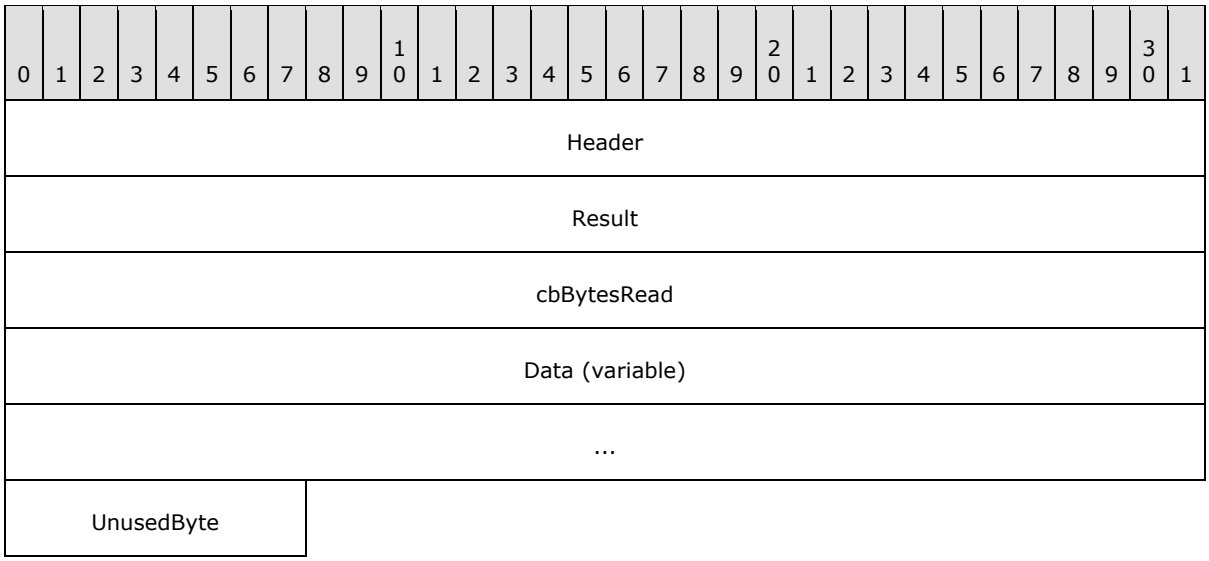
cbBytesToRead (4 bytes): A 32-bit unsigned integer. This field MUST specify how many bytes the server requested to read from the redirected client device.

OffsetHigh (4 bytes): A 32-bit unsigned integer. This field MUST specify the high offset value for the read operation.

OffsetLow (4 bytes): A 32-bit unsigned integer. This field MUST specify the low offset value for the read operation.

2.2.2.3.4 Read Reply Message

The client responds to the read file request from the server with this message.



Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to RESPONSE_PACKET (0x00). The **RequestId** field MUST match the value in the **RequestId** field in the corresponding Read Request Message.

Result (4 bytes): An HRESULT that describes the result of the read operation.

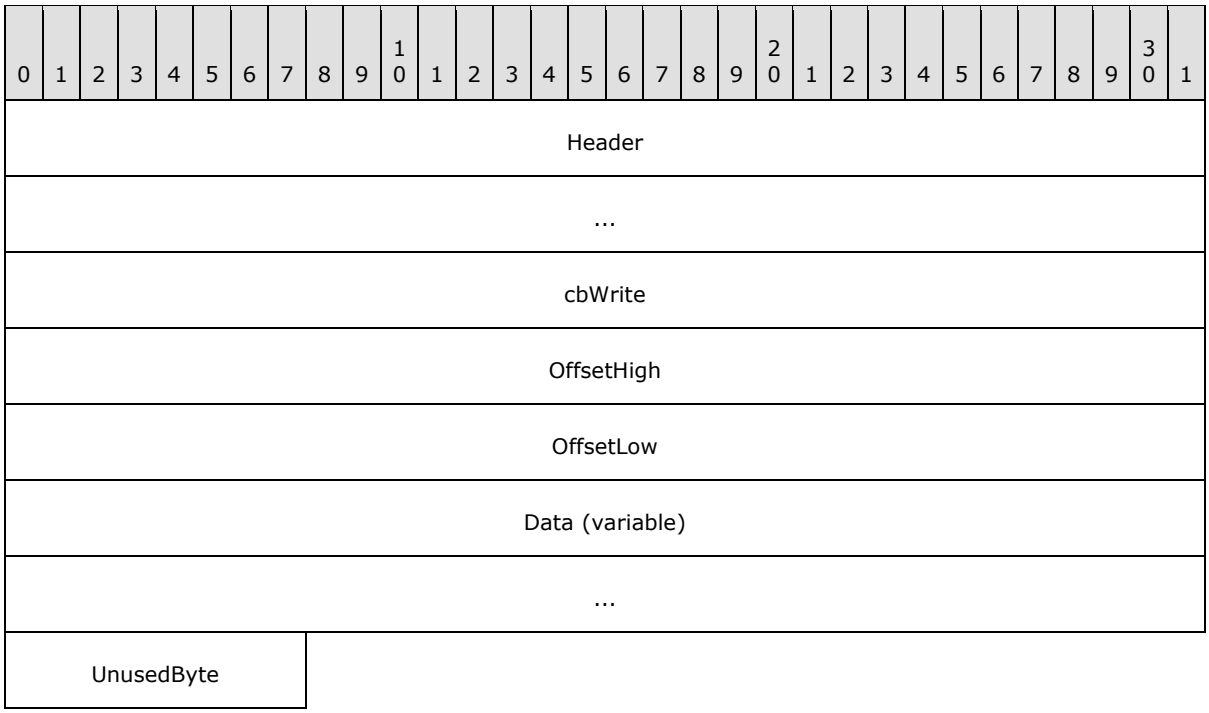
cbBytesRead (4 bytes): A 32-bit unsigned integer. This field MUST specify the number of bytes read.

Data (variable): An array of bytes. A variable-length field that MUST contain the data read from the client.

UnusedByte (1 byte): An 8-bit unsigned integer. This field is unused and MUST be ignored.

