[MS-RDPERP]:
Remote Desktop Protocol:
Remote Programs Virtual Channel Extension

Revision Summary

Date	Revision History	Revision Class	Comments
03/02/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.0.1	Editorial	Revised and edited the technical content.
08/10/2007	1.0.2	Editorial	Revised and edited the technical content.
09/28/2007	1.0.3	Editorial	Revised and edited the technical content.
10/23/2007	2.0	Major	Added new normative references.
11/30/2007	2.1	Minor	Corrected some section numbering.
01/25/2008	2.1.1	Editorial	Revised and edited the technical content.
03/14/2008	3.0	Major	Updated and revised the technical content.
05/16/2008	3.0.1	Editorial	Revised and edited the technical content.
06/20/2008	4.0	Major	Updated and revised the technical content.
07/25/2008	4.0.1	Editorial	Revised and edited the technical content.
08/29/2008	4.0.2	Editorial	Revised and edited the technical content.
10/24/2008	4.0.3	Editorial	Revised and edited the technical content.
12/05/2008	5.0	Major	Updated and revised the technical content.
01/16/2009	5.0.1	Editorial	Revised and edited the technical content.
02/27/2009	5.0.2	Editorial	Revised and edited the technical content.
04/10/2009	5.1	Minor	Updated the technical content.
05/22/2009	6.0	Major	Updated and revised the technical content.
07/02/2009	6.0.1	Editorial	Revised and edited the technical content.
08/14/2009	6.0.2	Editorial	Revised and edited the technical content.
09/25/2009	6.1	Minor	Updated the technical content.
11/06/2009	6.1.1	Editorial	Revised and edited the technical content.
12/18/2009	7.0	Major	Updated and revised the technical content.
01/29/2010	8.0	Major	Updated and revised the technical content.
03/12/2010	8.0.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	9.0	Major	Updated and revised the technical content.
06/04/2010	10.0	Major	Updated and revised the technical content.
07/16/2010	11.0	Major	Significantly changed the technical content.
08/27/2010	11.1	Minor	Clarified the meaning of the technical content.
10/08/2010	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	12.0	Major	Significantly changed the technical content.
03/25/2011	13.0	Major	Significantly changed the technical content.
05/06/2011	13.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	13.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

Remote Programs, also known as **remote applications integrated locally (RAIL)**, is a Remote Desktop Protocol (RDP) feature (as specified in the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification) that presents a **remote application** (running remotely on a RAIL server) as a local user application (running on the RAIL client machine). RAIL extends the core RDP protocol to deliver this seamless windows experience.

1.1 Glossary

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

Application Desktop Toolbar
balloon tooltip
client area
Control menu
desktop switch
notification area
notification icon
remote application
screen coordinates
system command
System menu
taskbar
tooltip
Unicode
Unicode character
window coordinates
window menu
window visible region
z-order

The following terms are specific to this document:

RAIL notification icon: An icon placed in the **notification area** of the client machine by the **remote applications integrated locally (RAIL)** client.

RAIL window: A local client window that mimics a **remote application** window.

remote applications integrated locally (RAIL): A software component that enables remoting of individual windows and **notification icons**.

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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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".

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

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

[MS-RDPEGDI] Microsoft Corporation, "Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions".

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

1.2.2 Informative References

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

[MSDN-CREATEWINEX] Microsoft Corporation, "CreateWindowEx Function", http://msdn.microsoft.com/en-us/library/ms632680.aspx

[MSDN-HIGHCONTRAST] Microsoft Corporation, "HIGHCONTRAST", http://msdn.microsoft.com/en-us/library/ms695609.aspx

[MSDN-SHELLNOTIFY] Microsoft Corporation, "Shell_NotifyIcon Function", http://msdn.microsoft.com/en-us/library/bb762159.aspx

[MSDN-SysParamsInfo] Microsoft Corporation, "SystemParametersInfo Function", http://msdn.microsoft.com/en-us/library/ms724947(VS.85).aspx

[MSDN-VIRTUALSCR] Microsoft Corporation, "The Virtual Screen", http://msdn.microsoft.com/en-us/library/dd145136(VS.85).aspx

[MSDN-WINFEATURE] Microsoft Corporation, "Window Features", http://msdn.microsoft.com/en-us/library/ms632599.aspx

[MSDN-WINSTYLE] Microsoft Corporation, "Window Styles", http://msdn.microsoft.com/en-us/library/ms632600.aspx

1.3 Overview

Remote Programs, also known as remote applications integrated locally (RAIL), is an RDP feature (as specified in the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification) that presents a remote application (running remotely on a RAIL server) as a local user application (running on the RAIL client machine). RAIL extends the core RDP protocol to deliver this seamless windows experience. Support for RAIL is optional in RDP, and it is negotiated as part of the capability negotiation process.

The RAIL client, running on the user's local machine, creates one local window or **notification icon** for every window or notification icon running on the RAIL server. These local windows/icons, called

RAIL windows/icons, exactly mimic the appearance of their corresponding remote windows/icons, which are created by remote applications running on the RAIL server. All local user input to the RAIL windows/icons is captured by the RAIL client and redirected to the server. All display updates to the remote windows/icons on the RAIL server are captured by the server and redirected to the client

RAIL relies on the core RDP protocol for basic connection establishment, connection security, local input redirection to server, and drawing order updates from server to client (as specified in the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification). In addition, RAIL adds the following extensions to the RDP protocol:

- Extensions to the RDP core protocol to send drawing orders from the server to the client describing individual windows and notification icons. This enables the RAIL client to mimic their geometry in RAIL windows/icons.
- Virtual channel messages from client to server containing client information, system parameters information, and RAIL-specific commands, such as remote program launch.
- Virtual channel updates from server to client containing responses to client messages, server system parameters information, or information regarding other RAIL-specific features such as local move/resize (specified in section 1.3.2.5).
- Certain classes of user input are not directly received by the RAIL window/icon as keyboard or mouse input. Examples include right-clicking the window's taskbar icon; key combinations to minimize, maximize, or restore all windows; and all user interactions with notification icons. These interactions are posted to the RAIL window/icon as non-keyboard or non-mouse messages, and, hence, cannot be sent over the core RDP channel. The client sends these interactions to the server as RAIL Virtual Channel messages.

1.3.1 Relationship to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification

Remote applications integrated locally (RAIL) protocol messages travel over two separate RDP channels:

- Window information orders from server to client are encapsulated in Alternate Secondary Drawing Orders (as specified in [MS-RDPEGDI] section 2.2.2.2.1.3.1.1).
- All other RAIL-specific messages travel over a static virtual channel, called the RAIL virtual channel, that is created by the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting during connection establishment (as specified in [MS-RDPBCGR] sections 1.3.3 and 2.2.1).

