[MS-RDPEUSB]:
Remote Desktop Protocol:
USB Devices Virtual Channel Extension

Revision Summary

Date	Revision History	Revision Class	Comments
04/23/2010	0.1	Major	First Release.
06/04/2010	1.0	Major	Updated and revised the technical content.
07/16/2010	2.0	Major	Significantly changed the technical content.
08/27/2010	3.0	Major	Significantly changed the technical content.
10/08/2010	4.0	Major	Significantly changed the technical content.
11/19/2010	5.0	Major	Significantly changed the technical content.
01/07/2011	6.0	Major	Significantly changed the technical content.
02/11/2011	7.0	Major	Significantly changed the technical content.

Contents

1 Introduction

This document specifies the Remote Desktop Protocol: USB Devices Virtual Channel Extension to the Remote Desktop Protocol. This protocol is used to redirect USB devices from a **terminal client** to the **terminal server**. This allows the server access to devices that are 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]:

American National Standards Institute (ANSI) character set
device driver
globally unique identifier (GUID)
HRESULT
terminal server
Unicode string

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 Unicode 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 program that runs on the client machine.

URB: This stands for USB Request Packet, as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.

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-RDPEDYC] Microsoft Corporation, "Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension", June 2007.

[MS-RDPEXPS] Microsoft Corporation, "Remote Desktop Protocol: XML Paper Specification (XPS) Print Virtual Channel Extension", 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

[USB-SPC2.0] USB Consortium, "USB 2.0 Specification", April 2000, http://www.usb.org/developers/docs/

1.2.2 Informative References

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

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

1.3 Protocol Overview (Synopsis)

The Remote Desktop Protocol: USB Devices Virtual Channel Extension is used to transfer USB packets from a terminal server to a terminal client. The client forwards the USB packets to a physical device. Then the client returns the results after the physical device reassembles the packets.

Because this protocol can redirect a USB device, the implementer has to provide a way for the client to specify the USB devices that are redirected using this protocol, or the devices that will use an alternative method or the devices that are not redirected at all. When the device is redirected it cannot be used on the client. Examples:

- A USB mouse is attached to the client. If redirected using this protocol the mouse cannot be used on the client locally. However, if the client doesn't have a driver for the USB mouse, or if this is a second USB mouse, then this is an appropriate scenario to redirect a USB mouse using this protocol.
- Flash drive: alternative methods for redirecting the drive, such as the one described in [MS-RDPEFS], may or may not be more successful because that protocol is optimized for drives.

The examples can become complicated if composite devices are behind one USB device, because there are several different devices that can be used. As a result there isn't one definitive answer to what method can be used; as a result, this protocol is not trying to enforce any decision. The implementer of this protocol should consider enough provisions to give the user flexibility to choose whether or not to redirect a device, and should attempt to prevent the user from losing control of a USB device that the user doesn't want to be redirected. Examples of such provisions are: group policies, notifications, User Interface for selecting the right device, and so on.

The following diagram describes the event sequences in relation to the hardware USB device and the USB driver stack on the server.

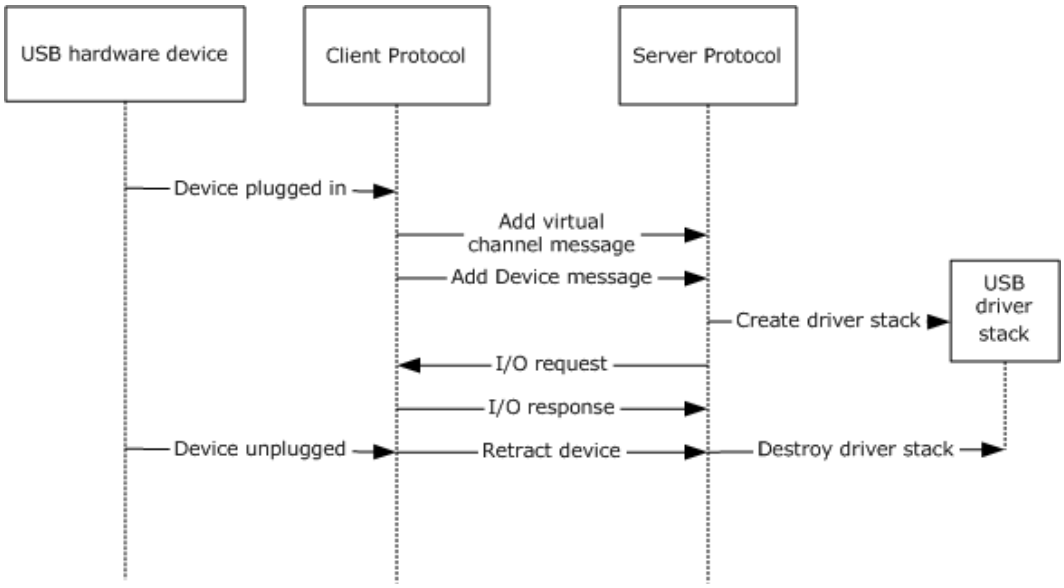


Figure 1: USB stack flow

When a USB device is plugged in, the client sends to the server the "Add Virtual Channel" and "Add Device" messages as described in section 1.3.1.2. The server in response creates a USB driver stack that will represent the device to the system. After that point, the server and the client are ready to exchange I/O packets as described in section 1.3.1.3.

When the device is unplugged from the client, it sends the "Retract Device" message to the server, which then destroys the driver stacks and stops any further I/O between the client and the server.

1.3.1 USB Devices Virtual Channel Protocol

The Remote Desktop Protocol: USB Devices Virtual Channel Extension is divided into the following logical sequences:

Channel setup sequence: A channel is opened, and capabilities are exchanged. The channel is assigned a specific identifier that is used by the client and the server to identify the USB device.

New device sequence: The client notifies the server about the arrival of a new device. The server creates a device on the server machine that corresponds to the device reported by the client.

I/O sequence: The server sends USB packets to the client and the client forwards the USB packets to the physical device and sends back the results after the physical device reassembles the packets.

1.3.1.1 Channel Setup Sequence

The Remote Desktop Protocol: USB Devices Virtual Channel Extension uses multiple channels within a single named dynamic virtual channel. There is one control channel and one channel for each of

the USB devices. The goal of this sequence is to set up the identifiers for the channel and to exchange the platform and version capabilities.

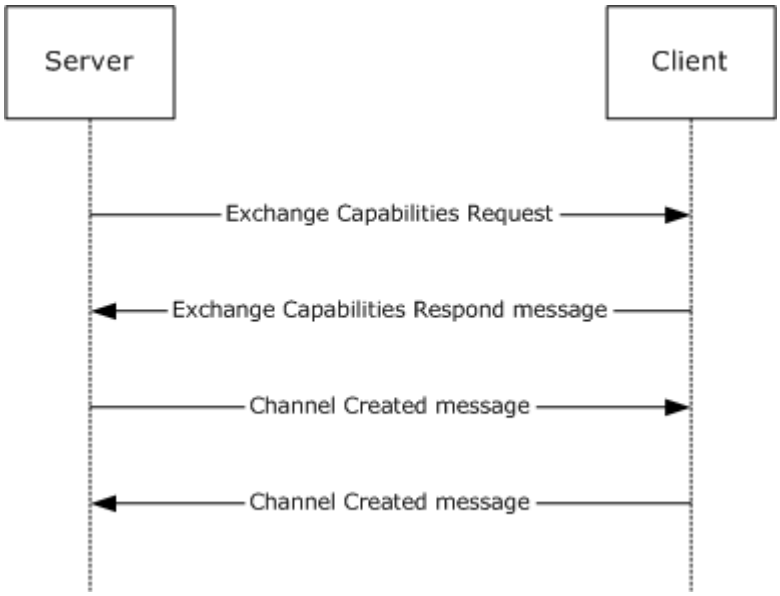


Figure 2: Channel setup sequence

1.3.1.2 New Device Sequence

The client uses the new device sequence to notify the server about a new device. It first notifies the server to create a new instance of the USB Redirection virtual channel. Once the new virtual channel is created, a new device message is sent to the server via the new virtual channel. The device is recognized based on the **HardwareIds** field of Add device message (section 2.2.4.2).

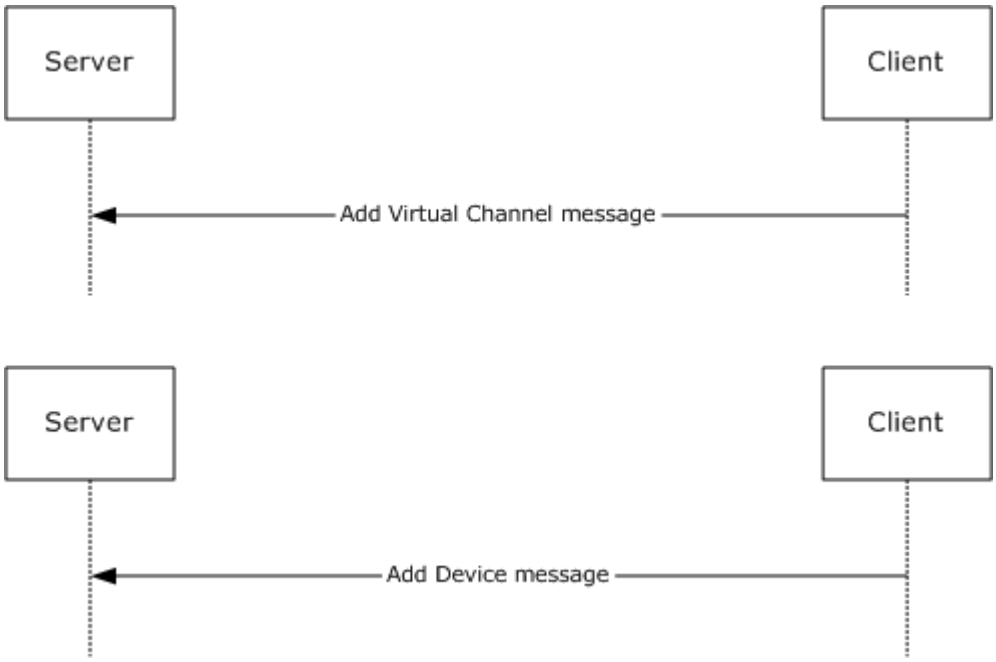


Figure 3: New device sequence

1.3.1.3 I/O Sequence

The server uses the I/O sequence to send I/O requests to the client. The server can send multiple I/O requests to the client without first waiting for the previously sent requests to be completed first.

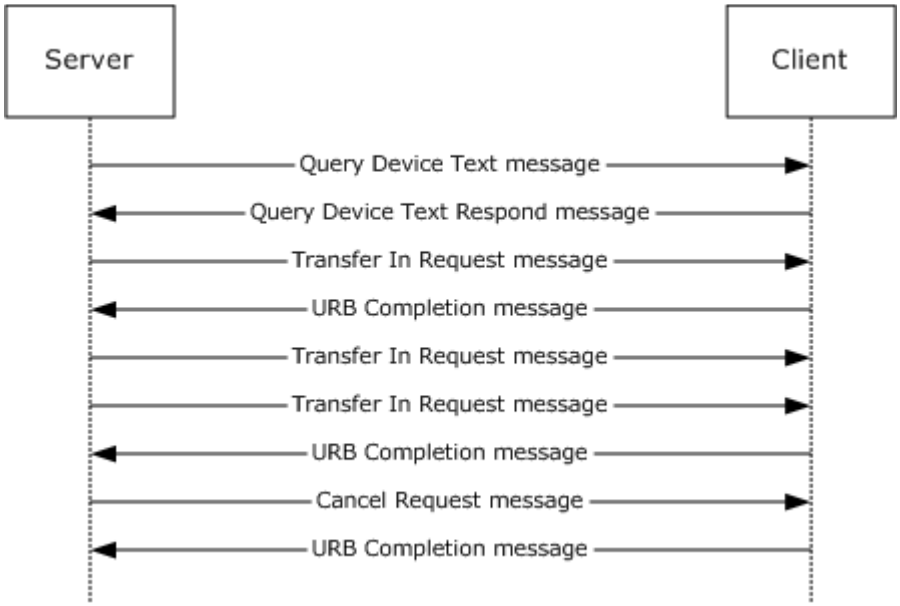


Figure 4: I/O sequence

1.4 Relationship to Other Protocols

The Remote Desktop Protocol: USB 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: USB 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: USB Devices Virtual Channel Extension is also terminated. The protocol is terminated by closing the underlying virtual channel. For details about closing the dynamic virtual channel, refer to [MS-RDPEDYC] section 3.2.5.2.

1.6 Applicability Statement

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

Device drivers and applications must meet the following requirements if they are to be redirected:

- This protocol is not intended for use with devices that require quality-of-service guarantees (because the I/O is sent over a network, there is no guarantee about the timeframe for delivering the I/O to and receiving it from the device).

- 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 the following operating system-specific I/O calls: Read, Write, and IOControl. Communication between the device driver and the application cannot be anything other than these basic calls. If there is any other I/O, the device cannot be redirected using this protocol hence the device will be treated as any other device attached to the client and this protocol will not be involved in any means.

1.7 Versioning and Capability Negotiation

This protocol supports versioning and capability negotiation at two levels. The first is supported through the use of interface manipulation messages, as specified in sections 2.2.2 and 2.2.3. The second is supported by the capability exchange messages, as specified in section 2.2.5.1.

The USB2.0 specification also includes versioning in the Device descriptor as described in section 9.6.1 of [USB-SPC2.0].

1.8 Vendors-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

None.

2 Messages

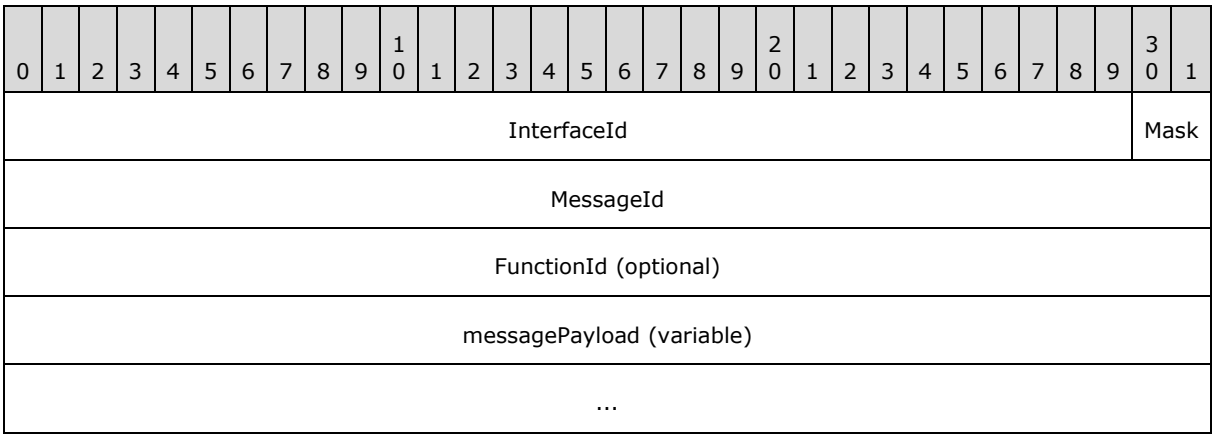
2.1 Transport

The Remote Desktop Protocol: USB Devices Virtual Channel Extension is designed to operate over dynamic virtual channels, as specified in [MS-RDPEDYC]. The dynamic virtual channel name is the **ANSI**-encoded null-terminated string "URBDRC". The usage of a channel name when opening a dynamic virtual channel is specified in [MS-RDPEDYC] section 2.2.2.1.

2.2 Message Syntax

2.2.1 Shared Message Header (SHARED_MSG_HEADER)

Every packet in this extension contains a common header.<1>



InterfaceId (30 bits): A 30-bit field that represents the common identifier for the interface. The default value is 0. If the message uses this default interface ID, the message is interpreted for the default interface for which this channel has been instantiated. All other values MUST be retrieved either from a Query Interface response (QI_RSP) ([MS-RDPEXPS] section 2.2.2.1.2) or from responses that contain interface IDs. The highest two bits of **NetInterfaceId** field in QI_RSP MUST be ignored.

This ID is valid until an Interface Release (IFACE_RELEASE) message ([MS-RDPEXPS] section 2.2.2.2) is sent or received with that ID. After an IFACE_RELEASE message is processed, this ID is considered invalid.

Mask (2 bits): The 2 bits of the **Mask** field MUST be set to one of the following values.

Value	Meaning
STREAM_ID_STUB 0x2	Indicates that the SHARED_MSG_HEADER is being used in a response message.
STREAM_ID_PROXY 0x1	Indicates that the SHARED_MSG_HEADER is not being used in a response message.
STREAM_ID_NONE 0x0	Indicates that the SHARED_MSG_HEADER is being used for interface manipulation capabilities exchange as specified in section 2.2.3. This value MUST NOT be used for any other messages.

MessageId (4 bytes): A 32-bit unsigned integer. A unique ID for the request or response pair. Requests and responses are matched based on this ID coupled with the **InterfaceId**.

FunctionId (4 bytes): A 32-bit unsigned integer. This field MUST be present in all packets except response packets. Its value is either used in interface manipulation messages or defined for a specific interface. The following values are categorized by the interface for which they are defined.

Common IDs for all interfaces are as follows.

Value	Meaning
RIMCALL_RELEASE 0x00000001	Release the given interface ID.
RIMCALL_QUERYINTERFACE 0x00000002	Query for a new interface.

Capabilities Negotiator Interface IDs are as follows.

Value	Meaning
RIM_EXCHANGE_CAPABILITY_REQUEST 0x00000100	The server sends the Interface Manipulation Exchange Capabilities Request message.

Client Request Completion Interface IDs are as follows.

Value	Meaning
IOCONTROL_COMPLETION 0x00000100	The client sends the IO Control Completion message.
URB_COMPLETION 0x00000101	The client sends the URB Completion message.
URB_COMPLETION_NO_DATA 0x00000102	The client sends the URB Completion No Data message.

Server USB Device Interface IDs are as follows.

Value	Meaning
CANCEL_REQUEST 0x00000100	The server sends the Cancel Request message.
REGISTER_REQUEST_CALLBACK 0x00000101	The server sends the Register Request Callback message.
IO_CONTROL 0x00000102	The server sends the IO Control message.
INTERNAL_IO_CONTROL 0x00000103	The server sends the Internal IO Control message.
QUERY_DEVICE_TEXT	The server sends the Query Device Text message.