2.2.2.3.5 Write Request Message

The server sends this message to perform a write operation on a redirected client device.



Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to WRITE_REQUEST (0x00000001).

cbWrite (4 bytes): A 32-bit unsigned integer. This field MUST specify the size of the data to be written.

OffsetHigh (4 bytes): A 32-bit integer. This field MUST specify the high offset value for the write operation.

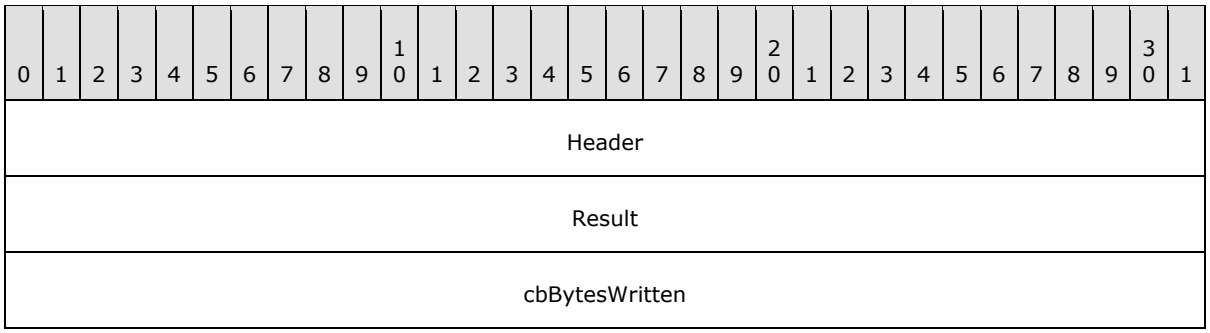
OffsetLow (4 bytes): A 32-bit unsigned integer. This field MUST specify the low offset value for the write operation.

Data (variable): An array of bytes. This field MUST contain the data to be written to the particular device.

UnusedByte (1 byte): An 8-bit unsigned integer. This field is unused and MUST be ignored.

2.2.2.3.6 Write Reply Message

A client responds to a Write Request message from the server with this message.



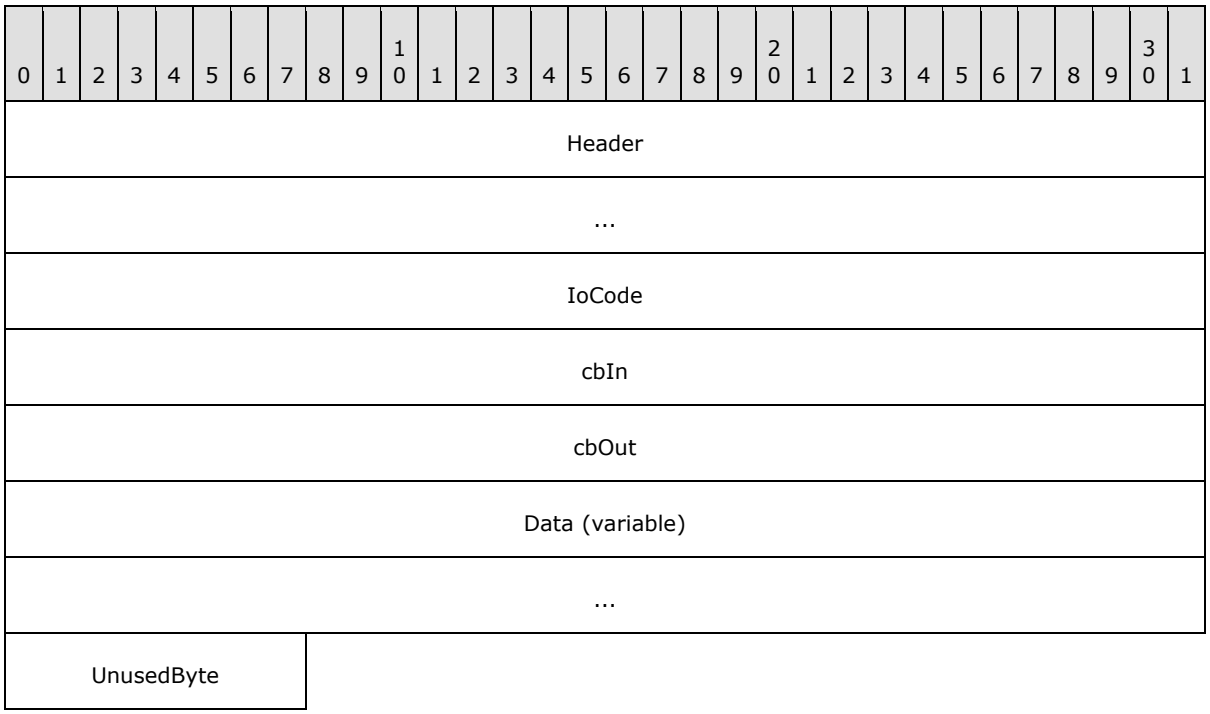
Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to RESPONSE_PACKET (0x00). The **RequestId** field MUST match the value in the **RequestId** field in the corresponding Write Request Message.

Result (4 bytes): An HRESULT value that specifies the result of the write operation.

cbBytesWritten (4 bytes): A 32-bit unsigned integer. This field MUST specify the size, in bytes, of the data written on the client device.

2.2.2.3.7 IOControl Request Message

A server sends this message to perform an IOControl operation on the client-side device.



Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to IOCONTROL_REQUEST (0x00000002).

IoCode (4 bytes): A 32-bit unsigned integer. This field MUST specify the I/O control code to be sent to the client device.

cbIn (4 bytes): A 32-bit unsigned integer. This field MUST specify the input buffer size.