1.3.2 Message Flows

1.3.2.1 RAIL Session Connection

RAIL connection establishment follows the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting connection establishment sequence (as specified in [MS-RDPBCGR] section 1.3.1.1). RAIL-specific information during connection establishment is outlined as follows:

- The client must create and initialize a static virtual channel to be used for RAIL protocol messages. Information regarding this channel is sent to the server in the Client MCS Connect Initial PDU with GCC Conference Create Request (as specified in [MS-RDPBCGR] section 2.2.1.3).
- The Client Info PDU (as specified in [MS-RDPBCGR] section 2.2.1.11) must indicate the client's request to establish a RAIL connection.

- The **Alternate Shell** field of the Client Info PDU, as specified in [MS-RDPBCGR] section 2.2.1.11, is NOT used to communicate the initial application started in the session. Instead, the initial application information is communicated to the server via the Client Execute PDU.
- If the server supports RAIL, the Demand Active PDU must contain the Remote Programs Capability Set and Window List Capability Set to indicate that it supports RAIL.
- The client must send corresponding Remote Programs Capability Set and Window Capability Set in the Confirm Active PDU.
- If, in the Demand Active PDU, the server does not indicate that it supports RAIL, the client requests a disconnection according to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting ([MS-RDPBCGR] section 1.3.1.4.1). Likewise, if the client does not indicate that it supports RAIL in the Confirm Active PDU, the server disconnects the client (see [MS-RDPBCGR] section 1.3.1.4.2).

After the RDP connection is established, a RAIL client and server exchange Handshake PDUs over the RAIL Virtual Channel to indicate that each is ready for data on the virtual channel.

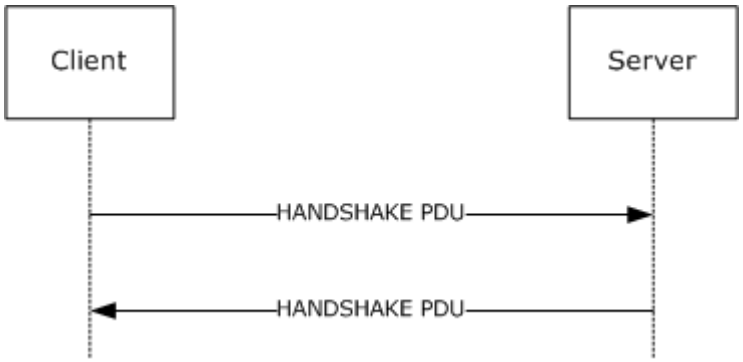


Figure 1: Handshake PDU

1.3.2.2 RAIL Session Disconnection and Reconnection

RAIL Session Disconnection and RAIL Session Reconnection follow the corresponding Remote Desktop Protocol: Basic Connectivity and Graphics Remoting sequences, as specified in [MS-RDPBCGR] section 1.3.1.4 (Disconnection Sequences) and [MS-RDPBCGR] section 1.3.1.5 (Automatic Reconnection).

1.3.2.3 RAIL Server/Client Synchronization

A RAIL server synchronizes with the RAIL client over the RDP channel upon connection establishment or when a **desktop switch** occurs.

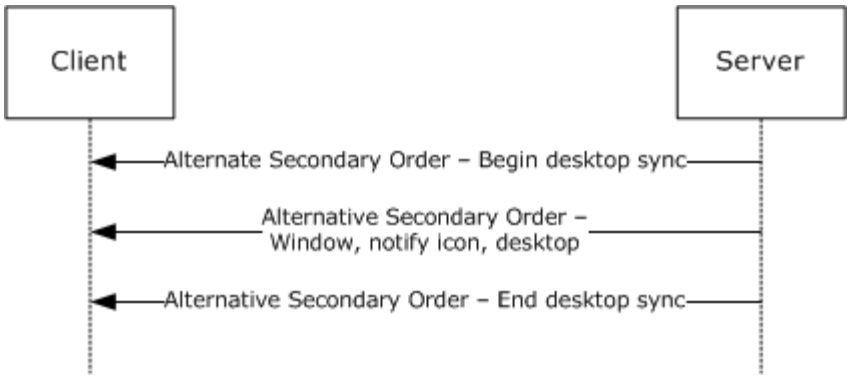


Figure 2: RAIL protocol client synchronization

The synchronization begins with a Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_ARC_BEGAN(0x00000008) flag set in the **Hdr** field (section 2.2.1.3.3.2.2). Upon receipt of this order, the client should clear all previously received information from the server. This order is followed by any number of Windowing Alternate Secondary Drawing Orders describing windows, notification icons, and desktop. Finally, the server sends a Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_ARC_COMPLETED (0x00000004) flag set to signal the end of synchronization data (section 2.2.1.3.3.2.1).

After the initial synchronization, Windowing Alternate Secondary Drawing Orders flow from server to client whenever a change occurs in a window, notification icon, or desktop state.

If the server is not capable of monitoring the desktop (for example, secure desktop), it sends a Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_NONE (0x00000001) flag set in the **Hdr** field (section 2.2.1.3.3.2.2). Upon receipt of this order, the client should clear out all previously received information from the server.

1.3.2.4 RAIL Virtual Channel Messages

Client/server or server/client messages can flow over the RAIL anytime after the virtual channel handshake sequence (section 2.2.2.2.1). The client should send the Client Information PDU and the Client System Parameters Update PDU immediately after the handshake to inform the server of its state and system parameters. The server should send the Server System Parameters Update PDU immediately after the handshake to inform the client of its system parameters. All other virtual channel messages are generated in response to events on the client or server.

1.3.2.5 RAIL Local Move/Resize

Local move/resize features are RAIL options designed to optimize bandwidth in certain situations where **RAIL windows** are moved or resized by the user. A RAIL client indicates to the RAIL server whether it supports local move/resize through the Client Capabilities PDU (section 2.2.2.2.2), sent after the Virtual Channel handshake sequence. RAIL servers do not have to explicitly report move/size support to the client.