Value	Meaning
0x00000104	
TRANSFER_IN_REQUEST 0x00000105	The server sends the Transfer In Request message.
TRANSFER_OUT_REQUEST 0x00000106	The server sends the Transfer Out Request message.
RETRACT_DEVICE 0x00000107	The server sends the Retract Device message.

Client Device Sink Interface IDs are as follows.

Value	Meaning
ADD_VIRTUAL_CHANNEL 0x00000100	The client sends the Add Virtual Channel message.
ADD_DEVICE 0x00000101	The client sends the Add Device message.

Channel Notification Interface IDs are as follows.

Value	Meaning
CHANNEL_CREATED 0x00000100	The server and the client send the Channel Created message.

messagePayload (variable): An array of unsigned 8-bit integers. The remainder of the message is interpreted based on the interface for which the packet is sent. This field is optional based on the packet length.

2.2.2 Interface Manipulation

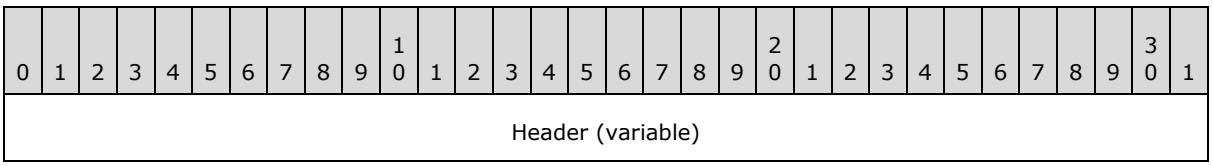
This protocol utilizes the same Interface Query and Interface Release messages that are defined in [MS-RDPEXPS] section 2.2.2.

2.2.3 Interface Manipulation Exchange Capabilities Interface

The Exchange Capabilities Interface is identified by the interface ID 0x00000000. This interface is used to exchange the client's and the server's capabilities for interface manipulation.

2.2.3.1 Interface Manipulation Exchange Capabilities Request (RIM_EXCHANGE_CAPABILITY_REQUEST)

This message is used by the server to request interface manipulation capabilities from the client.



...
CapabilityValue

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST be set to 0x00000000. The **Mask** field MUST be set to STREAM_ID_NONE. The **FunctionId** field MUST be set to RIM_EXCHANGE_CAPABILITY_REQUEST (0x00000100).

CapabilityValue (4 bytes): A 32-bit unsigned integer that identifies the server's capability. The valid values for this field are as follows.

Value	Meaning
RIM_CAPABILITY_VERSION_01 0x00000001	This capability MUST be present in the message.

2.2.3.2 Interface Manipulation Exchange Capabilities Response (RIM_EXCHANGE_CAPABILITY_RESPONSE)

This message is sent by the client in response to RIM_EXCHANGE_CAPABILITY_REQUEST.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
CapabilityValue																															
Result																															

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field and the **MessageId** field in this message header SHOULD contain the same values as the **InterfaceId** and **MessageId** fields in the corresponding RIM_EXCHANGE_CAPABILITY_REQUEST message. The **Mask** field MUST be set to STREAM_ID_NONE.

CapabilityValue (4 bytes): A 32-bit unsigned integer that identifies the client's capability. The valid values for this field are as follows.

Value	Meaning
RIM_CAPABILITY_VERSION_01 0x00000001	This capability MUST be present in the message.

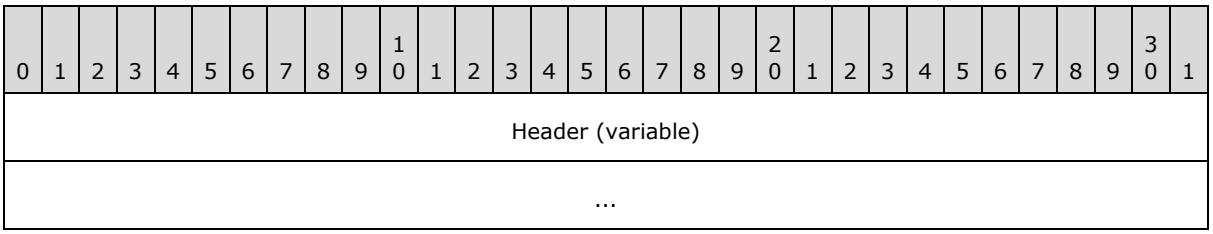
Result (4 bytes): A 32-bit unsigned integer that indicates the HRESULT of the operation.

2.2.4 Device Sink Interface

The device sink interface is identified by the interface ID 0x00000001. The device sink interface is used by the client to communicate with the server about new USB devices.

2.2.4.1 Add Virtual Channel Message (ADD_VIRTUAL_CHANNEL)

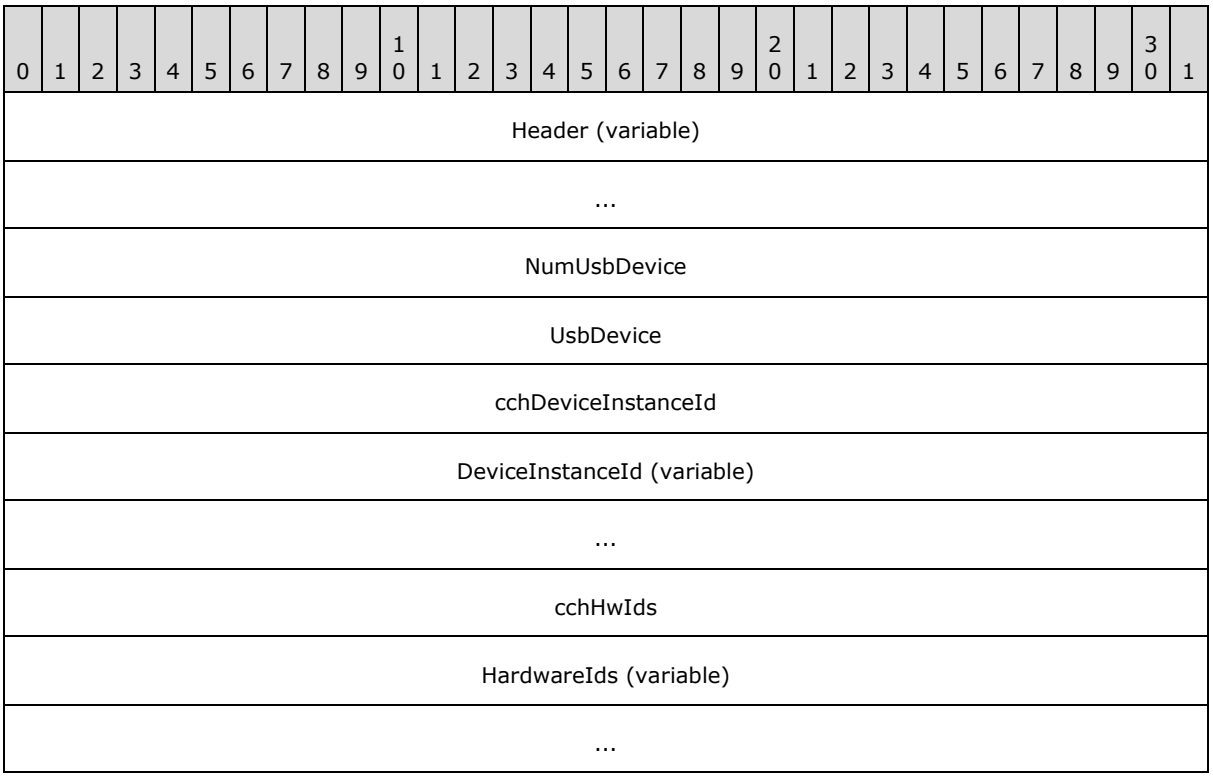
The ADD_VIRTUAL_CHANNEL message is sent from the client to the server to create a new instance of dynamic virtual channel.



Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST be set to 0x00000001. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to ADD_VIRTUAL_CHANNEL (0x00000100).

2.2.4.2 Add Device Message (ADD_DEVICE)

The ADD_DEVICE message is sent from the client to the server in order to create a redirected USB device on the server.



cchCompatIds
CompatibilityIds (variable)
...
cchContainerId
ContainerId (variable)
...
UsbDeviceCapabilities
...
...
...
...
...
...

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST be set to 0x00000001. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to ADD_DEVICE (0x00000101).

NumUsbDevice (4 bytes): A 32-bit unsigned integer. MUST be set to 0x00000001.

UsbDevice (4 bytes): A 32-bit unsigned integer. A unique interface ID to be used by request messages defined in USB device interface.

cchDeviceInstanceId (4 bytes): A 32-bit unsigned integer. This field MUST contain the number of Unicode characters in the **DeviceInstanceId** field.

DeviceInstanceId (variable): An array of bytes. A variable-length field that contains a null-terminated **Unicode string** that identifies an instance of a USB device.

cchHwIds (4 bytes): A 32-bit unsigned integer. This field MUST contain the number of Unicode characters in the **HardwareIds** field. This field MAY be 0x00000000.

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

cchCompatIds (4 bytes): A 32-bit unsigned integer. This field MUST contain the number of Unicode characters in the **CompatibilityIds** field.

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

cchContainerId (4 bytes): A 32-bit unsigned integer. This field MUST contain the number of Unicode characters in the **ContainerId** field.

ContainerId (variable): An array of bytes. A variable-length field that contains a null-terminated Unicode string that contains the container ID in **GUID**, as specified in [MS-DTYP] section 2.3.2.2, format of the USB device.

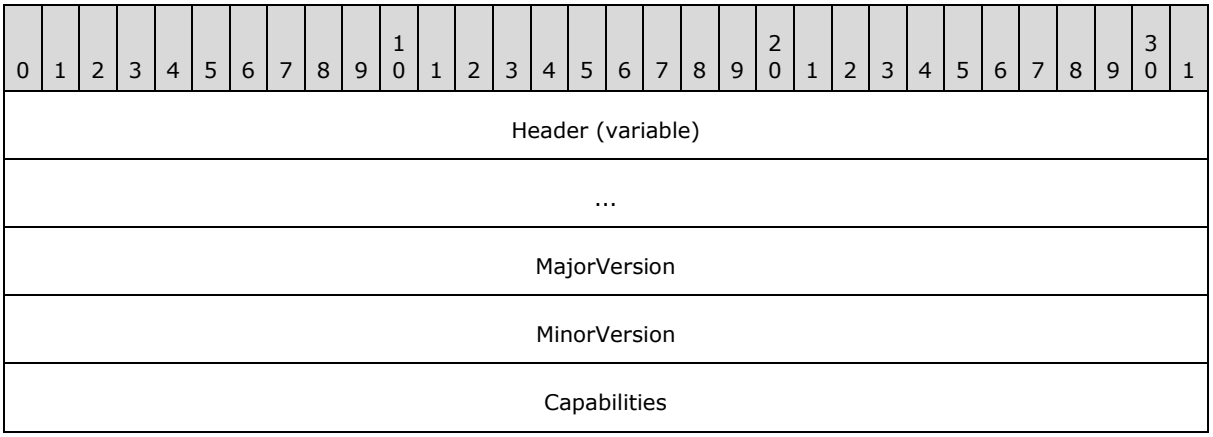
UsbDeviceCapabilities (28 bytes): A 28-byte structure as specified in section 2.2.11.

2.2.5 Channel Notification Interface

The channel notification interface is used by both the client and the server to communicate with the other side.

2.2.5.1 Channel Created Message (CHANNEL_CREATED)

The CHANNEL_CREATED message is sent from both the client and the server to inform the other side of the RDP USB device redirection version supported.



Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST be set to 0x00000002 if sent by the server and it MUST be set to 0x000000003 if sent by the client. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to CHANNEL_CREATED (0x00000100).

MajorVersion (4 bytes): A 32-bit unsigned integer. The major version of RDP USB redirection supported. This value MUST be set to one.

MinorVersion (4 bytes): A 32-bit unsigned integer. The minor version of RDP USB redirection supported. This value MUST be set to zero.

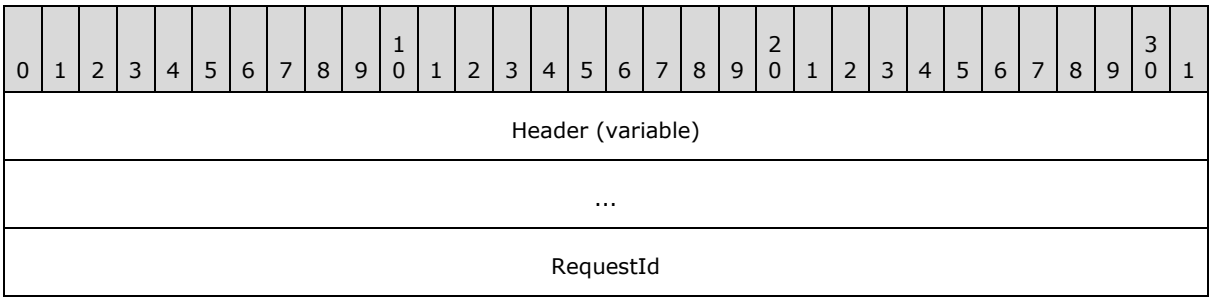
Capabilities (4 bytes): A 32-bit unsigned integer. The capabilities of RDP USB redirection supported. This value MUST be set to zero.

2.2.6 USB Device Interface

The USB **device interface** is used by the server to send IO-related requests to the client.

2.2.6.1 Cancel Request Message (CANCEL_REQUEST)

The CANCEL_REQUEST message is sent from the server to the client to cancel an outstanding IO request.

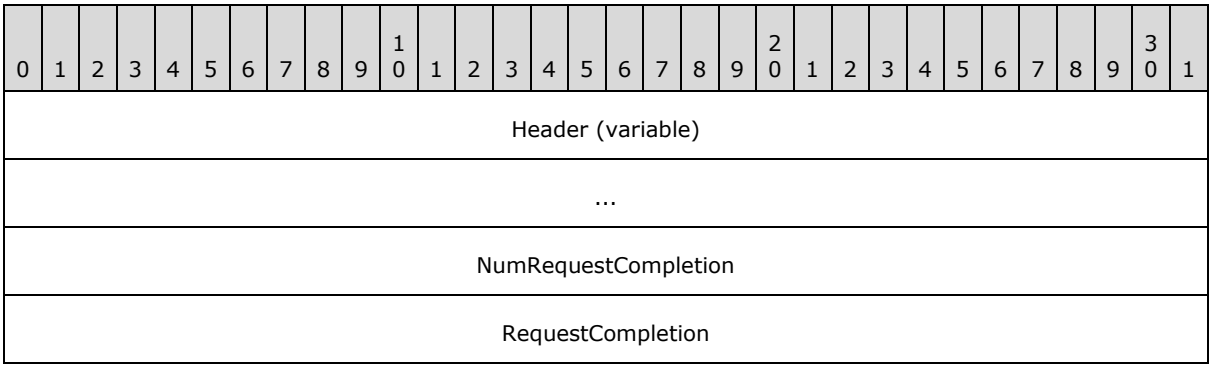


Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to CANCEL_REQUEST (0x00000100).

RequestId (4 bytes): A 32-bit unsigned integer. This value represents the ID of a request previously sent via IO_CONTROL, INTERNAL_IO_CONTROL, TRANSFER_IN_REQUEST, or TRANSFER_OUT_REQUEST message.

2.2.6.2 Register Request Callback Message (REGISTER_REQUEST_CALLBACK)

The REGISTER_REQUEST_CALLBACK message is sent from the server to the client in order to provide a Request Completion Interface to the client.



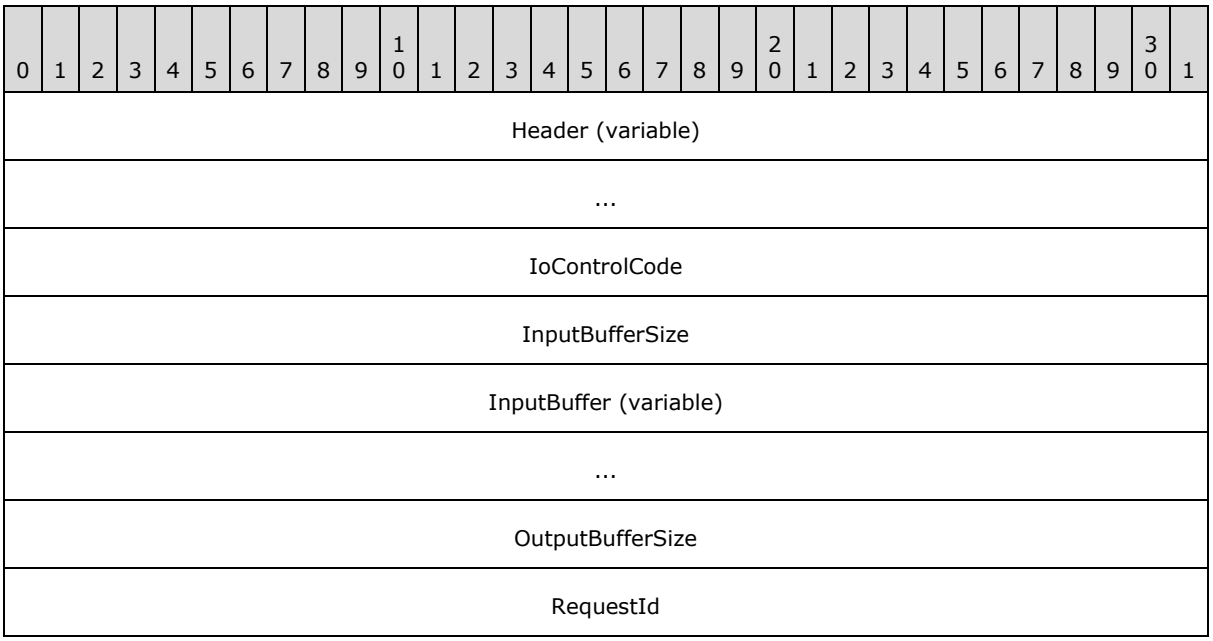
Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to REGISTER_REQUEST_CALLBACK (0x00000101).