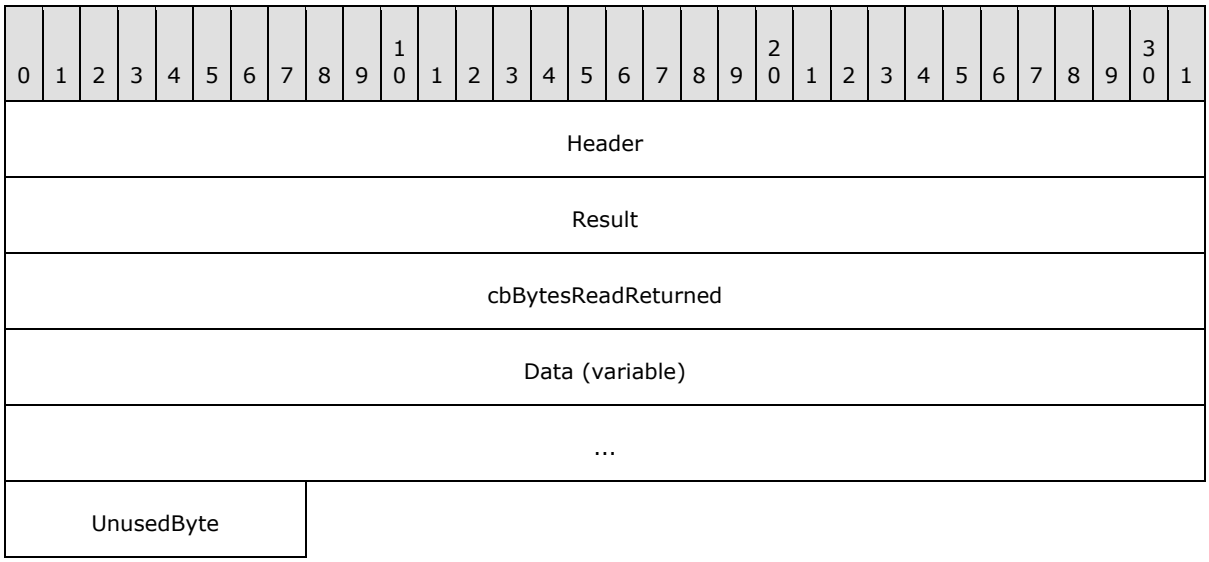
cbOut (4 bytes): A 32-bit unsigned integer. This field MUST specify the output buffer size.

Data (variable): An array of bytes. The **Data** buffer MUST contain only the input data.

UnusedByte (1 byte): An 8-bit unsigned integer. This field is unused, MAY be any value, and MUST be ignored.

2.2.2.3.8 IOControl Reply Message

The client responds to the IOControl Request message from the server with this message.



Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to RESPONSE_PACKET (0x00). The **RequestId** field MUST match the value in the **RequestId** field in the corresponding IOControl Request message.

Result (4 bytes): An HRESULT value that specifies the result of the IOControl operation.

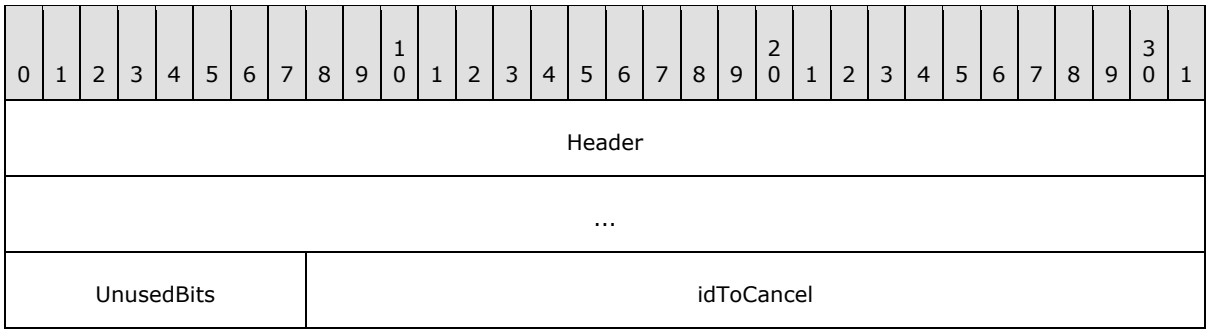
cbBytesReadReturned (4 bytes): A 32-bit unsigned integer. This field MUST specify the size, in bytes, of data read from the client device.

Data (variable): A variable-length array of bytes. This field MUST contain the data returned by the client IOControl operation.

UnusedByte (1 byte): An 8-bit unsigned integer. This field is unused and MUST be ignored.

2.2.2.3.9 Specific IoCancel Request Message

The server sends this message to the client to cancel a specific I/O request.



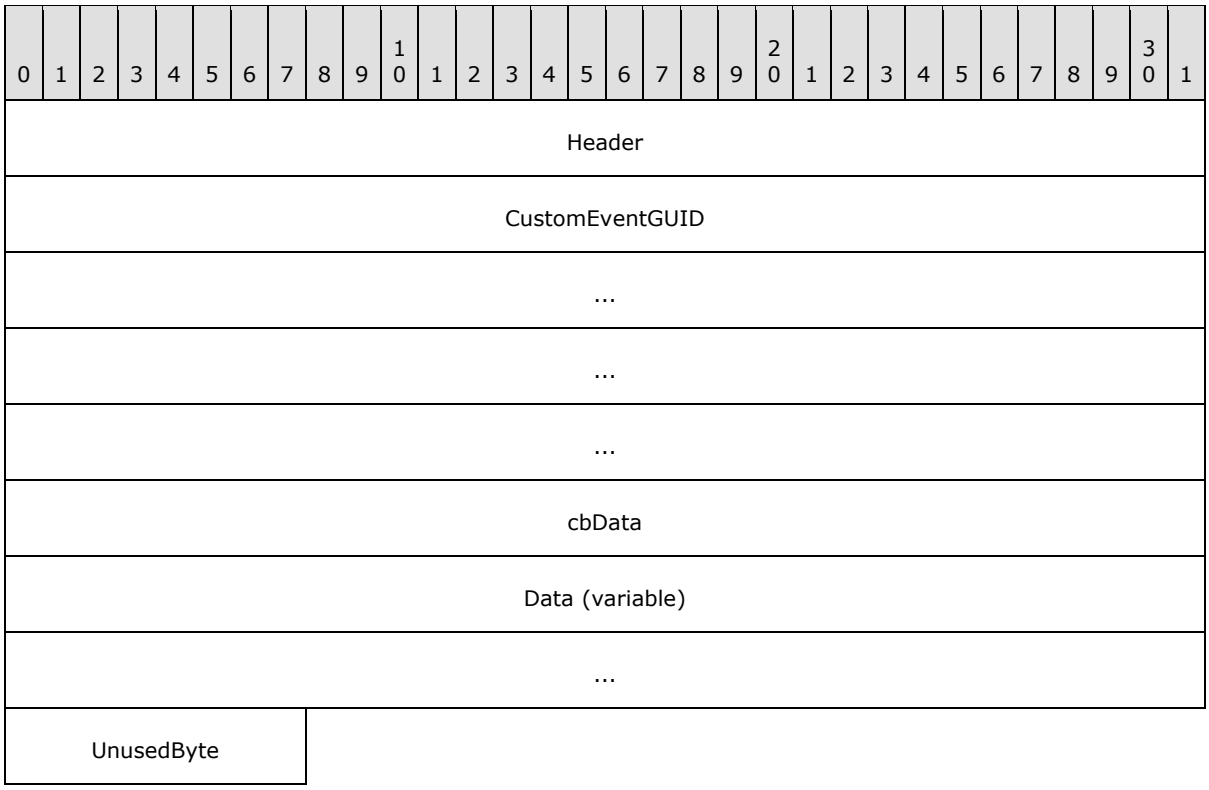
Header (8 bytes): A SERVER_IO_HEADER (section 2.2.2.1.1) request header. The **FunctionId** field MUST be set to SPECIFIC_IOCANCEL_REQUEST (0x00000006).