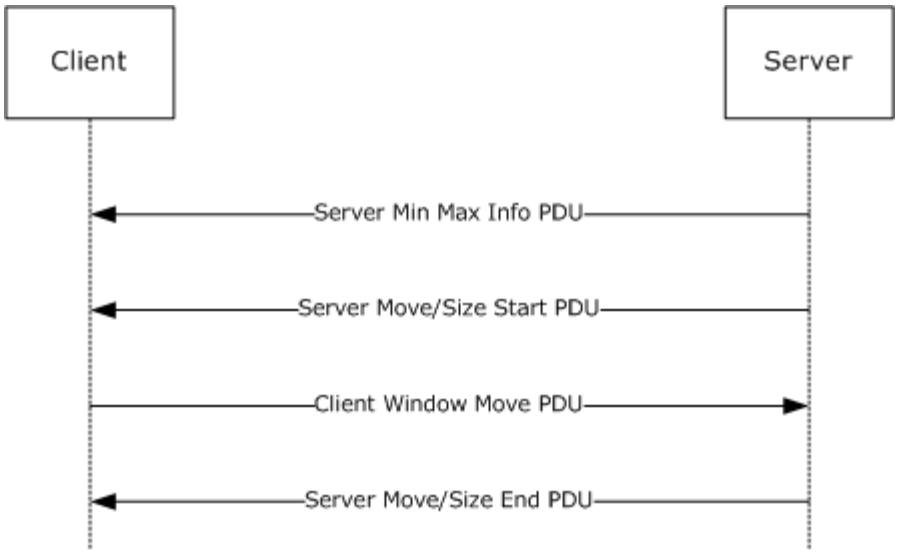


Figure 3: RAIL local move/resize operation

Local move/resize is based on the following logic:

1. When the server detects that a window is beginning to be moved or resized, it sends a Server Min Max Info PDU (section 2.2.2.7.1) to the client with the window extents. This is followed by a Server Move/Size Start PDU (section 2.2.2.7.2).

2. If the client supports local move/resize, it injects a mouse button-down at the position indicated by the move/size PDU (if the move/size was initiated via mouse) or posts a command to the window (if the move/size was initiated via keyboard) to initiate move/resize of the window by the local window manager.

3. At the same time, the client lets the local Window Manager handle all keyboard and mouse events for the RAIL window, instead of redirecting to the server, to ensure that the move/size is entirely happening locally.

4. Finally, when the user is done with the move/resize, the local RAIL window receives this notification and forwards a mouse button-up to the server to end move/size on the server. For keyboard-based moves and all resize operations, the client also sends a Client Window Move PDU (section 2.2.2.7.4) to the server to inform the server of the window's new position and size. (For mouse-based moves, the mouse button-up is sufficient to inform the window's final position).

5. When the server detects that move/size has ended, it sends a Server Move/Size End PDU (section 2.2.2.7.3) with the final window position and size. The client may adjust its local RAIL window if necessary using this information.

1.4 Relationship to Other Protocols

RAIL extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification.

1.5 Prerequisites/Preconditions

The Remote Programs Extensions for Remote Desktop Protocol has the assumption to operate in a fully operational RDP connection. A fully operational RDP connection is a connection that has passed the Connection Finalization phase, as specified in [MS-RDPBCGR] section 1.3.1.1.

1.6 Applicability Statement

The Remote Desktop Protocol: Remote Programs Virtual Channel Extension applies only to RDP 6.0 and later.

1.7 Versioning and Capability Negotiation

Versioning: RAIL is supported in RDP 6.0 and later clients only. The RDP version is negotiated as a part of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in section 1.7). Capability: RAIL-specific capabilities for Remote Programs and Window List are negotiated via the Demand Active and Confirm Active PDUs of the server and client, respectively (as specified in [MS-RDPBCGR], section 2.2.1.13).

1.8 Vendor-Extensible Fields

This protocol uses Win32 error codes as defined in [MS-ERREF] section 2.2. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

1.9 Standards Assignments

The Remote Desktop Protocol: Remote Programs Virtual Channel Extension does not use any assigned standards.

2 Messages

The following sections specify how Remote Desktop Protocol: Remote Programs Virtual Channel Extension messages are transported and Remote Desktop Protocol: Remote Programs Virtual Channel Extension message syntax.

This protocol references commonly used data types as defined in [MS-DTYP].

2.1 Transport

The Remote Desktop Protocol: Remote Programs Virtual Channel Extension messages are passed between the client and server, embedded within an RDP connection, as described in section 1.3.1 for an overview.

The protocol uses the TCP connection created by the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR] section 2.1) and does not establish any transport connections.

2.2 Message Syntax

2.2.1 Updates to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification

Support for RAIL is indicated by the client and server during the connection establishment phase of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, as described in section 1.3.2.1 for an overview of how the RAIL connection is established.

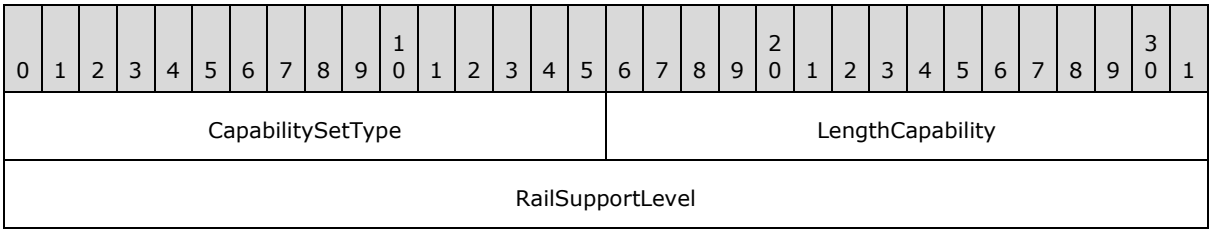
The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting has also been extended to support windowing-specific drawing orders for RAIL scenarios. These orders, called Windowing Alternate Secondary Drawing Orders, describe state for windows, notification icons, and desktop-related information on the server. The following sections outline the capability sets and drawing orders that make up the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting extensions for RAIL.

2.2.1.1 Capability Sets

A RAIL server and client indicate support for RAIL by exchanging two capability sets during the capabilities negotiation phase of RDP connection establishment. These sets are outlined in the following sections.

2.2.1.1.1 Remote Programs Capability Set

The Remote Programs Capability Set is sent by the server in the Demand Active PDU and by the client in the Confirm Active PDU, as specified in [MS-RDPBCGR] section 2.2.1.13. It indicates that the client and server are capable of communicating RAIL PDUs over the RAIL static virtual channel.



CapabilitySetType (2 bytes): An unsigned 16-bit integer. The type of the capability set. This field MUST be set to 0x0017 (CAPSETTYPE_RAIL).

LengthCapability (2 bytes): An unsigned 16-bit integer. The combined length of the **CapabilitySetType**, **LengthCapability**, and **RailSupportLevel** fields, in bytes.