NumRequestCompletion (4 bytes): A 32-bit unsigned integer. This field MUST be set to 0x00000001.

RequestCompletion (4 bytes): A 32-bit unsigned integer. A unique **InterfaceID** to be used by all Request Completion messages defined in the Request Completion Interface (section 2.2.7).

2.2.6.3 IO Control Message (IO_CONTROL)

The IO_CONTROL message is sent from the server to the client in order to submit an IO control request to the USB device.



Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to IO_CONTROL (0x00000102).

IoControlCode (4 bytes): A 32-bit unsigned integer. An IO control code as specified in section 2.2.12.

InputBufferSize (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **InputBuffer** field.

InputBuffer (variable): A byte array. This value represents the input buffer for the IO control request.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The maximum number of bytes the client can return to the server.

RequestId (4 bytes): A 32-bit unsigned integer. This ID uniquely identifies the I/O control request.

2.2.6.4 Internal IO Control Message (INTERNAL_IO_CONTROL)

The INTERNAL_IO_CONTROL message is sent from the server to the client in order to submit an internal IO control request to the USB device.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
IoControlCode																															
InputBufferSize																															
InputBuffer (variable)																															
...																															
OutputBufferSize																															
RequestId																															

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to INTERNAL_IO_CONTROL (0x00000103).

IoControlCode (4 bytes): A 32-bit unsigned integer. An internal IO control code as specified in section 2.2.13.

InputBufferSize (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **InputBuffer** field.

InputBuffer (variable): A byte array. This value represents the input buffer for the internal IO control request.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The maximum number of bytes the internal IO control request can return.

RequestId (4 bytes): A 32-bit unsigned integer. This value represents an ID that uniquely identifies this internal IO control request.

2.2.6.5 Query Device Text Message (QUERY_DEVICE_TEXT)

The QUERY_DEVICE_TEXT message is sent from the server to the client in order to query the USB device's text when the server receives a query device test request (IRP_MN_QUERY_DEVICE_TEXT) from the system as described in [MSFT-W2KDDK], Volume 1, Part 1, Chapter 2.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															

...
TextType
LocaleId

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to QUERY_DEVICE_TEXT (0x00000104).

TextType (4 bytes): A 32-bit unsigned integer. This value represents the type of text to query as described in [MSFT-W2KDDK], Volume 1, Part 1, Chapter 2.

LocaleId (4 bytes): A 32-bit unsigned integer. This value represents the locale of the text to query as described in [MSFT-W2KDDK], Volume 1, Part 1, Chapter 2.

2.2.6.6 Query Device Text Response Message (QUERY_DEVICE_TEXT_RSP)

The QUERY_DEVICE_TEXT_RSP message is sent from the client in response to a QUERY_DEVICE_TEXT message sent by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
cchDeviceDescription																															
DeviceDescription (variable)																															
...																															

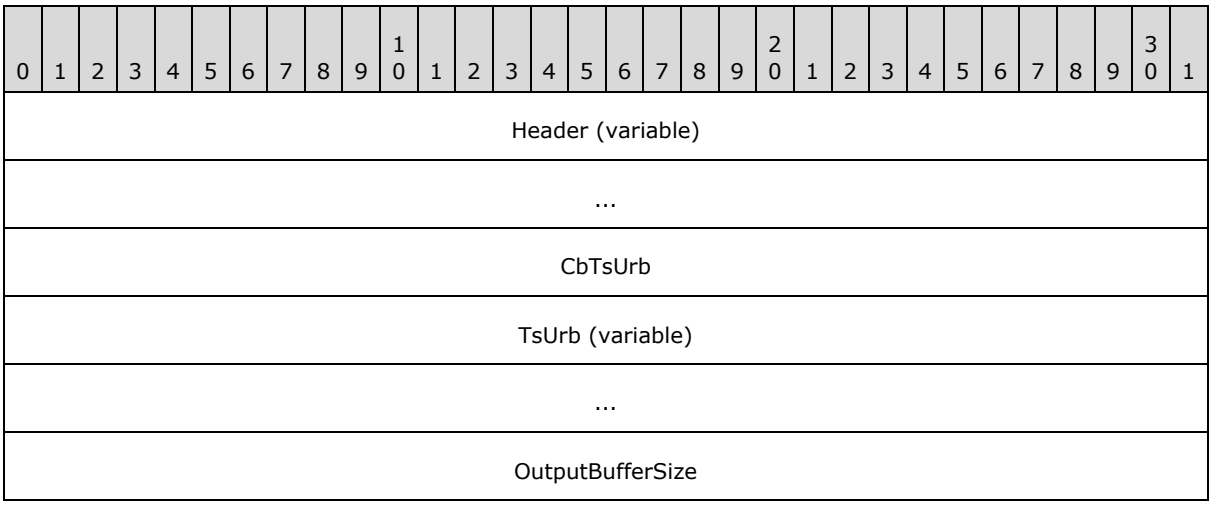
Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** and **MessageId** fields in this header MUST contain the same values as the **InterfaceId** and **MessageId** fields in the corresponding QUERY_DEVICE_TEXT. The **Mask** field MUST be set to STREAM_ID_STUB.

cchDeviceDescription (4 bytes): A 32-bit unsigned integer. This field MUST contain the number of Unicode characters in the **DeviceDescription** field.

DeviceDescription (variable): An array of bytes. A variable-length field that contains a null-terminated Unicode string that contains the requested device text.

2.2.6.7 Transfer In Request (TRANSFER_IN_REQUEST)

The TRANSFER_IN_REQUEST message is sent from the server to the client in order to request data from the USB device.



Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to TRANSFER_IN_REQUEST (0x00000105).

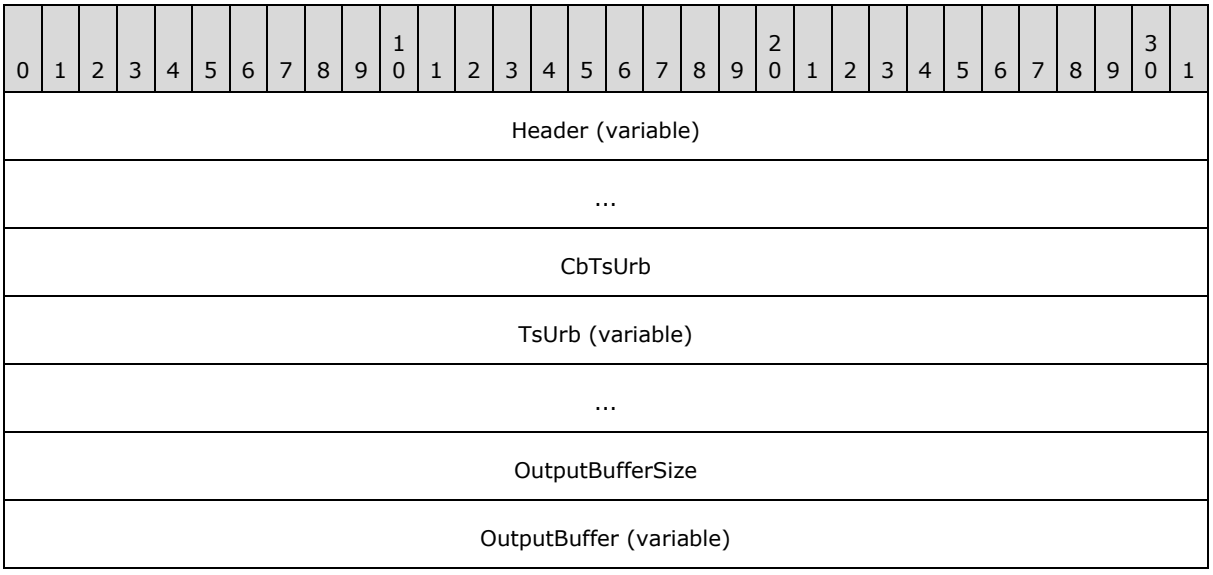
CbTsUrb (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **TsUrb** field.

TsUrb (variable): A TS_URB structure as defined in section 2.2.9.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. This value represents the maximum number of bytes of data that is requested from the USB device.

2.2.6.8 Transfer Out Request (TRANSFER_OUT_REQUEST)

The TRANSFER_OUT_REQUEST message is sent from the server to the client in order to submit data to the USB device.



...

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to TRANSFER_OUT_REQUEST (0x00000106).

CbTsUrb (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **TsUrb** field.

TsUrb (variable): A TS_URB structure as defined in section 2.2.9.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The size in bytes of the **OutputBuffer** field.

OutputBuffer (variable): An array of bytes. The raw data to be sent to the device.

2.2.6.9 Retract Device (RETRACT_DEVICE)

The RETRACT_DEVICE message is sent from the server to the client in order to stop redirecting the USB device.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
Reason																															

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **UsbDevice** field of the ADD_DEVICE message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to RETRACT_DEVICE (0x00000107).

Reason (4 bytes): A 32-bit unsigned integer. The reason code, as specified in section 2.2.8, to stop redirecting the USB device.

2.2.7 Request Completion Interface

The Request Completion Interface is used by the client to send the final result for a request previously sent from the server.

2.2.7.1 IO Control Completion (IOCONTROL_COMPLETION)

The IOCONTROL_COMPLETION request is sent from the client to the server as the final result of an IO Control request or internal IO Control request.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
RequestId																															
HResult																															
Information																															
OutputBufferSize																															
OutputBuffer (variable)																															
...																															

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **RequestCompletion** field of the REGISTER_REQUEST_CALLBACK message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to IOCONTROL_COMPLETION (0x00000100).

RequestId (4 bytes): A 32-bit unsigned integer. This field MUST match the value sent previously in the **RequestId** field of the IO_CONTROL message.

HResult (4 bytes): A 32-bit unsigned integer that indicates the HRESULT of the operation.

Information (4 bytes): A 32-bit unsigned integer. The number of bytes of data to be transferred by the request.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **OutputBuffer** field.

OutputBuffer (variable): A data buffer that results from processing the request.

2.2.7.2 URB Completion (URB_COMPLETION)

The URB_COMPLETION request is sent from the client to the server as the final result of a TRANSFER_IN_REQUEST that contains output data.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															

RequestId
CbTsUrbResult
TsUrbResult (variable)
...
HResult
OutputBufferSize
OutputBuffer (variable)
...

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **RequestCompletion** field of the REGISTER_REQUEST_CALLBACK message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to URB_COMPLETION (0x00000101).

RequestId (4 bytes): A 32-bit unsigned integer. This field MUST match the value sent previously in the **RequestId** field of TsUrb structure in the TRANSFER_IN_REQUEST message.

CbTsUrbResult (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **TsUrbResult** field.

TsUrbResult (variable): A TS_URB_RESULT structure as defined in 2.2.10.

HResult (4 bytes): A 32-bit unsigned integer that indicates the HRESULT of the operation.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **OutputBuffer** field.

OutputBuffer (variable): A data buffer that results from processing the request.

2.2.7.3 URB Completion No Data (URB_COMPLETION_NO_DATA)

The URB_COMPLETION_NO_DATA request is sent from the client to the server as the final result of a TRANSFER_IN_REQUEST that contains no output data or a TRANSFER_OUT_REQUEST.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Header (variable)																															
...																															
RequestId																															

CbTsUrbResult
TsUrbResult (variable)
...
HResult
OutputBufferSize

Header (variable): The SHARED_MSG_HEADER (as specified in section 2.2.1). The **InterfaceId** field MUST match the value sent previously in the **RequestCompletion** field of the REGISTER_REQUEST_CALLBACK message. The **Mask** field MUST be set to STREAM_ID_PROXY. The **FunctionId** field MUST be set to URB_COMPLETION_NO_DATA (0x00000102).

RequestId (4 bytes): A 32-bit unsigned integer. This field MUST match the value sent previously in the **RequestId** field of TsUrb structure in the TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST message.

CbTsUrbResult (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **TsUrbResult** field.

TsUrbResult (variable): A TS_URB_RESULT structure as defined in 2.2.10.

HResult (4 bytes): A 32-bit unsigned integer that indicates the HRESULT of the operation.

OutputBufferSize (4 bytes): A 32-bit unsigned integer. The size, in bytes, of data sent to the device of the **RequestId** that corresponds to a TRANSFER_OUT_REQUEST. This field MUST be zero if the **RequestId** corresponds to a TRANSFER_IN_REQUEST.

2.2.8 USB_RETRACT_REASON Constants

The reason why the server requests the client to stop redirecting a USB device.

Symbolic name/value	Description
UsbRetractReason_BlockedByPolicy 0x00000001	The USB device is to be stopped from being redirected because the device is blocked by the server's policy.

2.2.9 TS_URB Structures

The TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST is sent in response to a URB request received from the system.

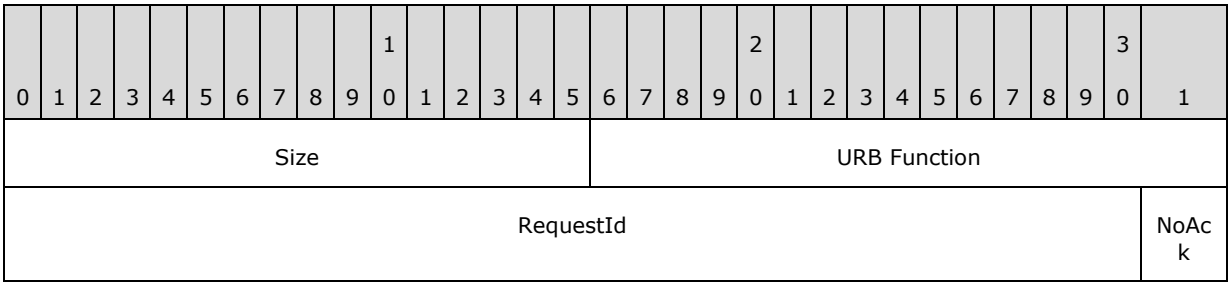
For information on **URB** definitions, see [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.

2.2.9.1 Common Structures

This section specifies common structures that are used by more than one TS_URB structure.

2.2.9.1.1 TS_URB_HEADER

Every TS_URB structure begins with a common header called TS_URB_HEADER.



Size (2 bytes): A 16-bit unsigned integer. The size in bytes of the TS_URB structure.

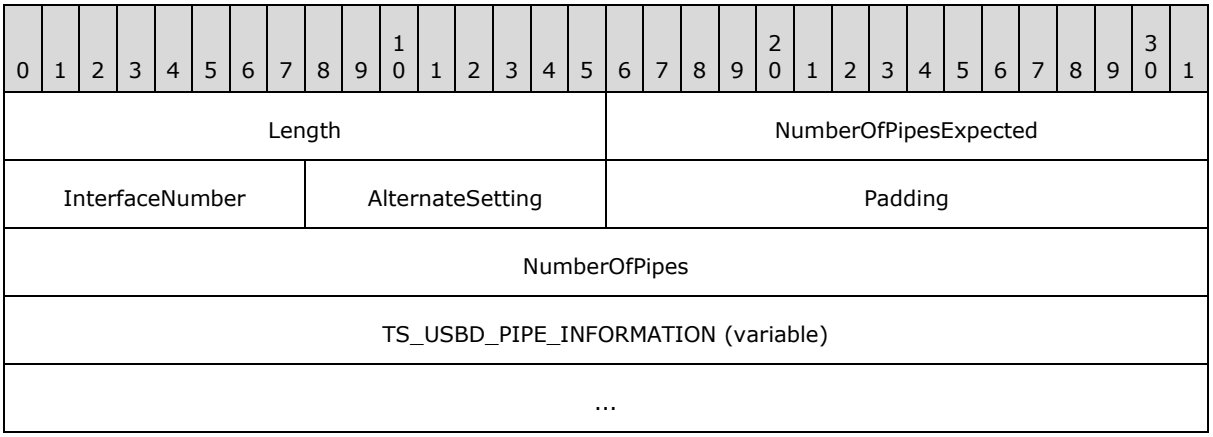
URB Function (2 bytes): A 16-bit unsigned integer. The URB function as specified in in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3. The URB structure specified by the URB function is represented by appropriate TS_URB structure as it is described in this protocol.

RequestId (31 bits): A 31-bit field. An ID that uniquely identifies the TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST.

NoAck (1 bit): A 1-bit field. If this bit is nonzero the client should not send a URB_COMPLETION message for this TRANSFER_OUT_REQUEST. This bit can be nonzero only if the **NoAckIsochWriteJitterBufferSizeInMs** field in USB_DEVICE_CAPABILITIES is nonzero and URB Function is set to URB_FUNCTION_ISOCH_TRANSFER. If the **RequestId** field is set to TRANSFER_IN_REQUEST, this field MUST be set to zero.

2.2.9.1.2 TS_USBD_INTERFACE_INFORMATION

The TS_USBD_INTERFACE_INFORMATION is based on the USBD_INTERFACE_INFORMATION structure as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.



Length (2 bytes): A 16-bit unsigned integer. The size in bytes of the TS_USBD_INTERFACE_INFORMATION structure.

NumberOfPipesExpected (2 bytes): A 16-bit unsigned integer. The number of USBD_PIPE_INFORMATION structures found in the USBD_INTERFACE_INFORMATION.

InterfaceNumber (1 byte): A 8-bit unsigned integer. This value is from the **InterfaceNumber** field in USBD_INTERFACE_INFORMATION.

AlternateSetting (1 byte): A 8-bit unsigned integer. This value is from the **AlternateSetting** field in USBD_INTERFACE_INFORMATION.

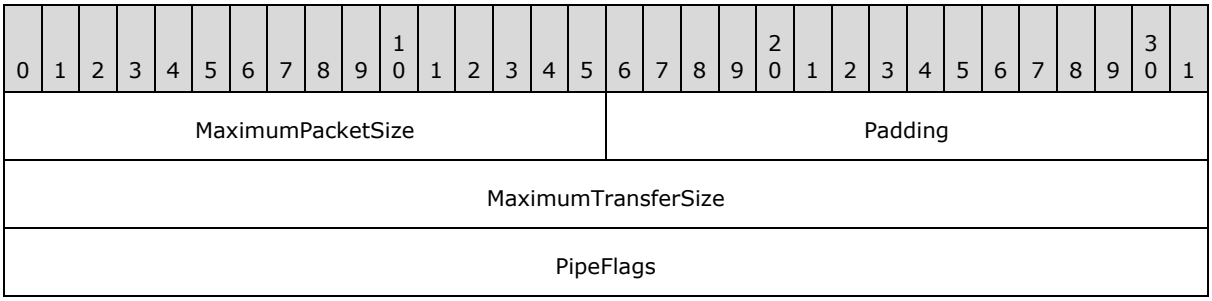
Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

NumberOfPipes (4 bytes): A 32-bit unsigned integer. This value is from the **NumberOfPipes** field in USBD_INTERFACE_INFORMATION.