UnusedBits (1 byte): An 8-bit unsigned integer. This field is unused and SHOULD be set to 0x00.

idToCancel (3 bytes): A 24-bit unsigned integer. This field MUST specify the RequestId for the I/O request to cancel.

2.2.2.3.10 Client Device Custom Event Message

A client sends this message to the server in response to a custom event occurring on the client device. This message MUST be sent only if both the server and client protocol version is 6 or greater.



Header (4 bytes): A CLIENT_IO_HEADER (section 2.2.2.1.2) reply header. The **PacketType** field MUST be set to CUSTOM_EVENT_PACKET (0x01). The **RequestId** field SHOULD be set to 0.

CustomEventGUID (16 bytes): A GUID associated with the custom event generated.

cbData (4 bytes): A 32-bit unsigned integer. This field MUST specify the size of the data associated with the custom event.

Data (variable): A variable-length array of bytes. This field MUST contain the data associated with the custom event.

UnusedByte (1 byte): An 8-bit unsigned integer. This field is unused and MUST be ignored.

3 Protocol Details

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.

Client device ID: A unique device ID that both the server and client maintain for each device within a session. The client generates this ID for each device when it sends device information in a Client Device Addition message. For subsequent operations on the devices, both client and server use this ID to refer to the device.

Client device list: Both client and server maintain a list of devices being redirected. This list contains the client device ID for each redirected device. Devices are added to the list when they are redirected and removed when redirection is canceled.

Request ID: For I/O request calls, the server generates a unique 24-bit ID and sends it with the I/O request to the client. The client and server use this ID to refer to the request in subsequent messages. When a request is sent to the client, the server adds it to a list of outstanding requests. When the client completes the request or when the request is canceled, the server removes the entry for the request from the outstanding requests list. A request ID may be reused after the reply message with that ID is received.

3.1.2 Timers

No common timers are used. Individual device drivers MAY implement time-out logic for I/O requests; however, the operation of these drivers is external to this specification.

3.1.3 Initialization

The dynamic virtual channel must be established by using the parameters described in section 2.1 before the protocol operation can commence.

3.1.4 Higher-Layer Triggered Events

No higher-layer triggered events are used.

3.1.5 Message-Processing Events and Sequencing Rules

There are no common message-processing events or sequencing rules. See sections 3.2.5 and 3.3.5 for client- and server-specific message processing.

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 PNP Device Info Subprotocol

3.2.5.1.1 Initialization Messages

Initialization messages exchange the basic information necessary to establish the connection and to perform capabilities negotiation.

3.2.5.1.1.1 Processing a Server Version Message

The structure and fields of the Server Version message are described in section 2.2.1.2.1.

The Server Version message MUST be the first message that the client receives in the protocol sequence. The client uses the version and capabilities of the server to discover what messages the protocol supports. For example, if the client receives a major version of 6 or above (future versions of the protocol may send a higher version, although current ones do not) from the packet described in section 2.2.2.2.1, the client MAY send packets that describe custom events, as described in section 2.2.2.3.10. However, if the major version is below 6, the client MUST not send packets that describe custom events.

Similarly, the client MUST acknowledge the message by sending its own version and capabilities information. This way, the server knows what messages the client supports. Future versions of the protocol MAY support new packets that current versions do not support. As a result, this negotiation is important to ensure that no packets are sent from one side that the other cannot interpret.

3.2.5.1.1.2 Sending a Client Version Message

The structure and fields of the Client Version message are described in section 2.2.1.2.2.

No client-specific events or rules are required.

3.2.5.1.1.3 Processing an Authenticated Client Message

The structure and fields of the Authenticated Client message are specified in section 2.2.1.2.3.

The server sends the Authenticated Client message after it authenticates the client to the server session. The client MUST not send any device addition or removal messages until it receives this message. Only after receiving this message MAY the client send a Client Device Addition message (section 2.2.1.3.1).

3.2.5.1.2 Device Addition and Removal Messages

3.2.5.1.2.1 Sending a Client Device Addition Message

The structure and fields of the Client Device Addition messageare as specified in section 2.2.1.3.1.

The client MUST generate and assign a unique client device ID for each of the devices that it wants to redirect to the server. This message MUST be sent only after the client receives an Authenticated Client message (section 2.2.1.2.3).

3.2.5.1.2.2 Sending a Client Device Removal Message

The structure and fields of the Client Device Removal message are as specified in section 2.2.1.3.2.

Before the client sends this message to stop redirecting a particular device, the corresponding device MUST have previously been sent as part of a Client Device Addition message (section 2.2.1.3.1).