RailSupportLevel (4 bytes): A 4 byte bitfield specifying support for Remote Programs and the Docked Language Bar for Remote Programs. <1>

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
S	L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Where the bits are defined as:

Value	Description
S TS_RAIL_LEVEL_SUPPORTED	Set to 1 if the client/server is capable of supporting Remote Programs; set to 0 otherwise.
L TS_RAIL_LEVEL_DOCKED_LANGBAR_SUPPORTED	Set to 1 if the client/server is capable of supporting Docked Language Bar for Remote Programs; set to 0 otherwise. This flag MUST be set to 0 if TS_RAIL_LEVEL_SUPPORTED is 0.

2.2.1.1.2 Window List Capability Set

The Window List Capability Set is sent by the server in the Demand Active PDU and by the client in the Confirm Active PDU, as specified in [MS-RDPBCGR] section 2.2.1.13. It indicates that the client and server are capable of communicating Windowing Alternate Secondary Drawing Orders as extensions to the core RDP protocol drawing orders (see section 2.2.1.3).

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
CapabilitySetType																LengthCapability															
WndSupportLevel																															
NumIconCaches										NumIconCacheEntries																					

CapabilitySetType (2 bytes): An unsigned 16-bit integer. The type of capability set. This field MUST be set to 0x0018 (CAPSETTYPE_WINDOW).

LengthCapability (2 bytes): An unsigned 16-bit integer. Specifies the combined length of the **CapabilitySetType**, **LengthCapability**, **WndSupportLevel**, **NumIconCaches**, and **NumIconCacheEntries** fields, in bytes.

WndSupportLevel (4 bytes): An unsigned 32-bit integer. The windowing support level. This field MUST be set to one of the following values. <2>

Value	Meaning
TS_WINDOW_LEVEL_NOT_SUPPORTED 0x00000000	The client or server is not capable of supporting Windowing Alternate Secondary Drawing Orders.
TS_WINDOW_LEVEL_SUPPORTED 0x00000001	The client or server is capable of supporting Windowing Alternate Secondary Drawing Orders.
TS_WINDOW_LEVEL_SUPPORTED_EX 0x00000002	The client or server is capable of supporting Windowing Alternate Secondary Drawing Orders and the following flags: ▪ WINDOW_ORDER_FIELD_CLIENTAREASIZE ▪ WINDOW_ORDER_FIELD_RPCONTENT ▪ WINDOW_ORDER_FIELD_ROOTPARENT

NumIconCaches (1 byte): An unsigned 8-bit integer. The number of icon caches requested by the server (Demand Active PDU) or supported by the client (Confirm Active PDU).

The server maintains an icon cache and refers to it to avoid sending duplicate icon information (see section 2.2.1.3.1.2.3). The client also maintains an icon cache and refers to it when the server sends across a Cached Icon Window Information Order.

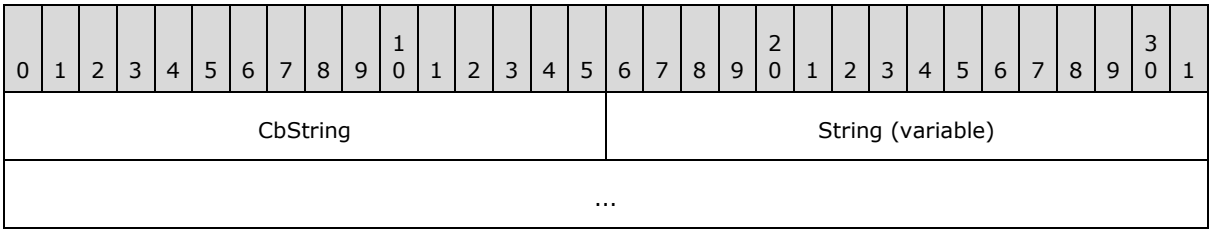
NumIconCacheEntries (2 bytes): An unsigned 16-bit integer. The number of entries within each icon cache requested by the server (Demand Active PDU) or supported by the client (Confirm Active PDU).

The server maintains an icon cache and refers to it to avoid sending duplicate icon information (see section 2.2.1.3.1.2.3). The client also maintains an icon cache and refers to it when the server sends across a Cached Icon Window Information Order.

2.2.1.2 Common Structures

2.2.1.2.1 Unicode String (UNICODE_STRING)

The UNICODE_STRING packet is used to pack a variable-length Unicode string.

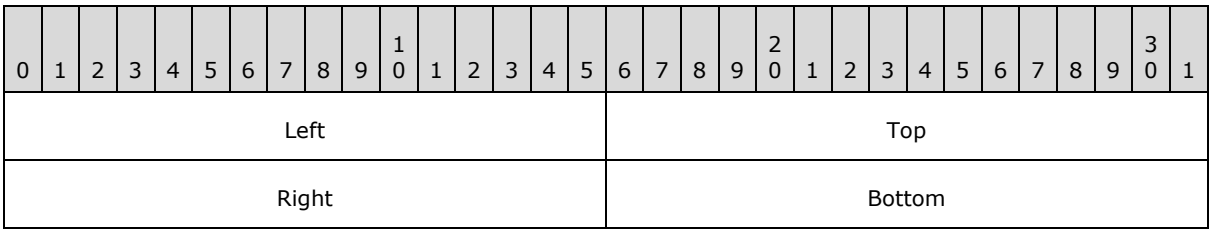


CbString (2 bytes): An unsigned 16-bit integer. The number of bytes in the **String** field. If CbString is zero (0), then the **String** field is absent. The maximum allowed value for **CbString** depends on the context in which the string is used.

String (variable): Optional and of variable length. A non-null-terminated Unicode character string. The number of characters in the string is equal to the value of **CbString** divided by 2.

2.2.1.2.2 Rectangle (TS_RECTANGLE_16)

The TS_RECTANGLE_16 structure describes a rectangle by using its top-left and bottom-right coordinates. The units depend on the context in which this structure is used.



Left (2 bytes): An unsigned 16-bit integer. The x-coordinate of the rectangle's top-left corner.