TS_USBD_PIPE_INFORMATION (variable): An array of TS_USBD_PIPE_INFORMATION structures. The number of array elements is determined by the **NumberOfPipes** field.

2.2.9.1.3 TS_USBD_PIPE_INFORMATION

The TS_USBD_PIPE_INFORMATION is based on the USBD_PIPE_INFORMATION structure as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.<2>



MaximumPacketSize (2 bytes): A 16-bit unsigned integer. This value is from the **MaximumPacketSize** field in USBD_PIPE_INFORMATION.

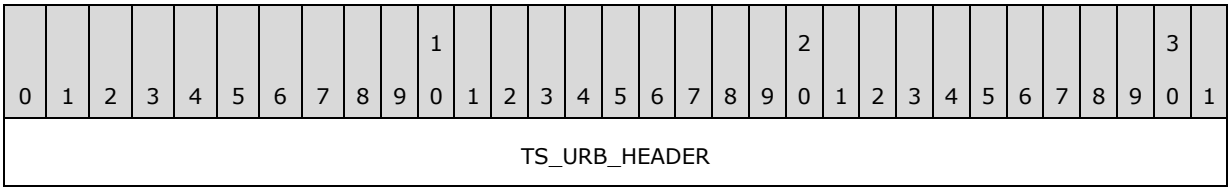
Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

MaximumTransferSize (4 bytes): A 32-bit unsigned integer. This value is from the **MaximumTransferSize** field in USBD_PIPE_INFORMATION.

PipeFlags (4 bytes): A 32-bit unsigned integer. This value is from the **PipeFlags** field in USBD_PIPE_INFORMATION.

2.2.9.2 TS_URB_SELECT_CONFIGURATION

This packet represents the URB structure URB_SELECT_CONFIGURATION, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using TRANSFER_IN_REQUEST. **OutputBufferSize** MUST be set to zero.



...	
ConfigurationDescriptorIsValid	Padding
NumInterfaces	
TS_USBD_INTERFACE_INFORMATION (variable)	
...	
USB_CONFIGURATION_DESCRIPTOR (variable)	
...	

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

ConfigurationDescriptorIsValid (1 byte): A 8-bit unsigned integer. A non-zero value indicates that the TS_URB_SELECT_CONFIGURATION contains the USB_CONFIGURATION_DESCRIPTOR field.

Padding (3 bytes): A 24-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

NumInterfaces (4 bytes): A 32-bit unsigned integer. The number of TS_USBD_INTERFACE_INFORMATION structures that are in the TS_URB_SELECT_CONFIGURATION.

TS_USBD_INTERFACE_INFORMATION (variable): A TS_USBD_INTERFACE_INFORMATION structure as specified in section 2.2.9.1.2.

USB_CONFIGURATION_DESCRIPTOR (variable): A USB_CONFIGURATION_DESCRIPTOR as described in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3.

2.2.9.3 TS_URB_SELECT_INTERFACE

This packet represents the URB structure URB_SELECT_INTERFACE, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with **OutputBufferSize** set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															
ConfigurationHandle																															
TS_USBD_INTERFACE_INFORMATION (variable)																															

...

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

ConfigurationHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_CONFIGURATION request.

TS_USBD_INTERFACE_INFORMATION (variable): A TS_USBD_INTERFACE_INFORMATION structure as specified in section 2.2.9.1.2.

2.2.9.4 TS_URB_PIPE_REQUEST

This packet represents the URB structure URB_PIPE_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with **OutputBufferSize** set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															
PipeHandle																															

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

PipeHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_INTERFACE request.

2.2.9.5 TS_URB_GET_CURRENT_FRAME_NUMBER

This packet represents the URB structure URB_GET_CURRENT_FRAME_NUMBER, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST. **OutputBufferSize** MUST be set to the size of the **FrameNumber** field in URB_GET_CURRENT_FRAME_NUMBER.

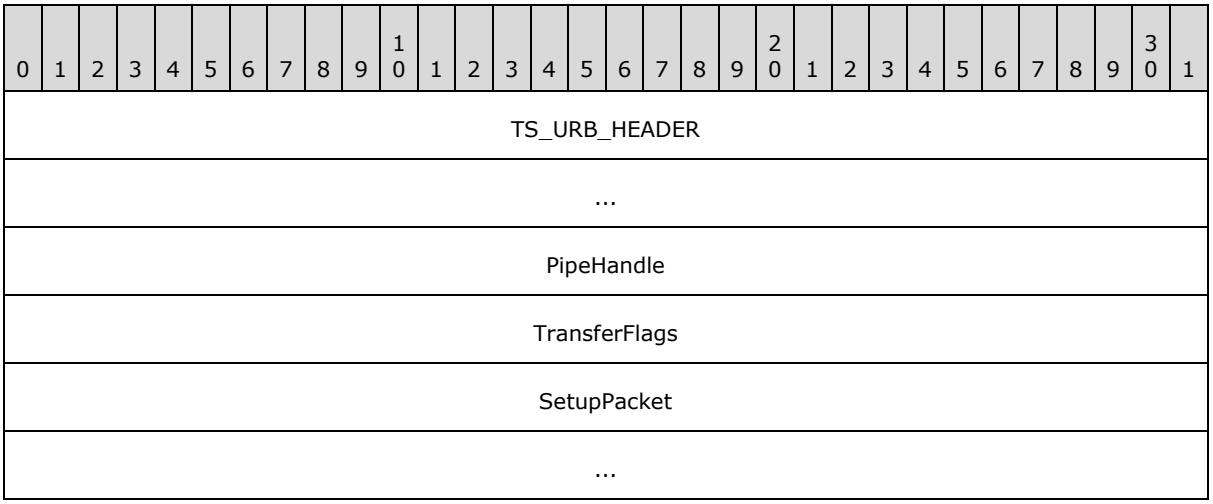
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

2.2.9.6 TS_URB_CONTROL_TRANSFER

This packet represents the URB structure URB_CONTROL_TRANSFER, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. If the **TransferFlags** field in URB_CONTROL_TRANSFER contains the USBD_TRANSFER_DIRECTION_IN flag, the packet is sent using the

TRANSFER_IN_REQUEST message with **OutputBufferSize** set to **TransferBufferLength** as defined in URB_CONTROL_TRANSFER; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with **InputBufferSize** set to **TransferBufferLength** and **InputBuffer** set to data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_CONTROL_TRANSFER.



TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

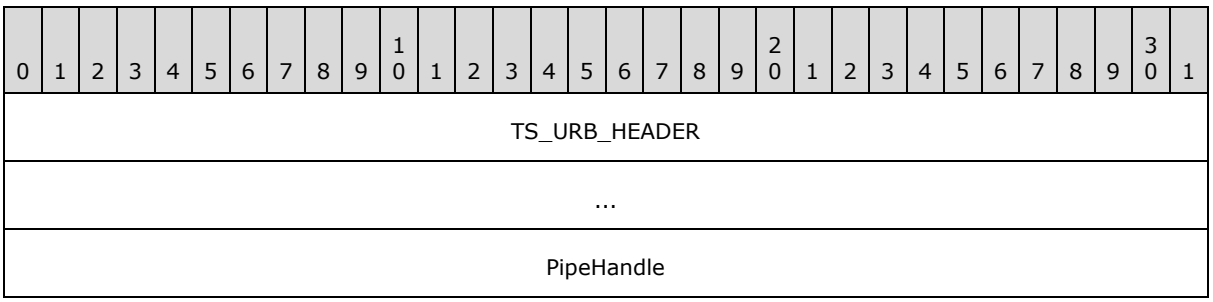
PipeHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_INTERFACE request.

TransferFlags (4 bytes): A 32-bit unsigned integer. This value is from the **TransferFlags** field in URB_CONTROL_TRANSFER.

SetupPacket (8 bytes): An 8-byte array. This value is from the **SetupPacket** field in URB_CONTROL_TRANSFER.

2.2.9.7 TS_URB_BULK_OR_INTERRUPT_TRANSFER

The packet represents the URB structure URB_BULK_OR_INTERRUPT_TRANSFER, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. If the **TransferFlags** field in URB_BULK_OR_INTERRUPT_TRANSFER contains the USBD_TRANSFER_DIRECTION_IN flag, the packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_BULK_OR_INTERRUPT_TRANSFER; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with the **InputBufferSize** field set to **TransferBufferLength** and the **InputBuffer** field set to the data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_BULK_OR_INTERRUPT_TRANSFER.



TransferFlags

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

PipeHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_INTERFACE request.

TransferFlags (4 bytes): A 32-bit unsigned integer. This value is from the **TransferFlags** field in URB_BULK_OR_INTERRUPT_TRANSFER.

2.2.9.8 TS_URB_ISOCH_TRANSFER

This packet represents the URB structure URB_ISOCH_TRANSFER, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. If the **TransferFlags** field in URB_ISOCH_TRANSFER contains the USBD_TRANSFER_DIRECTION_IN flag, the packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_ISOCH_TRANSFER; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with the **InputBufferSize** field set to **TransferBufferLength** and the **InputBuffer** field set to the data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_ISOCH_TRANSFER.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															
PipeHandle																															
TransferFlags																															
StartFrame																															
NumberOfPackets																															
ErrorCount																															
IsoPacket (variable)																															
...																															

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

PipeHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_INTERFACE request.

TransferFlags (4 bytes): A 32-bit unsigned integer. This value is from the **TransferFlags** field in URB_ISOCH_TRANSFER.

StartFrame (4 bytes): A 32-bit unsigned integer. This value is from the **StartFrame** field in URB_ISOCH_TRANSFER.

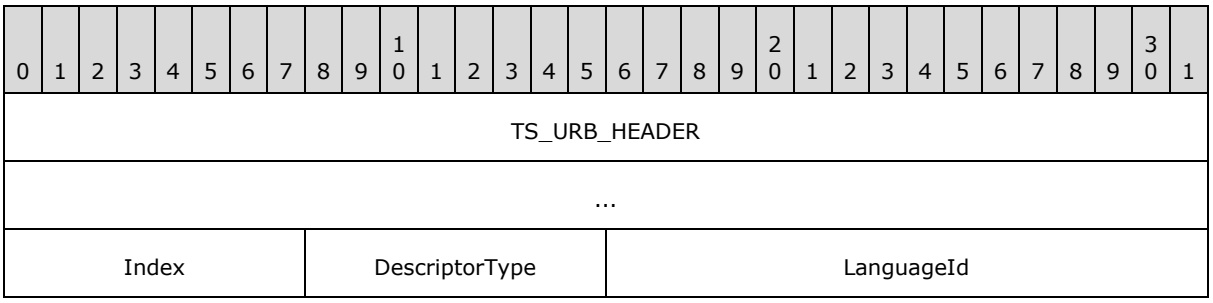
NumberOfPackets (4 bytes): A 32-bit unsigned integer. This value is from the **NumberOfPackets** field in URB_ISOCH_TRANSFER.

ErrorCount (4 bytes): A 32-bit unsigned integer. This value is from the **ErrorCount** field in URB_ISOCH_TRANSFER.

IsoPacket (variable): An array of USBD_ISO_PACKET_DESCRIPTOR structures. This value is from the **IsoPacket** field in URB_ISOCH_TRANSFER.

2.2.9.9 TS_URB_CONTROL_DESCRIPTOR_REQUEST

This packet represents the URB structure URB_CONTROL_DESCRIPTOR_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. If the URB Function in URB_CONTROL_DESCRIPTOR_REQUEST is URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE, URB_FUNCTION_GET_DESCRIPTOR_FROM_ENDPOINT, or URB_FUNCTION_GET_DESCRIPTOR_FROM_INTERFACE, the packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_DESCRIPTOR_REQUEST; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with the **InputBufferSize** field set to **TransferBufferLength** and the **InputBuffer** field set to the data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_CONTROL_DESCRIPTOR_REQUEST.



TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

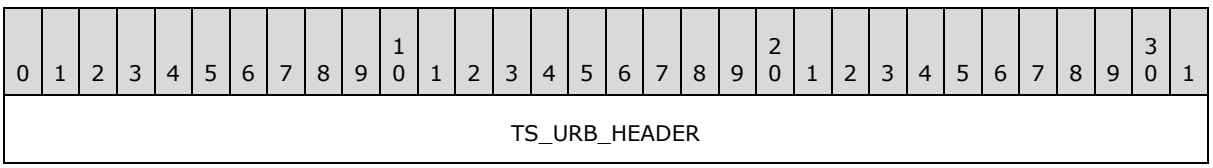
Index (1 byte): A 8-bit unsigned integer. This value is from the **Index** field in URB_CONTROL_DESCRIPTOR_REQUEST.

DescriptorType (1 byte): A 8-bit unsigned integer. This value is from the **DescriptorType** field in URB_CONTROL_DESCRIPTOR_REQUEST.

LanguageId (2 bytes): A 16-bit unsigned integer. This value is from the **LanguageId** field in URB_CONTROL_DESCRIPTOR_REQUEST.

2.2.9.10 TS_URB_CONTROL_FEATURE_REQUEST

This packet represents the URB structure URB_CONTROL_FEATURE_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to zero.



...	
FeatureSelector	Index

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

FeatureSelector (2 bytes): A 16-bit unsigned integer. This value is from the **Index** field in URB_CONTROL_FEATURE_REQUEST.

Index (2 bytes): A 16-bit unsigned integer. This value is from the **Index** field in URB_CONTROL_FEATURE_REQUEST.

2.2.9.11 TS_URB_CONTROL_GET_STATUS_REQUEST

This packet represents the URB structure URB_CONTROL_GET_STATUS_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_GET_STATUS_REQUEST.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															
Index																Padding															

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

Index (2 bytes): A 16-bit unsigned integer. This value is from the **Index** field in URB_CONTROL_GET_STATUS_REQUEST.

Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

2.2.9.12 TS_URB_CONTROL_VENDOR_OR_CLASS_REQUEST

This packet represents the URB structure URB_CONTROL_VENDOR_OR_CLASS_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. If the **TransferFlags** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST contains the USBD_TRANSFER_DIRECTION_IN flag, the packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_VENDOR_OR_CLASS_REQUEST; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with the **InputBufferSize** field set to **TransferBufferLength** and the **InputBuffer** field set to the data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															

...		
TransferFlags		
RequestTypeReservedBits	Request	Value
Index		Padding

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

TransferFlags (4 bytes): A 32-bit unsigned integer. This value is from the **TransferFlags** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

RequestTypeReservedBits (1 byte): A 8-bit unsigned integer. This value is from the **RequestTypeReservedBits** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

Request (1 byte): A 8-bit unsigned integer. This value is from the **Request** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

Value (2 bytes): A 16-bit unsigned integer. This value is from the **Value** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

Index (2 bytes): A 16-bit unsigned integer. This value is from the **Index** field in URB_CONTROL_VENDOR_OR_CLASS_REQUEST.

Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

2.2.9.13 TS_URB_CONTROL_GET_CONFIGURATION_REQUEST

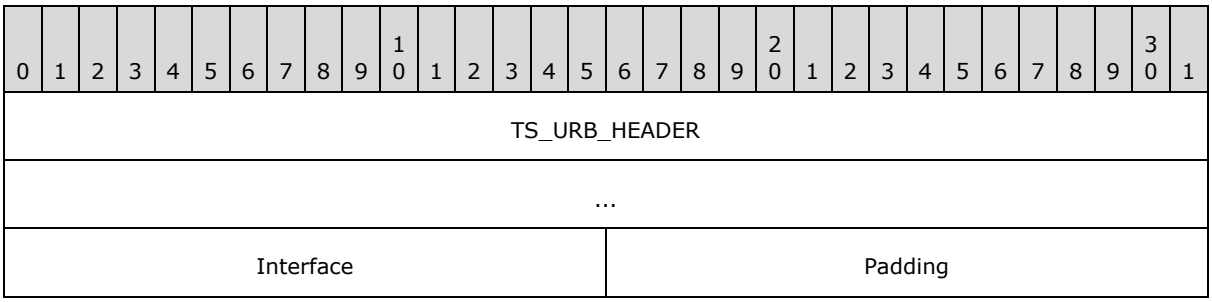
This packet represents the URB structure URB_CONTROL_GET_CONFIGURATION_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_GET_CONFIGURATION_REQUEST.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TS_URB_HEADER																															
...																															

TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

2.2.9.14 TS_URB_CONTROL_GET_INTERFACE_REQUEST

This packet represents the URB structure URB_CONTROL_GET_INTERFACE_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_GET_INTERFACE_REQUEST.



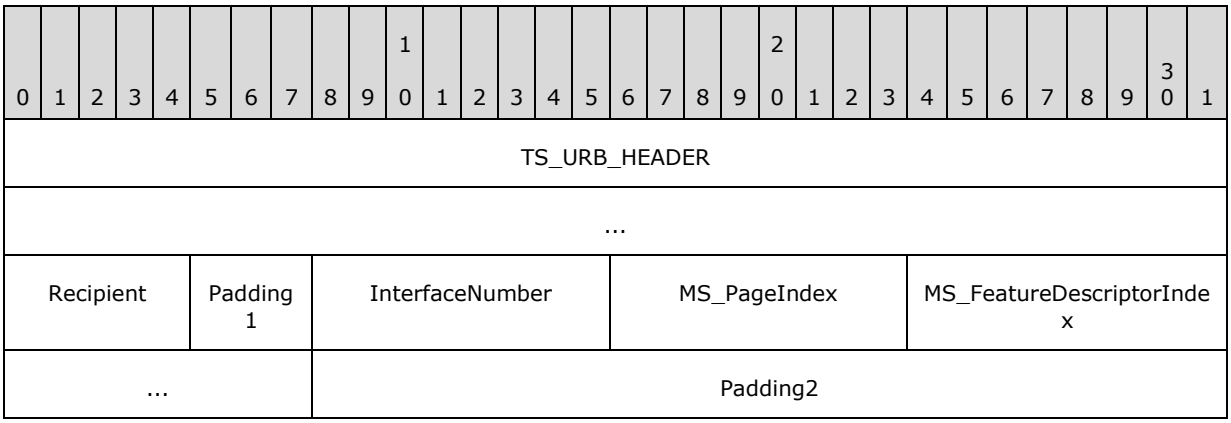
TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

Interface (2 bytes): A 16-bit unsigned integer. This value is from the **Interface** field in URB_CONTROL_GET_INTERFACE_REQUEST.

Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

2.2.9.15 TS_URB_OS_FEATURE_DESCRIPTOR_REQUEST

This packet represents the URB structure URB_OS_FEATURE_DESCRIPTOR_REQUEST, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. The packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_OS_FEATURE_DESCRIPTOR_REQUEST.



TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

Recipient (5 bits): A 5-bit field. This value is from the **Interface** field in URB_OS_FEATURE_DESCRIPTOR_REQUEST. When setting this field in URB_OS_FEATURE_DESCRIPTOR_REQUEST, the value of the **Recipient** field is used for the low 5 bits and the highest bits MUST be set to 0.

Padding1 (3 bits): A 3-bit field for padding. This field can be set to any value and MUST be ignored upon receipt.

InterfaceNumber (1 byte): An 8-bit unsigned integer. This value is from the **InterfaceNumber** field in URB_OS_FEATURE_DESCRIPTOR_REQUEST.

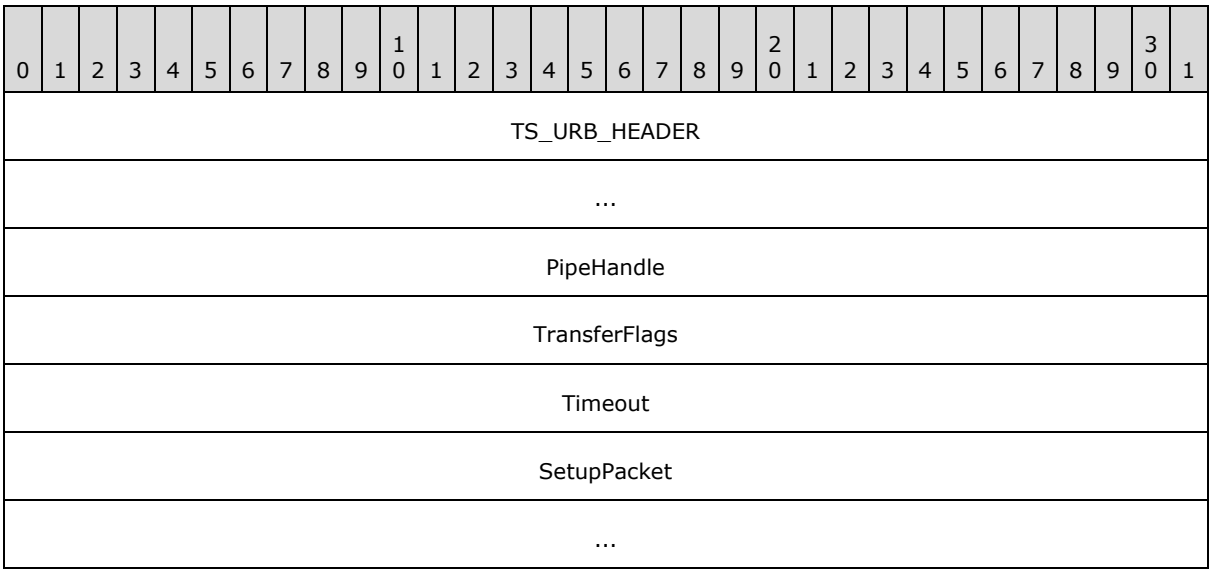
MS_PageIndex (1 byte): An 8-bit unsigned integer. This value is from the **MS_PageIndex** field in URB_OS_FEATURE_DESCRIPTOR_REQUEST.

MS_FeatureDescriptorIndex (2 bytes): A 16-bit unsigned integer. This value is from the **MS_FeatureDescriptorIndex** field in URB_OS_FEATURE_DESCRIPTOR_REQUEST.

Padding2 (3 bytes): A 24-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

2.2.9.16 TS_URB_CONTROL_TRANSFER_EX

This packet represents the URB structure URB_CONTROL_TRANSFER_EX, as specified in [MSFT-W2KDDK] Volume 2, Part 4, Chapter 3. URB_CONTROL_TRANSFER_EX is same as URB_CONTROL_TRANSFER except URB_CONTROL_TRANSFER_EX contains a new field called timeout following the TransferBufferMDL field. The timeout field in URB_CONTROL_TRANSFER_EX is 32-bit unsigned integer. If the **TransferFlags** field in URB_CONTROL_TRANSFER_EX contains the USBD_TRANSFER_DIRECTION_IN flag, the packet is sent using the TRANSFER_IN_REQUEST message with the **OutputBufferSize** field set to **TransferBufferLength** as defined in URB_CONTROL_TRANSFER_EX; otherwise, the packet is sent using the TRANSFER_OUT_REQUEST message with the **InputBufferSize** field set to **TransferBufferLength** and the **InputBuffer** field set to the data in **TransferBuffer** or **TransferBufferMDL** as defined in URB_CONTROL_TRANSFER_EX.



TS_URB_HEADER (8 bytes): A TS_URB_HEADER as specified in section 2.2.9.1.1.

PipeHandle (4 bytes): A 32-bit unsigned integer. The handle returned from the client after it successfully completes a TS_URB_SELECT_INTERFACE request.

TransferFlags (4 bytes): A 32-bit unsigned integer. This value is from the **TransferFlags** field in URB_CONTROL_TRANSFER_EX.

Timeout (4 bytes): A 32-bit unsigned integer. This value is from the **Timeout** field in URB_CONTROL_TRANSFER_EX. This value indicates the time, in milliseconds, before the request times out. A value of zero indicates that there is no timeout for this request. The value of this field is passed to the physical device.

SetupPacket (8 bytes): An array of 8-bytes. This value is from the **SetupPacket** field in URB_CONTROL_TRANSFER_EX.

2.2.10 TS_URB_RESULT Structures

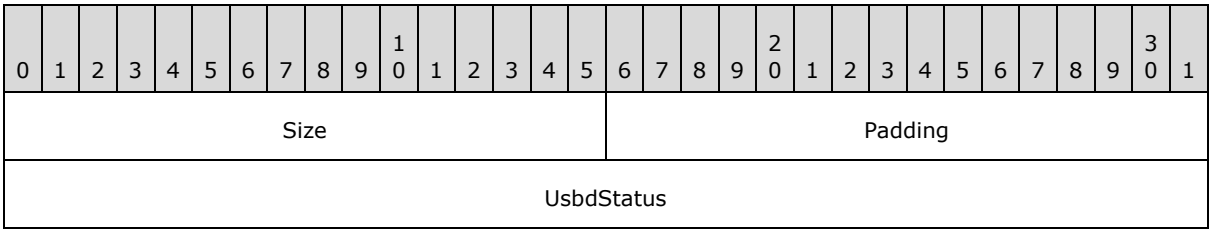
The TS_URB_RESULT structures sent in response to the TRANSFER_IN_REQUEST and TRANSFER_OUT_REQUEST messages, are sent via the URB_COMPLETION or URB_COMPLETION_NO_DATA messages. These messages contain the TS_URB_RESULT field, which is described in this section.

All the fields in TS_URB_RESULT are the output fields defined in URB. For information on URB definitions, see [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.

2.2.10.1 Common Structures

2.2.10.1.1 TS_URB_RESULT_HEADER

Every TS_URB_RESULT structure begins with a common header called TS_URB_RESULT_HEADER.



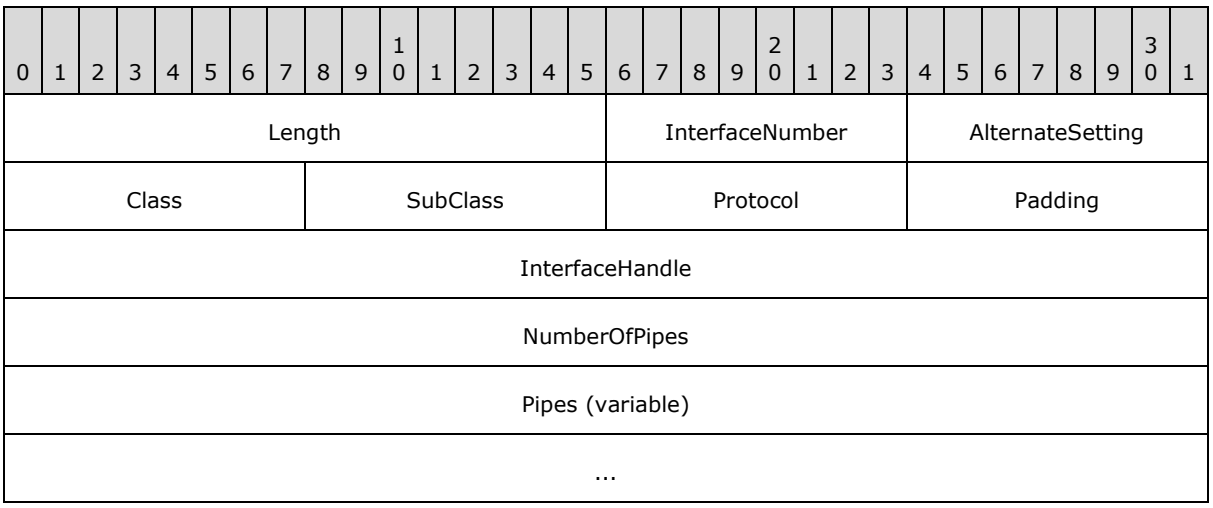
Size (2 bytes): A 16-bit unsigned integer. The size, in bytes, of the TS_URB_RESULT structure.

Padding (2 bytes): A 16-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

UsbdStatus (4 bytes): A 32-bit unsigned integer. This value represents the **Status** field of the **URB_STATUS** structure as specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.

2.2.10.1.2 TS_USBD_INTERFACE_INFORMATION_RESULT

The TS_USBD_INTERFACE_INFORMATION_RESULT structure is based on the USBD_INTERFACE_INFORMATION structure as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.



Length (2 bytes): A 16-bit unsigned integer. The size, in bytes, of the TS_USBD_INTERFACE_INFORMATION_RESULT structure.

InterfaceNumber (1 byte): A 8-bit unsigned integer. This value represents the **InterfaceNumber** field in USBD_INTERFACE_INFORMATION.

AlternateSetting (1 byte): A 8-bit unsigned integer. This value represents the **AlternateSetting** field in USBD_INTERFACE_INFORMATION.

Class (1 byte): A 8-bit unsigned integer. This value represents the **Class** field in USBD_INTERFACE_INFORMATION.

SubClass (1 byte): A 8-bit unsigned integer. This value represents the **SubClass** field in USBD_INTERFACE_INFORMATION.

Protocol (1 byte): A 8-bit unsigned integer. This value represents the **Protocol** field in USBD_INTERFACE_INFORMATION.

Padding (1 byte): A 8-bit unsigned integer for padding. This field can be set to any value and MUST be ignored upon receipt.

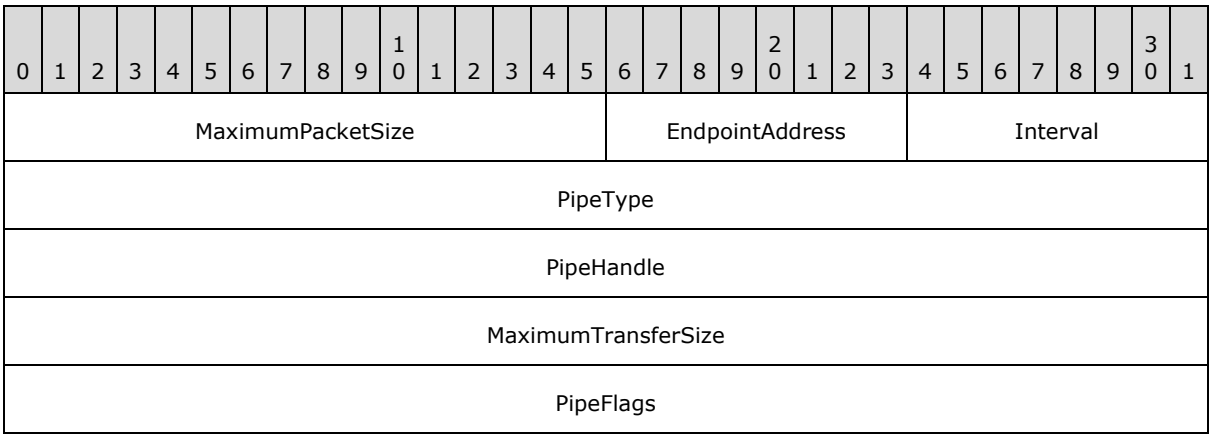
InterfaceHandle (4 bytes): A 32-bit unsigned integer. This value represents the **InterfaceHandle** field in USBD_INTERFACE_INFORMATION.

NumberOfPipes (4 bytes): A 32-bit unsigned integer. This value represents the **NumberOfPipes** field in USBD_INTERFACE_INFORMATION. It also indicates the number of Pipes array elements that are to follow.

Pipes (variable): An array of TS_USBD_PIPE_INFORMATION_RESULT structures. The number of array elements is determined by the **NumberOfPipes** field.

2.2.10.1.3 TS_USBD_PIPE_INFORMATION_RESULT

The TS_USBD_PIPE_INFORMATION_RESULT is based on the USBD_PIPE_INFORMATION structure as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 3.



MaximumPacketSize (2 bytes): A 16-bit unsigned integer. This value represents the **MaximumPacketSize** field in USBD_PIPE_INFORMATION.

EndpointAddress (1 byte): A 8-bit unsigned integer. This value represents the **EndpointAddress** field in USBD_PIPE_INFORMATION.

Interval (1 byte): A 8-bit unsigned integer. This value represents the **Interval** field in USBD_PIPE_INFORMATION.

PipeType (4 bytes): A 32-bit unsigned integer. This value represents the **PipeType** field in USBD_PIPE_INFORMATION.

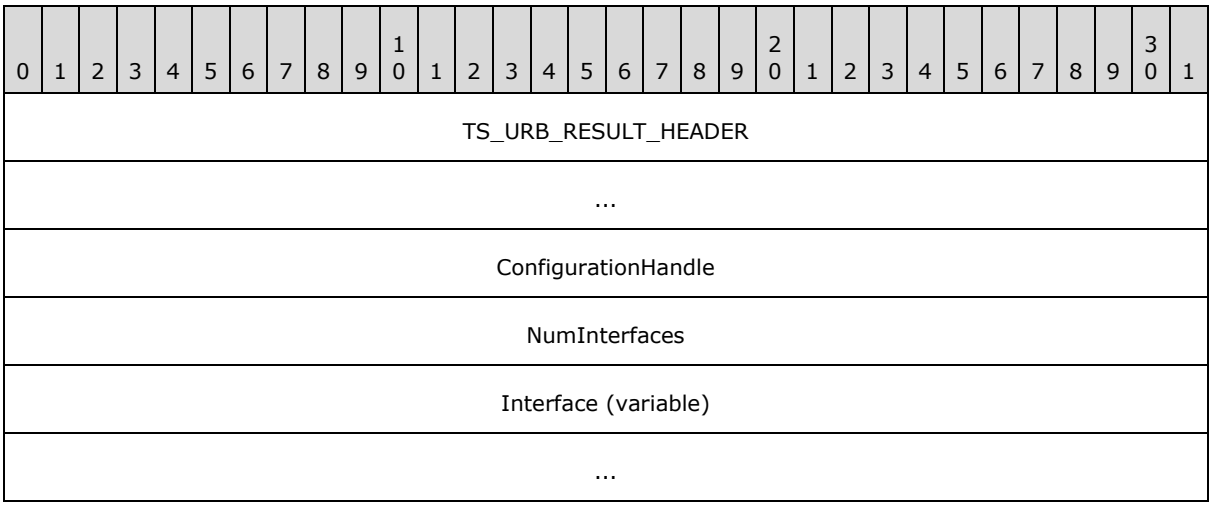
PipeHandle (4 bytes): A 32-bit unsigned integer. This value represents the **PipeHandle** field in USBD_PIPE_INFORMATION.

MaximumTransferSize (4 bytes): A 32-bit unsigned integer. This value represents the **MaximumTransferSize** field in USBD_PIPE_INFORMATION.

PipeFlags (4 bytes): A 32-bit unsigned integer. This value represents the **PipeFlags** field in USBD_PIPE_INFORMATION.

2.2.10.2 TS_URB_SELECT_CONFIGURATION_RESULT

This packet represents the result of the TRANSFER_IN_REQUEST with TS_URB_SELECT_CONFIGURATION. The TS_URB_SELECT_CONFIGURATION_RESULT is sent via the URB_COMPLETION_NO_DATA message.



TS_URB_RESULT_HEADER (8 bytes): A TS_URB_RESULT_HEADER as specified in section 2.2.10.1.1.

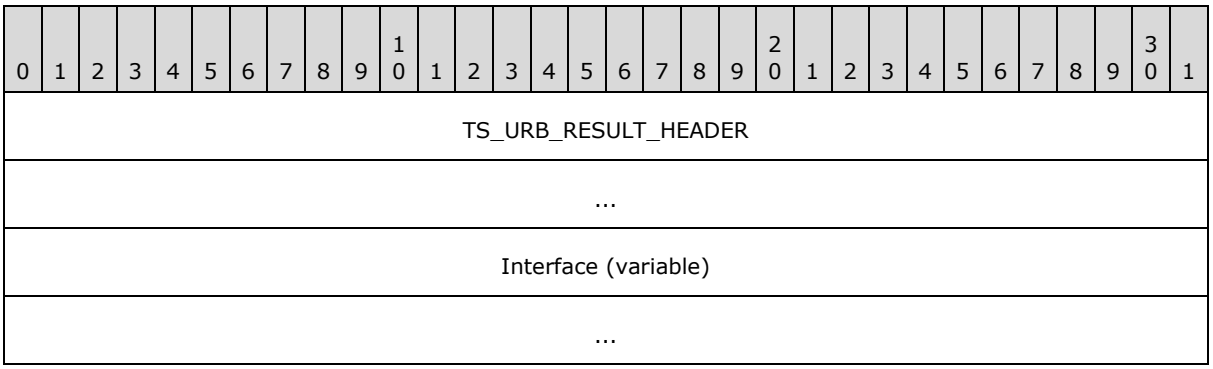
ConfigurationHandle (4 bytes): A 32-bit unsigned integer. An opaque handle that identifies the configuration described by the TS_URB_SELECT_CONFIGURATION operation.