3.2.5.2 PNP Device I/O Subprotocol

3.2.5.2.1 Initialization Messages

These messages establish the logical connection between server and client, in addition to capabilities. Initialization messages MUST be sent immediately after creating a new dynamic channel connection within the **FileRedirectorChannel** channel. A new channel connection MUST be established for each CreateFile call. These messages are generally followed by the CreateFile message and then by Read, Write, or IOControl messages.

3.2.5.2.1.1 Processing a Server Capabilities Request Message

The structure and fields of the Server Capabilities Request message are defined in section 2.2.2.2.1.

This MUST be the first message that a client receives on each connection within the PNP Device Info Subprotocol. The client inspects the version and capabilities fields. The client MUST reply with its own capabilities by sending a Client Capabilities Reply message.

3.2.5.2.1.2 Sending a Client Capabilities Reply

The structure and fields of the Client Capabilities Reply message are defined in section 2.2.2.2.2.

This message MUST be sent only after receiving a Server Capabilities Request message (section 2.2.2.2.1).

3.2.5.2.2 Device I/O Messages

The device I/O messages in the PNP Device Info Subprotocol (section 2.2.1) are used to perform real I/O operations on the client devices and to return the result to the server.

3.2.5.2.2.1 Processing a CreateFile Request Message

The structure and fields of the CreateFile Request message are defined in section 2.2.2.3.1.

The client MUST use the client device ID passed in the CreateFile Request message to identify the device to use. The client interacts with the local device driver, using the attributes and flags specified in the CreateFile Request message, to service the I/O request.

3.2.5.2.2.2 Sending a CreateFile Reply Message

The structure and fields of the CreateFile Reply message are defined in section 2.2.2.3.2.

The result of the client's interaction with the local device driver in servicing the CreateFile Request message (section 2.2.2.3.1) MUST be returned by the client in the CreateFile Reply message. The client MUST maintain the association of the file handle obtained with the dynamic virtual channel connection on which it received the CreateFile Request message, because all I/O requests on the connection are associated with the file handle.

3.2.5.2.2.3 Processing a Read Request Message

The structure and fields of the Read Request message are described in section 2.2.2.3.3.

This message MUST be received only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. On receiving the Read Request message, the client MUST use the associated file handle and the parameters specified in the Read Request message to interact with the local device driver in servicing this request.

3.2.5.2.2.4 Sending a Read Reply Message

The structure and fields of the Read Reply message are described in section 2.2.2.3.4.

The client MUST use the **RequestId** received in the corresponding Read Request message when constructing this reply. The result of the Read operation performed, along with all data read, MUST be returned in the response message.

3.2.5.2.2.5 Processing a Write Request Message

The structure and fields of the Write Request message are described in section 2.2.2.3.5.

This message MUST be received only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. On receiving the Write Request message, the client MUST use the associated file handle and the parameters specified in the Write Request message to interact with the local device driver in servicing this request.

3.2.5.2.2.6 Sending a Write Reply Message

The structure and fields of the Write Reply message are described in section 2.2.2.3.6.

The client MUST use the RequestId received in the corresponding Write Request message when constructing this reply. The result of the Write operation performed MUST be returned in the response message.

3.2.5.2.2.7 Processing an IOControl Request Message

The structure and fields of the IOControl Request message are described in section 2.2.2.3.7.

This message MUST be received only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. On receiving the IOControl Request message, the client MUST use the associated file handle and the parameters specified in the IOControl Request message to interact with the local device driver in servicing this request.

The **Data** field MUST contain input data of the size specified in the **cbIn** field, followed by output data of the size specified in the **cbOut** field.

3.2.5.2.2.8 Sending an IOControl Reply Message

The structure and fields of the IOControl Reply message are described in section 2.2.2.3.8.

The client MUST use the **RequestId** received in the corresponding IOControl Request message when constructing this reply. The result of the IOControl operation performed and any output data MUST be returned in the response message.

3.2.5.2.2.9 Processing a Specific IoCancel Request Message

The structure and fields of the Specific IoCancel Request message are described in section 2.2.2.3.9.

This message MUST be received only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. On receiving this message, the client MUST cancel the I/O operation associated with the device that is identified by the value in the **RequestId** field.

3.2.5.2.2.10 Sending a Client Device Custom Event Message

The structure and fields of the Client Device Custom Event message are described in section 2.2.2.3.10.

When a redirected device generates any custom PNP event, the client MUST notify the server of the event by sending a Client Device Custom Event message to the server. The message MUST contain all the data regarding the custom PNP event, as described in section 2.2.2.3.10. This message MUST be sent only if the protocol version running on both the client and server is 6, or greater. The version number is exchanged in packets described in sections 2.2.2.2.1 and 2.2.2.2.2.

3.2.6 Timer Events

No client timer events are used.

3.2.7 Other Local Events

No additional client events are used.

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

No server timers are used.

3.3.3 Initialization

Initialization is specified in section 3.1.3.

3.3.4 Higher-Layer Triggered Events

No higher-layer triggered events are used.

3.3.5 Message-Processing Events and Sequencing Rules

3.3.5.1 PNP Device Info Subprotocol

3.3.5.1.1 Initialization Messages

This section contains information about sending version request messages, processing version response messages, sending authenticated client messages, and processing device addition and device removal messages.

3.3.5.1.1.1 Sending a Server Version Message

The structure and fields of the Server Version message are described in section 2.2.1.2.1.

This MUST be the first message that the server sends after creating a dynamic virtual channel connection with the client. The server indicates its version and capabilities in this message.

3.3.5.1.1.2 Processing a Client Version Message

The structure and fields of the Client Version message are described in section 2.2.1.2.2.

When the server receives a Client Version message, the server MUST use the version received from the client to discover what messages the client understands. Although there is currently only one possible client protocol version, future protocol versions MAY have packets that the current version will not understand.

The server MUST receive this message before any meaningful exchange can take place.

3.3.5.1.1.3 Sending an Authenticated Client Message

The structure and fields of the Authenticated Client message are described in section 2.2.1.2.3.