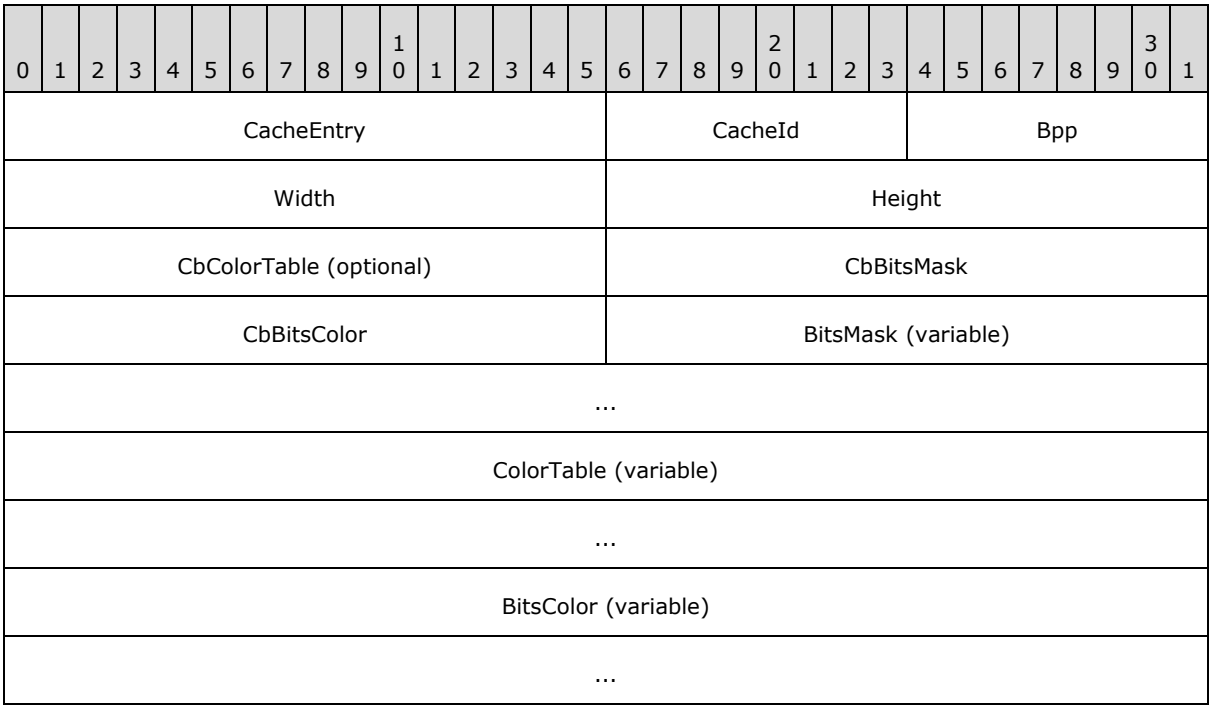
Top (2 bytes): An unsigned 16-bit integer. The y-coordinate of the rectangle's top-left corner.

Right (2 bytes): An unsigned 16-bit integer. The x-coordinate of the rectangle's bottom-right corner.

Bottom (2 bytes): An unsigned 16-bit integer. The y-coordinate of the rectangle's bottom-right corner.

2.2.1.2.3 Icon Info (TS_ICON_INFO)

The TS_ICON_INFO packet describes an icon.



CacheEntry (2 bytes): An unsigned 16-bit integer. The index within an icon cache at which this icon MUST be stored at the client. The index is unique within a given **CacheId** (see following description). The maximum value of **CacheEntry** is negotiated between server and client

through the **NumIconCacheEntries** field of the Window List Capability Set during the connection establishment phase.

CacheId (1 byte): An unsigned 8-bit integer. The index of the icon cache at which this icon MUST be stored at the client. If the value is 0xFFFF, the icon SHOULD NOT be cached. The **CacheId** is unique within a remote session.

The maximum value of **CacheId** is negotiated between server and client through the **NumIconCaches** field of the Window List Capability Set while establishing the connection.

Bpp (1 byte): An unsigned 8-bit integer. The color depth of the icon. Valid values are as follows:

1
4
8
16
24
32

Width (2 bytes): An unsigned 16-bit integer. The width, in pixels, of the icon.

Height (2 bytes): An unsigned 16-bit integer. The height, in pixels, of the icon.

CbColorTable (2 bytes): An unsigned 16-bit integer. The size, in bytes, of the color table data. This field is ONLY present if the **bits per pixel (Bpp)** value is 1, 4, or 8.

CbBitsMask (2 bytes): An unsigned 16-bit integer. The size, in bytes, of the icon's one-bit color-depth mask image.

CbBitsColor (2 bytes): An unsigned 16-bit integer. The size, in bytes, of the icon's color image.

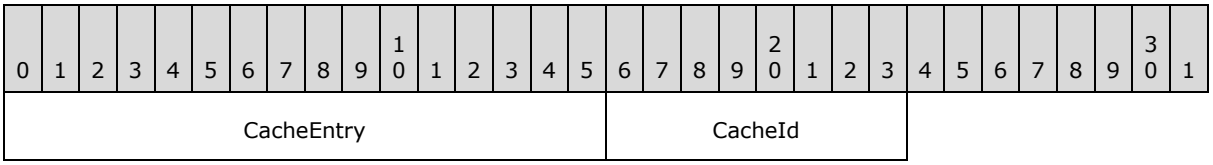
BitsMask (variable): The image data for the 1-bpp bitmap. The length, in bytes, of this field is equal to the value of **CbBitsMask**. This field is optional.

ColorTable (variable): The image data for the color bitmap. The length, in bytes, of this field is equal to the value of **CbColorTable**. This field is only present if the **Bpp** value is 1, 4, or 8.

BitsColor (variable): The image data for the icon's color image. The length, in bytes, of this field is equal to the value of **CbBitsColor**. This field is optional.

2.2.1.2.4 Cached Icon Info (TS_CACHED_ICON_INFO)

The TS_CACHED_ICON_INFO packet describes a cached icon.



CacheEntry (2 bytes): An unsigned 16-bit integer. The index within an icon cache at the client that refers to the cached icon. This value MUST have been previously specified by the server in the IconInfo structure (section 2.2.1.2.3) of a Window Information Order (section 2.2.1.3.1) or Icon structure of a New or Existing Notification Icon (section 2.2.1.3.2.2.1).

CacheId (1 byte): An unsigned 8-bit integer. The index of the icon cache containing the cached icon. This value MUST have been previously specified by the server in the IconInfo structure of a Window Information Order or Icon structure of a New or Existing Notification Icon.