NumInterfaces (4 bytes): A 32-bit unsigned integer. The number of **Interface** fields that are to follow.

Interface (variable): TS_USBD_INTERFACE_INFORMATION_RESULT structure as specified in section 2.2.10.1.2.

2.2.10.3 TS_URB_SELECT_INTERFACE_RESULT

This packet represents the result of the TRANSFER_IN_REQUEST with TS_URB_SELECT_INTERFACE. The TS_URB_SELECT_CONFIGURATION_RESULT structure is sent via the URB_COMPLETION_NO_DATA message.

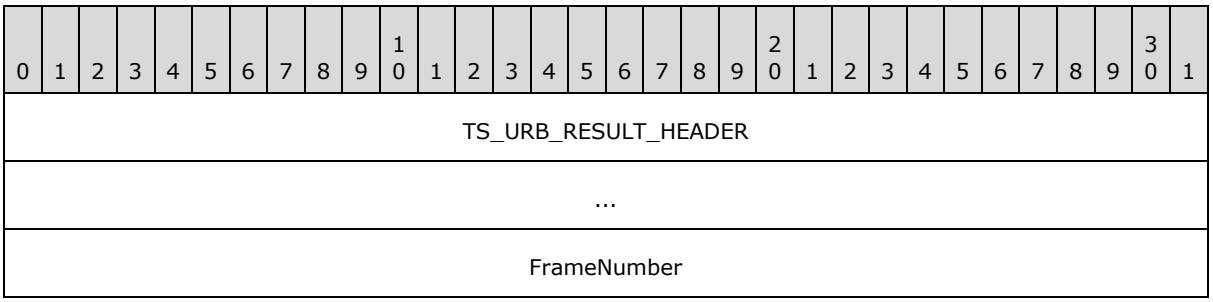


TS_URB_RESULT_HEADER (8 bytes): A TS_URB_RESULT_HEADER as specified in section 2.2.10.1.1.

Interface (variable): A TS_USBD_INTERFACE_INFORMATION_RESULT structure as specified in section 2.2.10.1.2.

2.2.10.4 TS_URB_GET_CURRENT_FRAME_NUMBER_RESULT

This packet represents the result of the TRANSFER_IN_REQUEST with TS_URB_GET_CURRENT_FRAME_NUMBER. The TS_URB_GET_CURRENT_FRAME_NUMBER_RESULT structure is sent via the URB_COMPLETION_NO_DATA message.

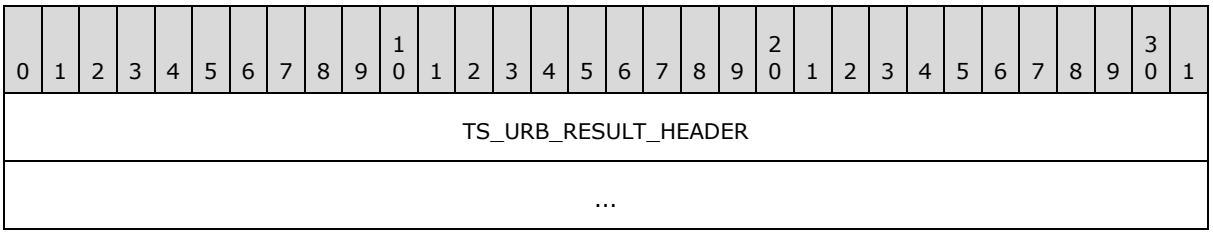


TS_URB_RESULT_HEADER (8 bytes): A TS_URB_RESULT_HEADER as specified in section 2.2.10.1.1.

FrameNumber (4 bytes): A 32-bit unsigned integer. The current frame number.

2.2.10.5 TS_URB_ISOCH_TRANSFER_RESULT

This packet represents the result of TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST with TS_URB_ISOCH_TRANSFER. The TS_URB_ISOCH_TRANSFER_RESULT structure is sent via the URB_COMPLETION message if the result contains the data buffer to be sent back; otherwise, the TS_URB_ISOCH_TRANSFER_RESULT is sent via the URB_COMPLETION_NO_DATA message.



StartFrame
NumberOfPackets
ErrorCount
IsoPacket (variable)
...

TS_URB_RESULT_HEADER (8 bytes): A TS_URB_RESULT_HEADER as specified in section 2.2.10.1.1.

StartFrame (4 bytes): A 32-bit unsigned integer. The resulting StartFrame value as specified in URB_ISOCH_TRANSFER.

NumberOfPackets (4 bytes): A 32-bit unsigned integer. This value is the number of URB_ISOCH_TRANSFER following the **IsoPacket** field.

ErrorCount (4 bytes): A 32-bit unsigned integer. The resulting ErrorCount value as described in URB_ISOCH_TRANSFER.

IsoPacket (variable): The resulting array of USBD_ISO_PACKET_DESCRIPTOR structures as described in URB_ISOCH_TRANSFER.

For the TRANSFER_IN_REQUEST operation, the **IsoPacket** field describes the data validity in the stream of data that the physical device has generated. Each **IsoPacket** field describes a different part of the data stream. If the **IsoPacket** field indicates an error, the part of the data stream it describes does not contain valid data and the client SHOULD NOT send it to the server. When a client constructs the **OutputDataBuffer** field for a URB_COMPLETION message that contains TS_URB_ISOCH_TRANSFER_RESULT structure, the client MUST copy the data from the data stream into the **OutputDataBuffer** field if and only if the corresponding **IsoPacket** indicates no error.

2.2.11 USB_DEVICE_CAPABILITIES

The USB_DEVICE_CAPABILITIES structure defines the capabilities of a USB device.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CbSize																															
UsbBusInterfaceVersion																															
USBDI_Version																															
Supported_USB_Version																															
HcdCapabilities																															

DeviceIsHighSpeed
NoAckIsochWriteJitterBufferSizeInMs

CbSize (4 bytes): A 32-bit unsigned integer. The byte size of this structure. This value MUST be 28.

UsbBusInterfaceVersion (4 bytes): A 32-bit unsigned integer. The USB version the device supports.

Value	USB version supported
0x00000000	0
0x00000001	1
0x00000002	2

USBDI_Version (4 bytes): A 32-bit unsigned integer. The highest USBDI version the device supports. This value can be 0x00000500 or 0x00000600.

Supported_USB_Version (4 bytes): A 32-bit unsigned integer. The version of USB the device supports. The value MUST be one of the following:

Name	Value
USB 1.0	0x100
USB 1.1	0x110
USB 2.0	0x200

HcdCapabilities (4 bytes): A 32-bit unsigned integer. The host capabilities supported. This value MUST always be zero.

DeviceIsHighSpeed (4 bytes): A 32-bit unsigned integer. This value represents the device speed. 0x00000000 if the device is full speed and 0x00000001 if the device is high speed. If **UsbBusInterfaceVersion** is 0x00000000, **DeviceIsHighSpeed** MUST be 0x00000000. A high speed device operates as a USB 2.0 device while a full speed device operates as a USB 1.1 device.

NoAckIsochWriteJitterBufferSizeInMs (4 bytes): A 32-bit unsigned integer. If the value is nonzero, the client supports TS_URB_ISOCH_TRANSFER messages that do not expect URB_COMPLETION messages; otherwise, if the value is zero, the client does not support TS_URB_ISOCH_TRANSFER messages. If the value is not zero, the value represents the amount of outstanding isochronous data the client expects from the server. If this value is nonzero, it MUST be greater than or equal to 10 and less than or equal to 512.

2.2.12 USB IO Control Code

The IO_CONTROL messages are sent for each I/O request that the device driver sends to the USB device. Each I/O request contains a value called the I/O control code. This I/O control code specifies what operation is requested in the I/O request. This section describes the I/O control codes that the server supports.

2.2.12.1 IOCTL_INTERNAL_USB_RESET_PORT

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_RESET_PORT, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to zero.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation and the **OutputBufferSize** field set to zero.

2.2.12.2 IOCTL_INTERNAL_USB_GET_PORT_STATUS

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_GET_PORT_STATUS, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to 0x4.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to 0x4 and set the **OutputBuffer** field to the USB port status. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

2.2.12.3 IOCTL_INTERNAL_USB_GET_HUB_COUNT

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_GET_HUB_COUNT, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to 0x4.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to 0x4 and set the **OutputBuffer** field to the hub count. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

2.2.12.4 IOCTL_INTERNAL_USB_CYCLE_PORT

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_CYCLE_PORT, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to zero.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation and the **OutputBufferSize** field set to zero.

2.2.12.5 IOCTL_INTERNAL_USB_GET_HUB_NAME

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_GET_HUB_NAME, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to Parameters.DeviceIoControl.OutputBufferLength as described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to length of the hub name and set the **OutputBuffer** field to the hub name. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

2.2.12.6 IOCTL_INTERNAL_USB_GET_BUS_INFO

This USB IOCTL is specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_GET_BUS_INFO, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to the size of USB_BUS_NOTIFICATION as specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to size of USB_BUS_NOTIFICATION and set the **OutputBuffer** field to USB_BUS_NOTIFICATION. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

2.2.12.7 IOCTL_INTERNAL_USB_GET_CONTROLLER_NAME

This USB IOCTL is described in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

The server converts this IOCTL into an IO_CONTROL message with the **IoControlCode** field set to IOCTL_INTERNAL_USB_GET_CONTROLLER_NAME, the **InputBufferSize** field set to zero, and the **OutputBufferSize** field set to Parameters.Others.Argument2 as specified in [MSFT-W2KDDK], Volume 2, Part 4, Chapter 1.

In response to the IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to size of controller name and set the **OutputBuffer** field to the controller name. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

2.2.13 USB Internal IO Control Code

2.2.13.1 IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME

The IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME value is defined as 0x00224000. The INTERNAL_IO_CONTROL message with IOCTL code IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME is sent when a request to query the device's current frame number as specified in [USB-SPC2.0] USB 2.0 Specification, section 10.2.3 Frame and Microframe Generation is received.

The server converts the query current frame number call request into an INTERNAL_IO_CONTROL message with **IoControlCode** set to IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME, the **InputBufferSize** field is set to zero, and the **OutputBufferSize** field is set to 0x4.

In response to the INTERNAL_IO_CONTROL message, an IOCONTROL_COMPLETION message is sent with the final result of the operation. If the operation is successful, the client MUST set the **OutputBufferSize** field to 0x4 and set the **OutputBuffer** field to the current frame number. If the operation is not successful, the client MUST set the **OutputBufferSize** field to zero.

3 Protocol Details

3.1 Common Details

The following state diagram illustrates the state transitions that both the client and the server go through.

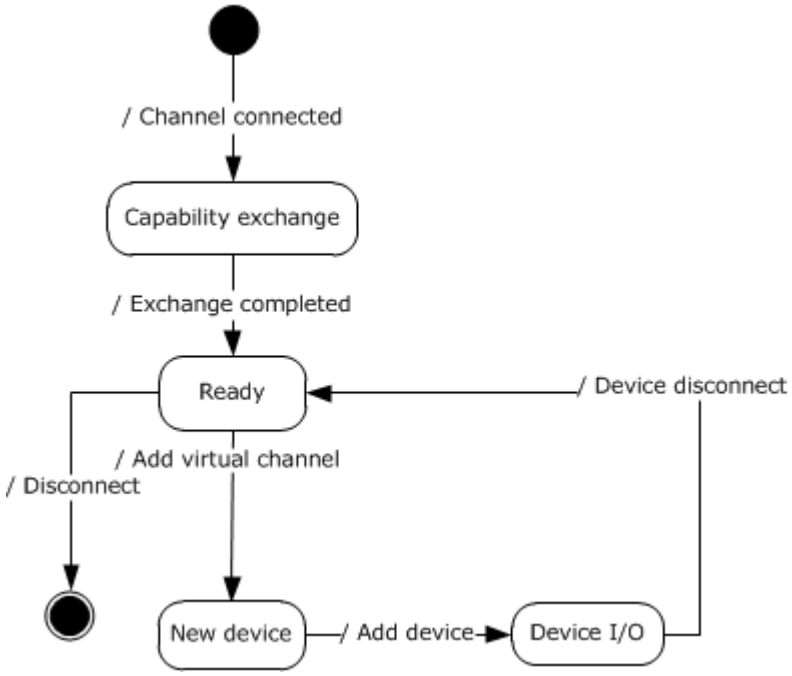


Figure 5: Client and server state transitions

Channel-connected event: This event signifies that the underlying transport channel is connected, as specified in section 2.1.

Capability-exchange state: The client and the server are exchanging capabilities, as described in section 1.3.1.1.

Exchange-completed event: Signifies that the capability exchange is completed, that is, the client has sent a Channel Created message (see section 2.2.5.1).

Ready state: The protocol is ready to redirect new devices.

Add virtual channel event: As described in section 1.3.1.2, a new device has arrived on the client and the protocol is ready to redirect it.

Add device event: This event signifies that the device is ready for I/O, as described in section 1.3.1.2.

Device I/O state: As described by section 1.3.1.3, the device is ready to exchange I/O.

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 described organization is provided to facilitate the

explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

RequestId: For each IO request that is sent to the client's actual USB device, the server generates a unique **RequestId** for the request. The server sends the **RequestId** to the client in the **RequestId** field of the IO_CONTROL or INTERNAL_IO_CONTROL message. If the request is to be sent as a TRANSFER_IN_REQUEST message or a TRANSFER_OUT_REQUEST message, the **RequestId** field is sent in the **TsUrb** field of the message. IO_CONTROL, INTERNAL_IO_CONTROL, TRANSFER_IN_REQUEST, and TRANSFER_OUT_REQUEST messages are classified as IO requests. A **RequestId** is unique among all four types of IO Requests. A **RequestId** value has to be unique until the client sends the final result of the IO request that has the **RequestId** value. Once this has happened, the **RequestId** value can be reused.

list of pending URB requests: For each TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST request that is sent to the client's USB device, the server stores in this list until the appropriate completion message is received. The matching of replies to requests is based on the **RequestId**.

PipeHandle: an ID used to issue TS_URB_PIPE_REQUEST, TS_URB_CONTROL_TRANSFER, TS_URB_BULK_OR_INTERRUPT_TRANSFER, TS_URB_ISOCH_TRANSFER or TS_URB_CONTROL_TRANSFER_EX. The value is send by the client in TS_USBD_PIPE_INFORMATION_RESULT structure in response to TS_URB_SELECT_INTERFACE request.

3.1.1.1 Interface Manipulation Data Model

The common details of the abstract data model for the interface manipulation infrastructure are specified in [MS-RDPEXPS] sections 3.1.1. The interface manipulation applies to the following fields: **InterfaceId**, **MessageId**, and **FunctionId**.

3.1.2 Timers

None.

3.1.3 Initialization

The dynamic virtual channel MUST be established, using the parameters specified in section 2.1, before protocol operation commences.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

Malformed packets are packets that don't adhere to the rules described in sections 2 and 3 with the exception of sections 3.2.5 and 3.3.5. Out-of-sequence packets are packets that don't adhere to the rules in sections 3.2.5 and 3.3.5. Malformed and out-of-sequence packets MUST be ignored by the server and the client.

There are no time-outs for receiving a reply for any request.

3.1.5.1 Processing a Shared Message Header

The common rules for processing the SHARED_MSG_HEADER for the interface manipulation infrastructure are defined in [MS-RDPEXPS] section 3.1.5.1.

3.1.5.2 Interface Manipulation

The common rules for processing the interface manipulation messages are defined in [MS-RDPEXPS] section 3.1.5.2. Any interface, including the default one, MUST be released with an Interface Release message if the side that has received it or owned it as default is finished sending messages over that interface.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

The abstract data model is as specified in section 3.1.1.

3.2.2 Timers

None.

3.2.3 Initialization

Initialization is as specified in section 3.1.3.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Processing Events and Sequencing Rules

3.2.5.1 Device Sink Interface

3.2.5.1.1 Processing an Add Virtual Channel Message

The structure and fields of the ADD_VIRTUAL_CHANNEL message are specified in section 2.2.4.1.

After receiving the ADD_VIRTUAL_CHANNEL message, the server makes a new instance of a dynamic virtual channel for USB redirection.

If the server receives an invalid ADD_VIRTUAL_CHANNEL message, the server shall terminate the dynamic virtual channel.

3.2.5.1.2 Processing a Add Device Message

The structure and fields of the ADD_DEVICE message are specified in section 2.2.4.2.

After receiving the ADD_DEVICE message, the server MUST create a **remote device** instance on the server to represent the client-side physical device. The ADD_DEVICE message contains a unique USB device interface ID to represent the client-side physical device. The server maintains this interface ID and uses it to identify the client-side physical device when communicating to the client.

In the case of the server receiving a duplicate interface ID, the server MUST ignore the ADD_DEVICE message. The original device with the same interface ID MUST not be affected by this ADD_DEVICE message and continue to function with no interruption.

3.2.5.2 Channel Notification Interface

3.2.5.2.1 Sending a Channel Created Message

The structure and fields of the CHANNEL_CREATED message are specified in section 2.2.5.1.

The server sends the CHANNEL_CREATED message to the client to report the version of USB redirection it supports.

3.2.5.2.2 Processing a Channel Created Message

The structure and fields of the CHANNEL_CREATED message are specified in section 2.2.5.1.

After receiving the CHANNEL_CREATED message, the server validates the client USB redirection version. If the server does not support the client's USB redirection version, it MUST close the dynamic virtual channel. If the server supports the client's USB redirection version, it MUST begin processing the Device Sink interface messages.

3.2.5.3 USB Device Interface

3.2.5.3.1 Sending a Cancel Request Message

The structure and fields of the CANCEL_REQUEST message are specified in section 2.2.6.1.

The server sends the CANCEL_REQUEST message to request the client to stop processing the request specified by the **RequestId**. The request with the given **RequestId** could already have been completed by the client via the Request Completion Interface.