The server SHOULD not accept any device redirection commands until a user has logged on to the server session. This ensures that nonauthenticated users cannot cause a denial-of-service attack by sending huge volumes of device addition or removal requests. When a user logs on to the server session, the server MUST send the Authenticated Client message, which indicates to the client that the server is ready to process device addition or removal messages.

3.3.5.1.2 Device Addition and Removal Messages

The following messages are processed only after the client and server have completed initial versioning.

3.3.5.1.2.1 Processing a Client Device Addition Message

The structure and fields of the Client Device Addition message are described in section 2.2.1.3.1.

For each device contained in the **DeviceDescriptions** field of the Client Device Addition message, the server MUST create a remote device instance on the server to represent the client-side physical devices. The server MUST also maintain a client device ID for each device. A one-to-one correspondence exists between remote devices and client device IDs. This ID MUST be used to refer to a particular device when making I/O calls.

3.3.5.1.2.2 Processing a Client Device Removal Message

The structure and fields of the Client Device Removal message are described in section 2.2.1.3.2.

For a device already instantiated on the server and identified by the value in the **ClientDeviceId** field, the server MUST remove all references to the remote device when this message is received.

3.3.5.2 Device I/O Subprotocol

3.3.5.2.1 Initialization Messages

3.3.5.2.1.1 Sending a Server Capabilities Request Message

The structure and fields of the Server Capabilities Request message are described in section 2.2.2.2.1.

This MUST be the first message that the server sends for each dynamic virtual channel connection it establishes with the client.

3.3.5.2.1.2 Processing a Client Capabilities Reply Message

The structure and fields of the Client Capabilities Reply message are described in section 2.2.2.2.2. The server MUST receive this message prior to any other message that the client sends. The server MUST not complete the initialization of the remote device until it receives this message.

After receiving the Client Capabilities Reply message, the server MAY begin to process I/O messages. The server MUST not process any I/O messages until it receives a version and capabilities from the client.

3.3.5.2.2 Device I/O Messages

3.3.5.2.2.1 Sending a CreateFile Request Message

The structure and fields of the CreateFile Request message are described in section 2.2.2.3.1.

The server sends a CreateFile Request message to open or create a file on the client-side device on behalf of an application. The server MUST pass the client device ID to identify the device. The server MUST generate a unique ID for this request and pass it in the **RequestId** field of the SERVER_IO_HEADER along with any flags or attributes for the create-file request.

3.3.5.2.2.2 Processing a CreateFile Reply Message

The structure and fields of the CreateFile Reply message are described in section 2.2.2.3.2.

No server-specific events or rules are required other than that the server MUST pass the results of the operation contained in the reply to the actual application that made the create-file request.

3.3.5.2.2.3 Sending a Read Request Message

The structure and fields of the Read Request message are described in section 2.2.2.3.3.

This message MUST be sent only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. The server MUST generate a unique **RequestId** for this request and MUST specify the number of bytes to read. The server also stores all necessary information required to complete the request (for example, a data buffer to store information and the location of a variable to store the result), and associates this information with the **RequestId**.

3.3.5.2.2.4 Processing a Read Reply Message

The structure and fields of the Read Reply message are described in section 2.2.2.3.4.

To process this reply, the server MUST use the **RequestId** specified in the reply message to find the associated information stored after sending the request message. With this information, the server completes the original request. The server MUST redirect the result of the Read operation contained in the reply to the actual application that made the read request.

3.3.5.2.2.5 Sending a Write Request Message

The structure and fields of the Write Request message are described in section 2.2.2.3.5.

This message MUST be sent only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. The server MUST generate and pass a unique **RequestId** for this request, MUST specify the number of bytes to write in the **cbWrite** field, and MUST pass the actual data to be written in the **Data** buffer field. The server also stores all necessary information required to complete the request (for example, the location of a variable to store the result), and associates this information with the **RequestId**.

3.3.5.2.2.6 Processing a Write Reply Message

The structure and fields of the Write Reply message are described in section 2.2.2.3.6.

To process this reply, the server MUST use the **RequestId** specified in the reply message to find the associated information stored after sending the request message. With this information, the server completes the original request. The server MUST redirect the result of the Write operation contained in the reply to the actual application that made the write request.

3.3.5.2.2.7 Sending an IOControl Request Message

The structure and fields of the IOControl Request message are described in section 2.2.2.3.7.

This message MUST be sent only after the CreateFile request-response sequence has been sent, establishing a file handle for I/O on this connection. The server MUST generate a **RequestId** for this request and the server MUST pass along the rest of the IOControl parameters. The server also stores all necessary information required to complete the request (for example, the location of a variable to store the result), and associates this information with the **RequestId**.

3.3.5.2.2.8 Processing an IOControl Reply Message

The structure and fields of the IOControl Reply message are described in section 2.2.2.3.8.

To process this reply, the server MUST use the **RequestId** specified in the reply message to find the associated information stored after sending the request message. With this information, the server

completes the original request. The server MUST redirect the result of the I/O operation contained in the reply to the actual application that made the I/O request.

3.3.5.2.2.9 Sending a Specific IoCancel Request Message

The structure and fields of the Specific IoCancel Request message are described in section 2.2.2.3.9.

No server-specific events or rules are required.

3.3.5.2.2.10 Processing a Client Device Custom Event Message

The structure and fields of the Client Device Custom Event message are described in section 2.2.2.3.10.

On receiving a Client Device Custom Event message, the server MUST generate a similar event (using the parameters contained in the message) on the server system, so that the application registered for such an event is notified. This event MUST be generated only if the protocol version running on both the client side and server side is 6, or greater.

3.3.6 Timer Events

No server timer events are used.

3.3.7 Other Local Events

No additional server events are used.