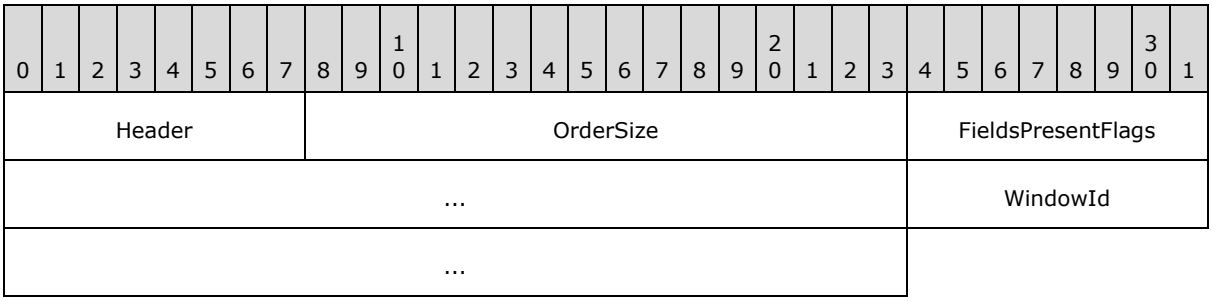
2.2.1.3 Windowing Alternate Secondary Drawing Orders

2.2.1.3.1 Window Information

Window Information Orders specify the state of windows on the server.

2.2.1.3.1.1 Common Header (TS_WINDOW_ORDER_HEADER)

The TS_WINDOW_ORDER_HEADER packet contains information common to every Windowing Alternate Secondary Drawing Order describing a window.



Header (1 byte): An unsigned 8-bit integer. An Alternate Secondary Order Header, as specified in [MS-RDPEGDI] section 2.2.2.2.1.3.1.1. The embedded **orderType** field MUST be set to 0x0B (TS_ALTSEC_WINDOW).

OrderSize (2 bytes): An unsigned 16-bit integer. The size of the entire packet, in bytes.

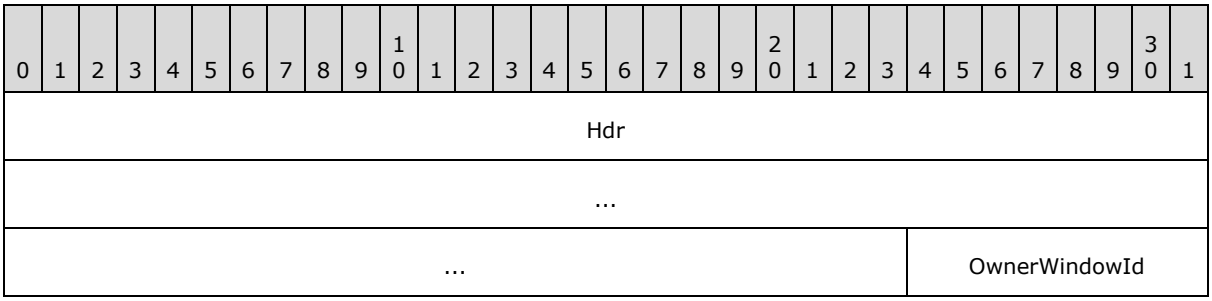
FieldsPresentFlags (4 bytes): An unsigned 32-bit integer. The flags indicating which fields are present in the packet. See Orders.

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window being described in the drawing order. It is generated by the server and is unique for every window in the session.

2.2.1.3.1.2 Orders

2.2.1.3.1.2.1 New or Existing Window

A Window Information Order is generated by the server whenever a new window is created on the server or when a property on a new or existing window is updated.



	(optional)
...	Style (optional)
...	ExtendedStyle (optional)
...	ShowState (optional)
TitleInfo (variable)	
...	
ClientOffsetX (optional)	
ClientOffsetY (optional)	
ClientAreaWidth (optional)	
ClientAreaHeight (optional)	
RPContent (optional)	RootParentHandle (optional)
...	WindowOffsetX (optional)
...	WindowOffsetY (optional)
...	WindowClientDeltaX (optional)
...	WindowClientDeltaY (optional)
...	WindowWidth (optional)
...	WindowHeight (optional)
...	NumWindowRects (optional) / WindowRects (variable)
...	
VisibleOffsetX (optional)	
VisibleOffsetY (optional)	
NumVisibilityRects (optional)	VisibilityRects (variable)
...	

Hdr (11 bytes): Eleven bytes. Common Window AltSec Order header, TS_WINDOW_ORDER_HEADER. The **FieldsPresentFlags** field of the header MUST conform to the values defined as follows.

Value	Meaning
0x01000000 WINDOW_ORDER_TYPE_WINDOW	Indicates a Windowing Alternate Secondary Drawing Order describing a window. This flag MUST be set.
0x10000000 WINDOW_ORDER_STATE_NEW	Indicates that the Windowing Alternate Secondary Drawing Order contains information for a new window. If this flag is not set, the order contains information for an existing window.
0x00000002 WINDOW_ORDER_FIELD_OWNER	Indicates that the **OwnerWindowId** field is present.
0x00000008 WINDOW_ORDER_FIELD_STYLE	Indicates that the **Style** and **ExtendedStyle** fields are present.
0x00000010 WINDOW_ORDER_FIELD_SHOW	Indicates that the **ShowState** field is present.
0x00000004 WINDOW_ORDER_FIELD_TITLE	Indicates that the **TitleInfo** field is present.
0x00004000 WINDOW_ORDER_FIELD_CLIENTAREAOFFSET	Indicates that the **ClientOffsetX** and **ClientOffsetY** fields are present.
0x00010000 WINDOW_ORDER_FIELD_CLIENTAREASIZE	Indicates that the **ClientAreaWidth** and **ClientAreaHeight** fields are present.<3>
0x00020000 WINDOW_ORDER_FIELD_RPCONTENT	Indicates that the **RPContent** field is present. <4>
0x00040000 WINDOW_ORDER_FIELD_ROOTPARENT	Indicates that the **RootParentHandle** field is present. <5>
0x00000800 WINDOW_ORDER_FIELD_WNDOFFSET	Indicates that the **WindowOffsetX** and **WindowOffsetY** fields are present.
0x00008000 WINDOW_ORDER_FIELD_WNDCLIENTDELTA	Indicates that the **WindowClientDeltaX** and **WindowClientDeltaY** fields are present.
0x00000400 WINDOW_ORDER_FIELD_WNDSIZE	Indicates that the **WindowWidth** and **WindowHeight** fields are present.
0x00000100 WINDOW_ORDER_FIELD_WNDRECTS	Indicates that the **NumWindowRects** and **WindowRects** fields are present.
0x00001000 WINDOW_ORDER_FIELD_VISOFFSET	Indicates that the **VisibleOffsetX** and **VisibleOffsetY** fields are present.
0x00000200	Indicates that the **NumVisibilityRects** and **VisibilityRects** fields are present.