3.2.5.3.2 Sending a Register Request Callback Message

The structure and fields of the REGISTER_REQUEST_CALLBACK message are specified in section 2.2.6.2.

The server sends the REGISTER_REQUEST_CALLBACK message to the client in order to provide a unique Request Completion Interface for the client to use. The server MUST send this message once for the same USB device and it MUST send this message before sending an IO_CONTROL, INTERNAL_IO_CONTROL, TRANSFER_IN_REQUEST, or TRANSFER_OUT_REQUEST message.

3.2.5.3.3 Sending a IO Control Message

The structure and fields of the IO_CONTROL message are specified in section 2.2.6.3.

The server sends the IO_CONTROL message to the client in order to forward an IO control request to the physical device on the client-side.

3.2.5.3.4 Sending an Internal IO Control Message

The structure and fields of the INTERNAL_IO_CONTROL message are specified in section 2.2.6.4.

The server sends the INTERNAL_IO_CONTROL message to the client in order to forward an Internal IO control request to the physical device on the client-side.

3.2.5.3.5 Sending a Query Device Text Message

The structure and fields of the QUERY_DEVICE_TEXT message are specified in section 2.2.6.5.

The server sends the QUERY_DEVICE_TEXT message to the client when it receives a request to query the USB's device text from the system.

3.2.5.3.6 Processing a Query Device Text Response Message

The structure and fields of the QUERY_DEVICE_TEXT RSP message are specified in section 2.2.6.6.

After receiving the QUERY_DEVICE_TEXT_RSP message, the server MUST return the description contained in **DeviceDescription** field of QUERY_DEVICE_TEXT_RSP to the actual application on behalf of which the QUERY_DEVICE_TEXT operation request was sent.

3.2.5.3.7 Sending a Transfer In Request Message

The structure and fields of the TRANSFER_IN_REQUEST message are specified in section 2.2.6.7.

The server sends the TRANSFER_IN_REQUEST message to the client in order to forward an URB to the physical device on the client-side and the URB requests data from the device. The request is stored in the **list of pending URB requests** until it is completed.

3.2.5.3.8 Sending a Transfer Out Request Message

The structure and fields of the TRANSFER_OUT_REQUEST message are specified in section 2.2.6.8.

The server sends the TRANSFER_OUT_REQUEST Message to the client in order to forward an URB to the physical device on the client-side and the URB requests to write data to the device. The request is stored in the **list of pending URB requests** until it is completed.

3.2.5.3.9 Sending a Retract Device Message

The structure and fields of the Retract Device message are specified in section 2.2.6.9.

The server sends the Retract Device message to the client when the server fails to start the device due to group policy.

3.2.5.4 Request Completion Interface

3.2.5.4.1 IO Control Completion Message

The structure and fields of the IOCONTROL_COMPLETION message are specified in section 2.2.7.1.

After receiving the IOCONTROL_COMPLETION message, the server MUST use the **RequestId** specified in the IOCONTROL_COMPLETION message to find the associated information stored after sending the IO_CONTROL or INTERNAL_IO_CONTROL message; that information is stored in the **HResult**, **Information**, **OutputBufferSize**, and **OutputBuffer** fields of the IOCONTROL_COMPLETION message. With this information, the server completes the original request. The server MUST redirect the result contained in the IOCONTROL_COMPLETION to the actual application that made the IO Control or Internal IO Control operation request.

The server expects one and only one IOCONTROL_COMPLETION message for each IO_CONTROL or INTERNAL_IO_CONTROL message it sends to the client. If the server receives more than one IOCONTROL_COMPLETION message for an IO_CONTROL or INTERNAL_IO_CONTROL message, the server SHOULD terminate the dynamic virtual channel.

If the server receives an IOCONTROL_COMPLETION message with an invalid **RequestId**, the server SHOULD terminate the dynamic virtual channel.

If the **OutputBufferSize** field in the IOCONTROL_COMPLETION message is greater than the **OutputBufferSize** field in the corresponding IO_CONTROL or INTERNAL_IO_CONTROL message, the server SHOULD terminate the dynamic virtual channel.

3.2.5.4.2 URB Completion Message

The structure and fields of the URB_COMPLETION message are specified in section 2.2.7.2.

After receiving the URB_COMPLETION message, the server MUST use the **RequestId** specified in the URB_COMPLETION message to find the associated information stored after sending the TRANSFER_IN_REQUEST message from the **list of pending URB requests**; that information is stored in the **CTsUrbResult**, **TsUrbResult**, **HResult**, **OutputBufferSize**, and **OutputBuffer** fields of the URB_COMPLETION message. With this information, the server completes the original request. The server MUST redirect the result contained in the URB_COMPLETION message to the actual application that made the Transfer In operation request.

The server expects one and only one URB_COMPLETION message for each TRANSFER_IN_REQUEST message it sends to the client, if the URB_COMPLETION message contains output data. If the server receives more than one URB_COMPLETION message for a TRANSFER_IN_REQUEST message, the server SHOULD terminate the dynamic virtual channel.

If the server receives an URB_COMPLETION message with an invalid **RequestId**, the server SHOULD terminate the dynamic virtual channel.

If the **OutputBufferSize** field in the URB_COMPLETION message is greater than the **OutputBufferSize** field in the corresponding TRANSFER_IN_REQUEST message, the server SHOULD terminate the dynamic virtual channel.

3.2.5.4.3 URB Completion No Data Message

The structure and fields of the URB_COMPLETION_NO_DATA message are specified in section 2.2.7.3.

After receiving the URB_COMPLETION_NO_DATA message, the server MUST use the **RequestId** specified in the URB_COMPLETION_NO_DATA message to find the associated information stored after sending the TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST message from the **list of pending URB requests**; that information is stored in the **CTsUrbResult**, **TsUrbResult**, **HResult**, and **OutputBufferSize** fields of the URB_COMPLETION_NO_DATA message. With this information, the server completes the original request. The server MUST redirect the result contained in the

URB_COMPLETION_NO_DATA message to the actual application that made the Transfer In or Transfer Out operation request.

The server expects one and only one URB_COMPLETION_NO_DATA message for each Transfer In operation that generates no output data or each Transfer Out operation. If the server receives more than one URB_COMPLETION_NO_DATA message for a TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST message, the server SHOULD terminate the dynamic virtual channel.

If the server receives an URB_COMPLETION_NO_DATA message with an invalid **RequestId**, the server SHOULD terminate the dynamic virtual channel.

If the **OutputBufferSize** field in the URB_COMPLETION_NO_DATA message is not zero and the URB_COMPLETION_NO_DATA message is the result of a Transfer In operation, the server SHOULD terminate the dynamic virtual channel.

If the **OutputBufferSize** field in the URB_COMPLETION_NO_DATA message is greater than the **OutputBufferSize** field in the corresponding TRANSFER_OUT_REQUEST message, the server SHOULD terminate the dynamic virtual channel.

3.2.5.5 Interface Manipulation Exchange Capabilities Interface

3.2.5.5.1 Sending an Interface Manipulation Exchange Capabilities Request Message

The structure and fields of the RIM_EXCHANGE_CAPABILITY_REQUEST message are specified in section 2.2.3.1.

The server MUST send this message when the USB redirection virtual channel is connected. This message MUST be sent before the Channel created message (section 2.2.5.1).

3.2.5.5.2 Processing an Interface Manipulation Exchange Capabilities Response Message

The structure and fields of the RIM_EXCHANGE_CAPABILITY_RESPONSE message are specified in section 2.2.3.2.

On receiving this message, the server confirms that the client meets the minimum capabilities for interface manipulation.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Client Details

3.3.1 Abstract Data Model

The abstract data model is as specified in section 3.1.1.

3.3.2 Timers

None.

3.3.3 Initialization

Initialization is as specified in section 3.1.3.

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Processing Events and Sequencing Rules

3.3.5.1 Device Sink Interface

3.3.5.1.1 Sending a Add Virtual Channel Message

The structure and fields of the ADD_VIRTUAL_CHANNEL message are specified in section 2.2.4.1.

The client sends the ADD_VIRTUAL_CHANNEL message to server to request the server to create a new instance of dynamic virtual channel for USB redirection. The client sends this message for every USB device to be redirected. This isolates messages for each USB device in its own instance of a dynamic virtual channel.

3.3.5.1.2 Sending a Add Device Message

The structure and fields of the ADD_DEVICE message are specified in section 2.2.4.2.

The client sends this ADD_DEVICE message to the server to redirect a USB device. The message contains a unique **InterfaceId** that is used for I/O requests.

3.3.5.2 Channel Notification Interface

3.3.5.2.1 Sending a Channel Created Message

The structure and fields of the CHANNEL_CREATED message are specified in section 2.2.5.1.

The client sends the CHANNEL_CREATED message to the server to report the version of the USB redirection it supports.

3.3.5.2.2 Processing a Channel Created Message

The structure and fields of the CHANNEL_CREATED message are specified in section 2.2.5.1.

After receiving the CHANNEL_CREATED message, the client validates the server USB redirection version. If the client does not support the server's USB redirection version, the client MUST close the dynamic virtual channel. If the client supports the server's USB redirection version, it MUST begin sending Device Sink interface messages.

3.3.5.3 USB Device Interface

3.3.5.3.1 Processing a Cancel Request Message

The structure and fields of the CANCEL_REQUEST message are specified in section 2.2.6.1.

After receiving the CANCEL_REQUEST message, the client MUST attempt to stop processing the request identified by the **RequestId** field in the CANCEL_REQUEST message. If the current request has not been completed it MUST be canceled. If the request has been completed, the client MUST ignore this CANCEL_REQUEST message.

3.3.5.3.2 Processing a Register Request Callback Message

The structure and fields of the REGISTER_REQUEST_CALLBACK message are specified in section 2.2.6.2.

After receiving the REGISTER_REQUEST_CALLBACK message, the client MUST use the **InterfaceId** value from the **RegisterCompletion** field when sending final results of IO requests received via the IO_CONTROL, INTERNAL_IO_CONTROL, TRANSFER_IN_REQUEST, or TRANSFER_OUT_REQUEST message.

3.3.5.3.3 Processing an IO Control Message

The structure and fields of the IO_CONTROL message are specified in section 2.2.6.3.

After receiving the IO_CONTROL message, the client MUST forward the request to the physical device by retrieving the IOCTL code and input/output buffers from **IoControlCode**, **InputBuffer**, **InputBufferSize** and **OutputBufferSize** fields from the message. The output buffer parameter in the forwarded IOCTL is allocated with the size of **OutputBufferSize** field. When the physical device completes the request, the client MUST send the result of the request to the server via the IOCONTROL_COMPLETION message and the **RequestId** field in the IOCONTROL_COMPLETION message MUST match the **RequestId** in the IO_CONTROL message.

The IO_CONTROL message contains the **OutputBufferSize** field. This indicates the maximum amount of data the client can send to the server when sending the final result of this request. If the physical device returns more data than the **OutputBufferSize** field specifies, the client MUST terminate the dynamic virtual channel.

3.3.5.3.4 Processing an Internal IO Control Message

The structure and fields of the INTERNAL_IO_CONTROL message are specified in section 2.2.6.4.

After receiving the INTERNAL_IO_CONTROL message, the client MUST forward the request to the physical device by using the same rules as specified in section 3.3.5.3.3. When the physical device completes the request, the client MUST send the result of the request to the server via the IOCONTROL_COMPLETION message and the **RequestId** field in the IOCONTROL_COMPLETION message MUST match the **RequestId** in the INTERNAL_IO_CONTROL message.

The INTERNAL_IO_CONTROL message contains **OutputBufferSize** field. This indicates the maximum amount of data the client can send to the server when sending the final result of this request. If the physical device returns more data than the **OutputBufferSize** field specifies, the client MUST terminate the dynamic virtual channel.

3.3.5.3.5 Processing a Query Device Text Message

The structure and fields of the QUERY_DEVICE_TEXT message are specified in section 2.2.6.5.

After receiving the QUERY_DEVICE_TEXT message, the client forwards the request to the physical device. When the physical device completes the request, the client sends the result of the request to the server via QUERY_DEVICE_TEXT_RSP message and the **RequestId** field in the message MUST match the **RequestId** in the QUERY_DEVICE_TEXT message.

3.3.5.3.6 Processing a Transfer In Request Message

The structure and fields of the TRANSFER_IN_REQUEST message are specified in section 2.2.6.7.

After receiving the TRANSFER_IN_REQUEST message, the client MUST forward the request to the physical device. When the physical device completes the request, the client MUST send the result of the request to the server via URB_COMPLETION or URB_COMPLETION_NO_DATA message and the **RequestId** field in the message MUST match the **RequestId** in the TRANSFER_IN_REQUEST message.

If TRANSFER_IN_REQUEST results in data to be returned to the server, the client MUST use the URB_COMPLETION message to send the result. If TRANSFER_IN_REQUEST results in no data to be returned to the server, the client MUST use the URB_COMPLETION_NO_DATA message to send the result and the **OutputBufferSize** field MUST be zero.

The TRANSFER_IN_REQUEST message contains **OutputBufferSize** field. This indicates the maximum amount of data the client can send to the server when sending the final result of this request via URB_COMPLETION. If the physical device returns more data than the **OutputBufferSize** field specifies, the client MUST terminate the dynamic virtual channel.

3.3.5.3.7 Processing a Transfer Out Request Message

The structure and fields of the TRANSFER_OUT_REQUEST message are specified in section 2.2.6.8.

After receiving the TRANSFER_OUT_REQUEST message, the client forwards the request to the physical device. When the physical device completes the request, the client sends the result of the request to the server via URB_COMPLETION_NO_DATA message and the **RequestId** field in the message MUST match the **RequestId** in the TRANSFER_OUT_REQUEST message.

The TRANSFER_OUT_REQUEST message contains the **OutputBufferSize** field. This indicates the amount of data the server is sending to the device. When the client sends URB_COMPLETION_NO_DATA message to the server to report the final result of the TRANSFER_OUT_REQUEST, the **OutputBufferSize** value MUST NOT be greater than the **OutputBufferSize** value in TRANSFER_OUT_REQUEST message.

3.3.5.3.8 Processing a Retract Device Message

The structure and fields of the RETRACT_DEVICE message are specified in section 2.2.6.9.

After receiving the RETRACT_DEVICE message, the client SHOULD terminate the dynamic channel and stop redirecting the physical USB device.

3.3.5.4 Request Completion Interface

3.3.5.4.1 IO Control Completion Message

The structure and fields of the IOCONTROL_COMPLETION message are specified in section 2.2.7.1.

The client MUST use the **RequestId** received in the corresponding IO_CONTROL or INTERNAL_IO_CONTROL message when constructing this reply. The result of the IO Control or Internal IO Control operation performed, along with all data read, MUST be returned in the IOCONTROL_COMPLETION message.

The client MUST send one and only one IOCONTROL_COMPLETION message with matching **RequestId** for each IO_CONTROL or Internal IO Control message it receives from the server.

If the physical device returns more data than the **OutputBufferSize** field specifies in the IO_CONTROL or INTERNAL_IO_CONTROL message, the client SHOULD terminate the dynamic virtual channel.

3.3.5.4.2 URB Completion Message

The structure and fields of the URB_COMPLETION message are specified in section 2.2.7.2.

The client MUST use the **RequestId** received in the corresponding TRANSFER_IN_REQUEST message when constructing this reply. The result of the Transfer In operation performed, along with all data read, MUST be returned in the URB_COMPLETION message. If the Transfer In operation generated no data, the client MUST use URB_COMPLETION_NO_DATA message instead.

The client MUST send one and only one URB_COMPLETION message if data is generated, or one and only one URB_COMPLETION_NO_DATA message if no data is generated, for each TRANSFER_IN_REQUEST message it receives from the server.

If the physical device returns more data than the **OutputBufferSize** field specifies in the TRANSFER_IN_REQUEST message, the client SHOULD terminate the dynamic virtual channel.

3.3.5.4.3 URB Completion No Data Message

The structure and fields of the URB_COMPLETION_NO_DATA message are specified in section 2.2.7.3.

The client MUST use the **RequestId** received in the corresponding TRANSFER_IN_REQUEST message if the Transfer In operation generates no data or Transfer Out operation when constructing this reply. The result of the Transfer In operation that generates no data or Transfer Out operation performed MUST be returned in the URB_COMPLETION_NO_DATA message. If the Transfer In operation generates return data, the client MUST use the URB_COMPLETION message instead.

The client MUST send one and only one URB_COMPLETION message with matching **RequestId** value if data is generated, or one and only one URB_COMPLETION_NO_DATA message with matching **RequestId** value if no data is generated, for each TRANSFER_IN_REQUEST message it receives from the server.

The client MUST send one and only one URB_COMPLETION_NO_DATA message for each TRANSFER_OUT_REQUEST message it receives from the server.

3.3.5.5 Interface Manipulation Exchange Capabilities Interface Messages

3.3.5.5.1 Processing an Interface Manipulation Exchange Capabilities Request Message

The structure and fields of the RIM_EXCHANGE_CAPABILITY_REQUEST message are specified in section 2.2.3.1.

On receiving a RIM_EXCHANGE_CAPABILITY_REQUEST message, the client MUST send an RIM_EXCHANGE_CAPABILITY_RESPONSE message.

3.3.5.5.2 Sending an Interface Manipulation Exchange Capabilities Response Message

The structure and fields of the RIM_EXCHANGE_CAPABILITY_RESPONSE message are specified in section 2.2.3.2.