4 Protocol Examples

4.1 PNP Device Redirection Initialization Sequence

```
(1) Server Version Message

ChannelName = PNPDR,20,server to client
00000000 14 00 00 00 65 00 00 00 01 00 00 00 05 00 00 00 ....e....
00000010 01 00 00 00                                     ....
14 00 00 00  -> Size = 0x00000014
65 00 00 00  -> Packet Id = 0x00000065
01 00 00 00  -> Major Version = 0x00000001
05 00 00 00  -> Minor Version = 0x00000005
01 00 00 00  -> Capabilities = 0x00000001


(2) Client Version Message

ChannelName = PNPDR,20,client to server
00000000 14 00 00 00 65 00 00 00 01 00 00 00 05 00 00 00 ....e....
00000010 01 00 00 00                                     ....

14 00 00 00  -> Size = 0x00000014
65 00 00 00  -> Packet Id = 0x00000065
01 00 00 00  -> Major Version = 0x00000001
05 00 00 00  -> Minor Version = 0x00000005
01 00 00 00  -> Capabilities = 0x00000001


(3) Authenticated Client Message

ChannelName = PNPDR,8,server to client
00000000 08 00 00 00 67 00 00 00                         ....g...

08 00 00 00  -> Size = 0x00000008
67 00 00 00  -> Packet Id = 0x00000067
```

4.2 Device Addition and Removal Messages

```
(1) Client Device Addition Message

ChannelName = PNPDR,106,client to server
00000000 6a 00 00 00 66 00 00 00 01 00 00 00 04 00 00 00 j...f...........
00000010 56 00 00 00 10 00 00 00 46 9c 4a 2b 8d 65 f2 4a V.......F.J+.e.J
00000020 a9 1d 1e 69 18 61 70 6c 12 00 00 00 57 00 55 00 ...i.apl....W.U.
00000030 44 00 46 00 5c 00 4c 00 42 00 00 00 00 00 00 00 D.F.\.L.B.......
00000040 00 00 1c 00 00 00 54 00 73 00 20 00 46 00 61 00 ......T.s. .F.a.
00000050 6b 00 65 00 20 00 44 00 65 00 76 00 69 00 63 00 k.e. .D.e.v.i.c.
00000060 65 00 04 00 00 00 02 00 00 00                   e.........

6a 00 00 00  -> Size = 0x0000006a
66 00 00 00  -> Packet Id = 0x00000066
01 00 00 00  -> Device Count = 0x00000001

PNP_DEVICE_DESCRIPTION (variable size)
04 00 00 00  -> Client Device Id = 0x00000004
56 00 00 00  -> Data Size = 0x00000056
10 00 00 00  -> cbInterface Length = 0x00000010
```

```
46 9c 4a 2b  -> Interface GUID array (variable size=cbInterface Length)
8d 65 f2 4a  -> Interface GUID array (continued)
a9 1d 1e 69  -> Interface GUID array (continued)
18 61 70 6c  -> Interface GUID array (continued)
12 00 00 00  -> cbHardwareID Length = 0x00000012
57 00 55 00  -> Hardware ID (variable size=cbHardwareID Length)
44 00 46 00  -> Hardware ID (continued)
5c 00 4c 00  -> Hardware ID (continued)
42 00 00 00  -> Hardware ID (continued)
00 00        -> Hardware ID (continued)
00 00 00 00  -> cbCompatId Length = 0x00000000
1c 00 00 00  -> cbDeviceDescriptionLength = 0x0000001c
54 00 73 00  -> Device Description (variable size=cbDeviceDescription Length)
20 00 46 00  -> Device Description (continued)
61 00 6b 00  -> Device Description (continued)
65 00 20 00  -> Device Description (continued)
44 00 65 00  -> Device Description (continued)
76 00 69 00  -> Device Description (continued)
63 00 65 00  -> Device Description (continued)
04 00 00 00  -> Custom flag length = 0x00000004
02 00 00 00  -> Custom flag = 0x00000002
```

(2) Client Device Removal Message

```
ChannelName = PNPDR,12,client to server
00000000 0c 00 00 00 68 00 00 00 04 00 00 00             ....h....

0c 00 00 00  -> Size = 0x0000000c
68 00 00 00  -> Packet Id = 0x00000068
04 00 00 00  -> Client Device Id = 0x00000004
```

4.3 Capabilities Initialization Messages

(1) Server Capabilities Request Message

```
ChannelName = FileRedirectorChannel,10,server to client
00000000 00 00 00 00 05 00 00 00 05 00                   .........

00           -> Unused = 0x00
00 00 00     -> Request Id = 0x000000
05 00 00 00  -> Function Id = 0x00000005
05 00        -> Version = 0x0005
```

(2) Client Capabilities Reply Message

```
ChannelName = FileRedirectorChannel,6, client to server
00000000 00 00 00 00 05 00                               ......

00           -> PacketType = 0x00
00 00 00     -> Request Id = 0x000000
05 00        -> Version = 0x0005
```

4.4 Device I/O Messages

```
(1) CreateFile Server Request Message

ChannelName = FileRedirectorChannel,28,server to client
00000000 00 00 00 00 04 00 00 00 04 00 00 00 00 00 00 c0 ................
00000010 03 00 00 00 03 00 00 00 80 00 00 40             ...........@

00           -> Unused = 0x00
00 00 00     -> Request Id = 0x000000
04 00 00 00  -> Function Id = 0x00000004
04 00 00 00  -> Device Id = 0x00000004
00 00 00 c0  -> dwDesiredAccess = 0xc0000000
03 00 00 00  -> dwShareMode = 0x00000003
03 00 00 00  -> dwCreationDisposition = 0x00000003
80 00 00 40  -> dwFlagsAndAttributes = 0x40000080



(2) CreateFile Client Response Message

ChannelName = FileRedirectorChannel,8,client to server
00000000 00 00 00 00 00 00 00 00                         ........

00           -> PacketType = 0x00
00 00 00     -> Request Id = 0x000000
00 00 00 00  -> Result (HRESULT) = 0x00000000




(3) Read Request Message

ChannelName = FileRedirectorChannel,20,server to client
00000000 00 00 00 00 00 00 00 00 08 00 00 00 01 00 00 70 ...............p
00000010 ff ff ff ff                                     ....

00          -> Unused = 0x00
00 00 00    -> Request Id = 0x000000
00 00 00 00 -> Function Id = 0x00000000
08 00 00 00 -> cbBytesToRead = 0x00000008
01 00 00 70 -> Offset High = 0x70000001
ff ff ff ff -> Offset Low = 0xffffffff
(

4) Read Reply Message

ChannelName = FileRedirectorChannel,21,client to server
00000000 00 00 00 00 00 00 00 00 08 00 00 00 2d 00 00 00 ............-...
00000010 20 72 00 00 00                                   r...

00          -> PacketType = 0x00
00 00 00    -> Request Id = 0x000000
00 00 00 00 -> Result = 0x00000000
08 00 00 00 -> cbBytesRead = 0x00000008
2d 00 00 00 -> Data (variable size = cbBytesRead)
20 72 00 00 -> Data (continued)
00          -> Unused = 0x00


(5) Write Request Message

ChannelName = FileRedirectorChannel,29,server to client
```

```
00000000 00 00 00 00 01 00 00 00 08 00 00 00 00 00 00 00 ................
00000010 01 00 00 00 01 00 00 00 2d 00 00 00 20          ........-...

00          -> Unused = 0x00
00 00 00    -> Request Id = 0x000000
01 00 00 00 -> Function Id = 0x00000001
08 00 00 00 -> cbWrite = 0x00000008
00 00 00 00 -> Offset High = 0x00000000
01 00 00 00 -> Offset Low = 0x00000001
01 00 00 00 -> Data (variable size = cbWrite)
2d 00 00 00 -> Data (continued)
20          -> Unused = 0x20
```