Value	Meaning
WINDOW_ORDER_FIELD_VISIBILITY	

OwnerWindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server that is the owner of the window specified in WindowId field of **Hdr**. For more information on owned windows, see [MSDN-WINFEATURE]. This field is present if and only if the WINDOW_ORDER_FIELD_OWNER flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

Style (4 bytes): An unsigned 32-bit integer. Describes the window's current style. Window styles determine the appearance and behavior of a window. For more information, see [MSDN-WINSTYLE]. This field is present if and only if the WINDOW_ORDER_FIELD_STYLE flag is set in the **FieldsPresentFlags** field of the TS_WINDOW_ORDER_HEADER.

ExtendedStyle (4 bytes): An unsigned 32-bit integer. Extended window style information. For more information about extended window styles, see [MSDN-CREATEWINEX].

This field is present if and only if the WINDOW_ORDER_FIELD_STYLE flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

ShowState (1 byte): An unsigned 8-bit integer. Describes the show state of the window.

This field is present if and only if the WINDOW_ORDER_FIELD_SHOW flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

The field MUST be one of the following values.

Value	Meaning
0x00	Do not show the window.
0x02	Show the window minimized.
0x03	Show the window maximized.
0x05	Show the window in its current size and position.

TitleInfo (variable): UNICODE_STRING. Variable length. Contains the window's title string. The maximum value for the **CbString** field of UNICODE_STRING is 520 bytes. This structure is present only if the WINDOW_ORDER_FIELD_TITLE flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

ClientOffsetX (4 bytes): An unsigned 32-bit integer. The X (horizontal) offset from the top-left corner of the screen to the top-left corner of the window's **client area**, expressed in **screen coordinates**.

This field is present only if the WINDOW_ORDER_FIELD_CLIENTAREAOFFSET flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

ClientOffsetY (4 bytes): An unsigned 32-bit integer. The Y (vertical) offset from the top-left corner of the screen to the top-left corner of the window's client area, expressed in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_CLIENTAREAOFFSET flag is set in the FieldsPresentFlags field of TS_WINDOW_ORDER_HEADER.

ClientAreaWidth (4 bytes): An unsigned 32-bit integer specifying the width of the client area rectangle of the target window.

This field only appears if the **WndSupportLevel** field of the Window List Capability Set message is set to TS_WINDOW_LEVEL_SUPPORTED_EX (as specified in section 2.2.1.1.2) and the WINDOW_ORDER_FIELD_CLIENTAREASIZE flag is set in the **FieldsPresentFlags** field of the TS_WINDOW_ORDER_HEADER packet (section 2.2.1.3.1.1).

ClientAreaHeight (4 bytes): An unsigned 32-bit integer specifying the height of the client area rectangle of the target window.

This field only appears if the **WndSupportLevel** field of the Window List Capability Set message is set to TS_WINDOW_LEVEL_SUPPORTED_EX (as specified in section 2.2.1.1.2) and the **Hdr** field has the WINDOW_ORDER_FIELD_CLIENTAREASIZE flag is set in the **FieldsPresentFlags** field of the TS_WINDOW_ORDER_HEADER packet (section 2.2.1.3.1.1).

RPContent (1 byte): An unsigned BYTE that MUST be set to one of the following possible values.

Value	Meaning
0x00	The window is not used by a render plug-in to do client-side rendering.
0x01	The window is used by a render plug-in to do client-side rendering.

This field only appears if the **WndSupportLevel** field of the Window List Capability Set message is set to TS_WINDOW_LEVEL_SUPPORTED_EX (as specified in section 2.2.1.1.2) and the **Hdr** field has the WINDOW_ORDER_FIELD_RPCONTENT flag is set in the **FieldsPresentFlags** field of the TS_WINDOW_ORDER_HEADER packet (section 2.2.1.3.1.1).

RootParentHandle (4 bytes): An unsigned 32-bit integer specifying the server-side target window's top-level parent window handle. A Top-Level parent window is the window immediately below "desktop" in the window hierarchy. If the target window is a top-level window, the window handle of the target window is sent.

This field only appears if the **WndSupportLevel** field of the Window List Capability Set message is set to TS_WINDOW_LEVEL_SUPPORTED_EX (as specified in section 2.2.1.1.2) and the **Hdr** field has the WINDOW_ORDER_FIELD_ROOTPARENT flag is set in the **FieldsPresentFlags** field of the TS_WINDOW_ORDER_HEADER packet (section 2.2.1.3.1.1).

WindowOffsetX (4 bytes): An unsigned 32-bit integer. The X (horizontal) offset from the top-left corner of the window to the top-left corner of the window's client area, expressed in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_WNDOFFSET flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowOffsetY (4 bytes): An unsigned 32-bit integer. The Y (vertical) offset from the top-left corner of the window to the top-left corner of the window's client area, expressed in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_WNDOFFSET flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowClientDeltaX (4 bytes): An unsigned 32-bit integer. The X (horizontal) delta between the top-left corner of the window and the window's client area.

This field is present only if the WINDOW_ORDER_FIELD_CLIENTDELTA flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowClientDeltaY (4 bytes): An unsigned 32-bit integer. The Y (vertical) delta between the top-left corner of the window and the window's client area.

This field is present only if the WINDOW_ORDER_FIELD_CLIENTDELTA flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowWidth (4 bytes): An unsigned 32-bit integer. The window width, in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_WNDSIZE flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowHeight (4 bytes): An unsigned 32-bit integer. The window height, in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_WNDSIZE flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

NumWindowRects (2 bytes): An unsigned 16-bit integer. A count of rectangles describing the window geometry.

This field is present only if the WINDOW_ORDER_FIELD_WNDRECTS flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

WindowRects (variable): An array of TS_RECTANGLE_16 structures, **NumWindowRects** wide, describing the window geometry. All coordinates are **window coordinates**.

This field is present only if the **NumWindowRects** field is greater than 0 and the WINDOW_ORDER_FIELD_WNDRECTS flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

VisibleOffsetX (4 bytes): An unsigned 32-bit integer. The X (horizontal) offset from the top-left corner of the screen to the top-left corner of the **window's visible region's** bounding rectangle, expressed in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_VISOFFSET flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

VisibleOffsetY (4 bytes): An unsigned 32-bit integer. The Y (vertical) offset from the top-left corner of the screen to the top-left corner of the window's visible region's bounding rectangle, expressed in screen coordinates.

This field is present only if the WINDOW_ORDER_FIELD_VISOFFSET flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

NumVisibilityRects (2 bytes): An unsigned 16-bit integer. A count of rectangles describing the window's visible region.