This message is sent in response to the RIM_EXCHANGE_CAPABILITY_REQUEST message.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

4 Protocol Examples

4.1 Server Data Interface Annotations

4.1.1 Channel Created Message

After a new channel is established, both the server and the client send the CHANNEL_CREATED message to each other. The message specifies the **MajorVersion**, **MinorVersion**, and **Capability** of the server for USB redirection. The following sequence shows the CHANNEL_CREATED message for a **MajorVersion** of 0x00000001, **MinorVersion** of 0x00000000, and **Capability** of 0x00000000.

```
Channel Created
ChannelName = URBDRC,24,server to client
00000000 02 00 00 40 00 00 00 00-00 01 00 00 01 00 00 00  ...@............
00000010 00 00 00 00 00 00 00 00-                         ................

02 00 00 40 -> Interface Id = 0x00000002 | mask STREAM_ID_PROXY (0x40000000)
00 00 00 00 -> Message Id = 0x00000000
00 01 00 00 -> CHANNEL_CREATED = 0x00000100
01 00 00 00 -> Major Version = 0x00000001
00 00 00 00 -> Minor Version = 0x00000000
00 00 00 00 -> Capability = 0x00000000

Channel Created
ChannelName = URBDRC,24,client to server
00000000 03 00 00 40 00 00 00 00-00 01 00 00 01 00 00 00  ...@............
00000010 00 00 00 00 00 00 00 00-                         ................

03 00 00 40 -> Interface Id = 0x00000003 | mask STREAM_ID_PROXY (0x40000000)
00 00 00 00 -> Message Id = 0x00000000
00 01 00 00 -> CHANNEL_CREATED = 0x00000100
01 00 00 00 -> Major Version = 0x00000001
00 00 00 00 -> Minor Version = 0x00000000
00 00 00 00 -> Capability = 0x00000000
```

4.1.2 Internal IO Control Message

The server sends the INTERNAL_IO_CONTROL message to the client in response to the request from the system specified in section 2.2.13. The INTERNAL_IO_CONTROL message described in this section is for IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME IO control code. There is no input parameter for this IO control code and the output parameter size is 0x00000004 bytes.

```
IO Control
ChannelName = URBDRC,0x1c,server to client
00000000 00 00 00 40 00 00 00 00-03 01 00 00 00 40 22 00
00000010 00 00 00 00 04 00 00 00-00 00 00 00

00 00 00 40 -> USB Device Interface Id = 0x00000000 | mask STREAM_ID_PROXY (0x40000000)
00 00 00 00 -> Message Id = 0x00000000
03 01 00 00 -> INTERNAL_IO_CONTROL = 0x00000103
00 40 22 00 -> IO control code = 0x00224000 (IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME)
00 00 00 00 -> Input Buffer Size = 0x00000000
04 00 00 00 -> Output Buffer Size = 0x00000004
```

```
00 00 00 00 -> Request Id = 0x00000000
```

4.1.3 IO Control Completion Message

In response to the INTERNAL_IO_CONTROL message described in section 4.1.2, the client sends the IOCONTROL_COMPLETION message (section 2.2.7.1) to the server containing the result returned from the physical device.

```
IO Control Completion
ChannelName = URBDRC,0x20,client to server
00000000 00 00 00 40 00 00 00 00-00 01 00 00 00 00 00 00
00000010 00 00 00 00 04 00 00 00-04 00 00 00 53 4b 5f 1a
00 00 00 40 -> RequestCompletion Interface Id = 0x00000000 | mask STREAM_ID_PROXY
(0x40000000)
00 00 00 00 -> Message Id = 0x00000000
00 01 00 00 -> IO_CONTROL_COMPLETION = 0x00000100
00 00 00 00 -> Request Id = 0x00000000 (from Internal IO Control message)
00 00 00 00 -> HResult = 0x00000000
04 00 00 00 -> Information = 0x00000004
04 00 00 00 -> Output Buffer Size = 0x00000004
53 4b 5f 1a -> Output Buffer Data = 0x1a5f4b53 (Current Frame)
```

4.1.4 Transfer In Request Message

The server sends the TRANSFER_IN_REQUEST message to the client in response to the request from the system specified in section 2.2.9. The TRANSFER_IN_REQUEST described in this section is for URB function URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER that reads 0x32 bytes from the physical device.

```
Transfer In
ChannelName = URBDRC,0x24,server to client
00000000 00 00 00 40 00 00 00 00-05 01 00 00 10 00 00 00
00000010 10 00 09 00 02 00 00 00-02 00 ff ff 03 00 00 00
00000020 32 00 00 00

00 00 00 40 -> USB Device Interface Id = 0x00000000 | mask STREAM_ID_PROXY (0x40000000)
00 00 00 00 -> Message Id = 0x00000000
05 01 00 00 -> TRANSFER_IN_REQUEST = 0x00000105
10 00 00 00 -> TS_URB size = 0x00000010
10 00       -> TS_URB CbSize = 0x0010
09 00       -> TS_URB Function = 0x0009 (TS_URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER)
02 00 00 00 -> TS_URB Request Id = 0x00000002
02 00 ff ff -> TS_URB PipeHandle = 0xffff0002
03 00 00 00 -> TS_URB TransferFlag = 0x00000003
32 00 00 00 -> Output Buffer Size = 0x00000032
```

4.1.5 URB Completion Message

In response to the TRANSFER_IN_REQUEST message described in section 4.1.3, the client sends the URB_COMPLETION message to the server containing the result returned from the physical device.

```
URB Completion
ChannelName = URBDRC,0x56,client to server
00000000 00 00 00 40 00 00 00 00-01 01 00 00 02 00 00 00
00000010 08 00 00 00 08 00 09 00-00 00 00 00 00 00 00 00
00000020 32 00 00 00 00 00 00 00-01 00 00 00 02 00 00 00
00000030 03 00 00 00 04 00 00 00-05 00 00 00 06 00 00 00
00000040 07 00 00 00 08 00 00 00-09 00 00 00 0a 00 00 00
00000050 0b 00 00 00 00 00

00 00 00 40 -> RequestCompletion Interface Id = 0x00000000 | mask STREAM_ID_PROXY
(0x40000000)
00 00 00 00 -> Message Id = 0x00000000
01 01 00 00 -> URB_COMPLETION = 0x00000101
02 00 00 00 -> Request Id = 0x00000002
08 00 00 00 -> TS_URB_RESULT Size = 0x0008
08 00       -> TS_URB_RESULT CbSize = 0x0008
09 00       -> filler
00 00 00 00 -> TS_URB_RESULT USBDStatus = 0x00000000
00 00 00 00 -> HResult = 0x00000000
32 00 00 00 -> Output Buffer Size = 0x00000032
00 00 00 00 -> Output Data
01 00 00 00
02 00 00 00
03 00 00 00
04 00 00 00
05 00 00 00
06 00 00 00
07 00 00 00
08 00 00 00
09 00 00 00
0a 00 00 00
0b 00 00 00
00 00
```

5 Security

5.1 Security Considerations for Implementers

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

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows® 7 operating system with Service Pack 1 (SP1)
- Windows Server 2008® R2 operating system with Service Pack 1 (SP1)

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.1: The server-side implementation of this protocol is applicable to Windows 7 SP1 and Windows Server 2008 R2 SP1. The client-side implementation of this protocol is applicable to Windows 7 SP1.

<2> Section 2.2.9.1.3: The field **MaximumTransferSize** of USBD_PIPE_INFORMATION is ignored.

7 Change Tracking

This section identifies changes that were made to the [MS-RDPEUSB] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.1 Glossary	61482 Added "American National Standards Institute (ANSI) character set" to the list of terms from [MS-GLOS].	N	Content updated.
1.2.1 Normative References	59135 Changed reference [USB-SPC] to [USB-SPC2.0].	Y	Content updated.
1.2.2 Informative References	61331 Added reference [MS-RDPEFS].	N	Content updated.
1.3 Protocol Overview (Synopsis)	61331 Updated the rules for USB device redirection by adding a list introduction, a list, and a closing paragraph.	N	Content updated.
1.3 Protocol Overview (Synopsis)	60019 Updated the sequence for adding and removing a device from the client.	Y	Content updated.
1.3.1.2 New Device Sequence	60298 Added information about the HardwareIds field.	Y	Content updated.
1.6 Applicability Statement	58638 Updated the description of allowed I/O	N	Content updated

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	operations by deleting "If it is," and adding "If there is any other I/O,".		.
1.6 Applicability Statement	60262 Clarified when the protocol is applicable.	Y	Content updated .
1.6 Applicability Statement	60267 Clarified why the protocol is not intended for devices requiring quality-of-service guarantees.	Y	Content updated .
1.6 Applicability Statement	60721 Updated constraints information by adding that the device will be treated as any other device attached to the client if there is any other I/O.	Y	Content updated .
1.7 Versioning and Capability Negotiation	60072 Updated for negotiation and versioning of the USB device by adding that the USB2.0 specification also includes versioning in the Device descriptor.	N	Content updated .
2.1 Transport	61482 Added the string encoding type of the dynamic virtual channel.	Y	Content updated .
2.2.1 Shared Message Header (SHARED_MSG_HEADER)	58345 Updated the InterfaceId field by changing the default value from 0x0000000 to 0.	Y	Content updated .
2.2.1 Shared Message Header (SHARED_MSG_HEADER)	58768 Changed "unsigned integer" to "field" in the description of the InterfaceID field.	N	Content updated .
2.2.1 Shared Message Header (SHARED_MSG_HEADER)	61483 Updated the description of the InterfaceId field by adding information about the highest two bits of the NetInterfaceId field.	N	Content updated .
2.2.6.2 Register Request Callback Message (REGISTER_REQUEST_CALLBACK)	59763 Included the Interface ID field.	Y	Content updated .
2.2.9 TS_URB Structures	58767 Revised description of the processing of the URB request by changing "it receives" to "received".	N	Editorially updated .
2.2.9.1.1 TS_URB_HEADER	58766 Revised the description of the URB	Y	Content updated

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	Function field by adding further description of the URB structure.		.
2.2.9.1.1 TS_URB_HEADER	58768 Updated the description of the RequestId and NoAck fields by deleting "unsigned integer" and inserting "field".	N	Content updated.
2.2.9.1.3 TS_USBD_PIPE_INFORMATION	58769 Added a product behavior note about ignoring the value of the MaximumTransferSize field.	Y	New product behavior note added.
2.2.9.2 TS_URB_SELECT_CONFIGURATION	59117 Updated the description of the URB_SELECT_CONFIGURATION by adding the phrase "URB structure".	N	Content updated.
2.2.9.3 TS_URB_SELECT_INTERFACE	59117 Updated the description of the URB_SELECT_INTERFACE by adding the phrase "URB structure".	N	Content updated.
2.2.9.4 TS_URB_PIPE_REQUEST	59117 Updated the description of the URB_PIPE_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.5 TS_URB_GET_CURRENT_FRAME_NUMBER	59117 Updated the description of the URB_GET_CURRENT_FRAME_NUMBER by adding the phrase "URB structure".	N	Content updated.
2.2.9.6 TS_URB_CONTROL_TRANSFER	59117 Updated the description of URB_CONTROL_TRANSFER by adding the phrase "URB structure".	N	Content updated.
2.2.9.7 TS_URB_BULK_OR_INTERRUPT_TRANSFER	59117 Updated the description of URB_BULK_OR_INTERRUPT_TRANSFER by adding the phrase "URB structure".	N	Content updated.
2.2.9.8 TS_URB_ISOCH_TRANSFER	59117 Updated the description of the URB_ISOCH_TRANSFER by adding the phrase "URB structure".	N	Content updated.
2.2.9.9 TS_URB_CONTROL_DESCRIPTOR_REQUEST	59117 Updated the description of URB_CONTROL_DESCRIPTOR_REQUEST by adding the phrase "URB structure".	N	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.9.10 TS_URB_CONTROL_FEATURE_REQUEST	59117 Updated the description of URB_CONTROL_FEATURE_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.11 TS_URB_CONTROL_GET_STATUS_REQUEST	59117 Updated the description of URB_CONTROL_GET_STATUS_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.12 TS_URB_CONTROL_VENDOR_OR_CLASS_REQUEST	59117 Updated the description of the URB_CONTROL_VENDOR_OR_CLASS_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.13 TS_URB_CONTROL_GET_CONFIGURATION_REQUEST	59117 Updated the description of URB_CONTROL_GET_CONFIGURATION_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.14 TS_URB_CONTROL_GET_INTERFACE_REQUEST	59117 Updated the description of URB_CONTROL_GET_INTERFACE_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.15 TS_URB_OS_FEATURE_DESCRIPTOR_REQUEST	58768 Updated the description of the Recipient and Padding1 fields by changing "unsigned integer" to "field".	N	Content updated.
2.2.9.15 TS_URB_OS_FEATURE_DESCRIPTOR_REQUEST	59117 Updated the description of the URB_OS_FEATURE_DESCRIPTOR_REQUEST by adding the phrase "URB structure".	N	Content updated.
2.2.9.15 TS_URB_OS_FEATURE_DESCRIPTOR_REQUEST	58986 Updated the description of the Recipient field by adding information about setting the field in URB_OS_FEATURE_DESCRIPTOR_REQUEST.	Y	Content updated.
2.2.9.15 TS_URB_OS_FEATURE_DESCRIPTOR_REQUEST	58988 Updated the description of the Recipient field.	Y	Content updated.
2.2.9.16 TS_URB_CONTROL_TRANSFER_EX	58972 Updated the description of the Timeout field by adding that the value is passed	N	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	to the physical device.		
2.2.9.16 TS_URB_CONTROL_TRANSFER_EX	59117 Updated the description of the URB_SELECT_CONFIGURATION by adding the phrase "URB structure".	N	Content updated.
2.2.10.1.1 TS_URB_RESULT_HEADER	59432 Revised the description of the UsbdStatus field by stating that the value described represents the Status field of the URB_STATUS structure.	Y	Content updated.
2.2.11 USB_DEVICE_CAPABILITIES	59434 Revised the description of the NoAckIsochWriteJitterBufferSizeInMs field.	Y	Content updated.
2.2.12 USB IO Control Code	59748 Updated content about I/O requests by clarifying that the IO_CONTROL messages are sent for each I/O request that the device driver sends to the USB device.	Y	Content updated.
2.2.13.1 IOCTL_TSUSBGD_IOCTL_USBDI_QUERY_BUS_TIME	59135 Changed reference [USB-SPC] to [USB-SPC2.0].	Y	Content updated.
3.1 Common Details	60015 Added a state diagram for the server.	Y	Content updated.
3.1 Common Details	60016 Added a state diagram for the client.	Y	Content updated.
3.1.1 Abstract Data Model	59766 Added the abstract data model element list of pending URB requests.	Y	Content updated.
3.1.1 Abstract Data Model	61329 Added the abstract data model element PipeHandle.	Y	Content updated.
3.1.1.1 Interface Manipulation Data Model	59762 Listed the fields that the interface manipulation applies to.	Y	Content updated.
3.1.5.2 Interface Manipulation	62166 Added prescription for releasing any interface, including the default interface.	N	Content updated.
3.2.5.1.2	60003	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Processing a Add Device Message	Updated the use of normative language regarding the processing of duplicate interface ID values.		updated.
3.2.5.2.2 Processing a Channel Created Message	60004 Updated the use of normative language regarding the processing of USB redirection version values.	Y	Content updated.
3.2.5.3.2 Sending a Register Request Callback Message	60005 Updated the use of normative language regarding when the server sends the REGISTER_REQUEST_CALLBACK message.	Y	Content updated.
3.2.5.3.6 Processing a Query Device Text Response Message	59764 Updated content about returning information to the application from the query by clarifying what the server MUST return.	Y	Content updated.
3.2.5.3.6 Processing a Query Device Text Response Message	59784 Revised DeviceDescription field information.	Y	Content updated.
3.2.5.3.7 Sending a Transfer In Request Message	59766 Updated the processing rules for the list of pending URB requests abstract data model element.	Y	Content updated.
3.2.5.3.8 Sending a Transfer Out Request Message	59771 Updated the processing rules for the list of pending URB requests abstract data model element.	Y	Content updated.
3.2.5.4.1 IO Control Completion Message	59785 Updated the server processing rules for the IOCONTROL_COMPLETION message to include where associated information is stored after the TRANSFER_IN_REQUEST message is sent.	Y	Content updated.
3.2.5.4.2 URB Completion Message	59786 Updated the server processing rules for the URB_COMPLETION message to include where associated information is stored after the TRANSFER_IN_REQUEST message is sent.	Y	Content updated.
3.2.5.4.2 URB Completion Message	59766 Updated the processing rules for the list of pending URB requests abstract data model element.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.2.5.4.3 URB Completion No Data Message	59787 Updated the server processing rules for the URB_COMPLETION_NO_DATA message to include where associated information is stored after the TRANSFER_IN_REQUEST or TRANSFER_OUT_REQUEST is sent.	Y	Content updated.
3.2.5.4.3 URB Completion No Data Message	59768 Updated the processing rules for the list of pending URB requests abstract data model element.	Y	Content updated.
3.3.5.1.2 Sending a Add Device Message	59773 Included the information regarding InterfaceID.	Y	Content updated.
3.3.5.2.2 Processing a Channel Created Message	60006 Changed "will" to "MUST".	Y	Content updated.
3.3.5.3.1 Processing a Cancel Request Message	60007 Changed "will" and "can" to "MUST".	Y	Content updated.
3.3.5.3.2 Processing a Register Request Callback Message	59774 Listed the fields for InterfaceID value.	Y	Content updated.
3.3.5.3.2 Processing a Register Request Callback Message	59794 Revised the use of normative language.	Y	Content updated.
3.3.5.3.2 Processing a Register Request Callback Message	59939 Revised the use of normative language.	Y	Content updated.
3.3.5.3.3 Processing an IO Control Message	59438 Updated the processing rules for forwarding the IO_CONTROL message by describing how the client forwards the request to the physical device.	Y	Content updated.
3.3.5.3.3 Processing an IO Control Message	60008 Changed "forwards", "sends", and "will terminate" to "MUST forward", "MUST send", and "MUST terminate" respectively.	Y	Content updated.
3.3.5.3.4 Processing an Internal IO Control Message	59438 Updated the processing rules for forwarding the INTERNAL_IO_CONTROL message by providing a cross-reference.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.3.5.3.4 Processing an Internal IO Control Message	60009 Changed "forwards", "sends", and "will" to "MUST forward", "MUST send", and "MUST", respectively.	Y	Content updated.
3.3.5.3.6 Processing a Transfer In Request Message	60010 Changed "forwards", "sends", and "will" to "MUST forward", "MUST send", and "MUST", respectively.	Y	Content updated.
3.3.5.4.1 IO Control Completion Message	59747 Revised content about IOCONTROL_COMPLETION messages and IO_CONTROL by adding "with matching RequestId."	Y	Content updated.
3.3.5.4.3 URB Completion No Data Message	59964 Clarified the client processing of the URB Completion No Data message.	Y	Content updated.

8 Index

R

S

T

U

V