(6) Write Reply Message

```
ChannelName = FileRedirectorChannel,12,client to server
00000000 00 00 00 00 00 00 00 00 08 00 00 00              ............

00          -> PacketType = 0x00
00 00 00    -> Request Id = 0x000000
00 00 00 00 -> Result = 0x00000000
08 00 00 00 -> cbBytesWritten = 0x00000008
```

(7) IoControl Request Message

```
ChannelName = FileRedirectorChannel,37,server to client
00000000 00 00 00 00 02 00 00 00 40 24 22 00 10 00 00 00 ........@$".....
00000010 08 00 00 00 02 00 00 00 2d 00 00 00 20 72 00 00 ........-... r..
00000020 6c 59 00 00 00                                  lY...

00          -> Unused = 0x00
00 00 00    -> Request Id = 0x000000
02 00 00 00 -> Function Id = 0x00000002
40 24 22 00 -> IoCode = 0x00222440
10 00 00 00 -> cbIn = 0x00000010
08 00 00 00 -> cbOut = 0x00000008
02 00 00 00 -> Data (variable size = cbIn)
2d 00 00 00 -> Data (continued)
20 72 00 00 -> Data (continued)
6c 59 00 00 -> Data (continued)
00          -> Unused = 0x00
```

(8) IoControl Reply Message

```
ChannelName = FileRedirectorChannel,21,client to server
00000000 00 00 00 00 00 00 00 00 08 00 00 00 2d 00 00 00 ............-...
20 72 00 00 00                                    r...

00          -> PacketType = 0x00
00 00 00    -> Request Id = 0x000000
00 00 00 00 -> Result = 0x00000000
08 00 00 00 -> cbBytesReadReturned = 0x00000008
2d 00 00 00 -> Data (variable size)
20 72 00 00 -> Data (continued)
00          -> Unused = 0x00
```

(9) Server IoCancel Request Message

```
ChannelName = FileRedirectorChannel,12,server to client
00000000 ff ff ff ff 06 00 00 00 00 00 00 00

ff          -> Unused = 0xff
ff ff ff    -> Request Id = 0xffffff
06 00 00 00 -> Function Id = 0x00000006
00          -> Unused = 0x00
00 00 00    -> IdToCancel = 0x000000
```

(10) Client Device Custom Event Message

```
ChannelName = FileRedirectorChannel,33,client to server
00000000 00 00 00 01 11 11 11 11 80 80 5f 42 92 2a da bf ..........  B.*..
00000010 3d e3 f6 9a 08 00 00 00 20 4c 0f 00 c4 00 0f 00 =.......  L......
00000020 00

01          -> PacketType = 0x01
00 00 00    -> Request Id = 0x000000
11 11 11 11 -> CustomEventGUID (128 bit)
80 80 5f 42 -> CustomEventGUID (continued)
92 2a da bf -> CustomEventGUID (continued)
3d e3 f6 9a -> CustomEventGUID (continued)
08 00 00 00 -> cbData = 0x00000008
20 4c 0f 00 -> Data (variable size = cbData)
c4 00 0f 00 -> Data (continued)
00          -> Unused = 0x00
```

5 Security

5.1 Security Considerations for Implementers

There are no security considerations for the Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension because all traffic is secured by the underlying Remote Desktop Protocol (RDP) core protocol. For more information about implemented security-related mechanisms, see [MS-RDPBCGR] section 5.

5.2 Index of Security Parameters

None.

6 Appendix A: 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: The Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension is available beginning in Windows Vista.

<2> Section 2.2.1.2.1: In the Windows implementation of this protocol, this value MUST be 0x00000001.

<3> Section 2.2.1.2.1: In the Windows implementation of this protocol, this value MUST be 0x00000005.

<4> Section 2.2.1.2.2: In the Windows implementation of this protocol, this value MUST be 0x00000001.

<5> Section 2.2.1.2.2: In the Windows implementation of this protocol, this value MUST be 0x00000005.

<6> Section 2.2.2.2.1: In the Windows implementation of this protocol, this field MUST contain one of the following values:

Value	Meaning
0x0004	This version does not support custom event redirection.
0x0006	This version supports custom event redirection. Only Windows Server 2008 supports custom event redirection.

<7> Section 2.2.2.2.2: In the Windowsimplementation of this protocol, this field MUST contain one of the following values.

Value	Meaning
0x0004	This version does not support custom event redirection.
0x0006	This version supports custom event redirection. Only Windows Server 2008 supports custom event redirection.

<8> Section 2.2.2.3.1: In the Windows implementation of this protocol, this value is set to 0xC0000000, meaning generic read and generic write.

<9> Section 2.2.2.3.1: The Windows implementation of this protocol sets this field to 0x00000003 (OPEN_EXISTING).

7 Index

R

S

T

V

W