This field is present only if the WINDOW_ORDER_FIELD_VISIBILITY flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

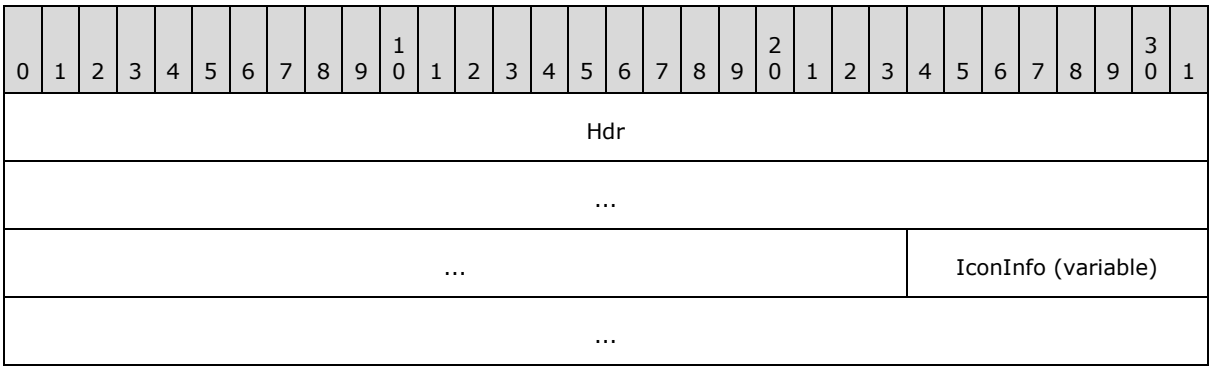
VisibilityRects (variable): An array of TS_RECTANGLE_16 structures, **NumVisibilityRects** wide, describing the window's visible region. All coordinates are window coordinates.

This field is present only if the value of the **NumVisibilityRects** field is greater than 0 and the WINDOW_ORDER_FIELD_VISIBILITY flag is set in the **FieldsPresentFlags** field of TS_WINDOW_ORDER_HEADER.

2.2.1.3.1.2.2 Window Icon

The Window Icon packet is a Window Information Order generated by the server when a new or existing window sets or updates its associated icon.

Icons are created by combining two bitmaps of the same size. The mask bitmap is always 1 bpp, although the color depth of the color bitmap can vary. The color bitmap may have an associated color table.



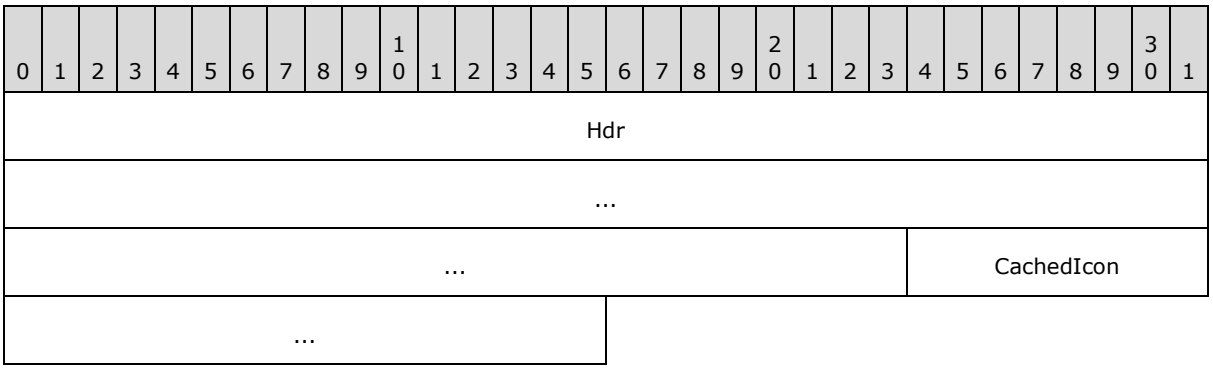
Hdr (11 bytes): Eleven bytes. A TS_WINDOW_ORDER_HEADER structure. The **FieldsPresentFlags** field of the header MUST be constructed using the following values.

Value	Meaning
0x01000000 WINDOW_ORDER_TYPE_WINDOW	Indicates a Windowing Alternate Secondary Drawing Order that describes a window. This flag MUST be set.
0x10000000 WINDOW_ORDER_STATE_NEW	Indicates that the Windowing Alternate Secondary Drawing Order contains information for a new window. If this flag is not set, the order contains information for an existing window.
0x40000000 WINDOW_ORDER_ICON	Indicates that the order contains icon information for the window. This flag MUST be set.
0x00002000 WINDOW_ORDER_FIELD_ICON_BIG	Indicates that the large version of the icon is being sent. If this flag is not present, the icon is a small icon. <6>

IconInfo (variable): Variable length. TS_ICON_INFO structure. Describes the window's icon.

2.2.1.3.1.2.3 Cached Icon

The Cached Icon Window Information Order is generated by the server when a new or existing window sets or updates the icon in its title bar or in the Alt-Tab dialog box. If the icon information was transmitted by the server in a previous Window Information Order or Notification Icon Information Order in the same session, and the icon was cacheable (that is, the server specified a cacheEntry and cacheId for the icon), the server reports the icon cache entries to avoid sending duplicate information.



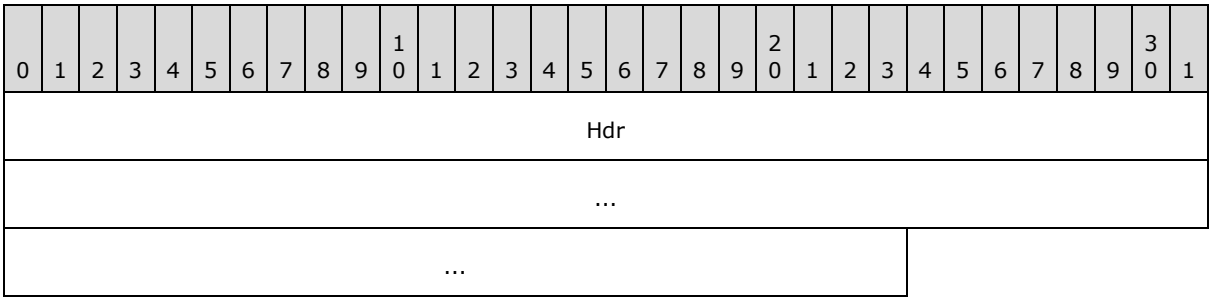
Hdr (11 bytes): Eleven bytes. A TS_WINDOW_ORDER_HEADER structure. The **FieldsPresentFlags** field of the header MUST conform to the values defined as follows.

Value	Description
WINDOW_ORDER_TYPE_WINDOW 0x01000000	Indicates a Windowing Alternate Secondary Drawing Order that describes a window. This flag MUST be set.
WINDOW_ORDER_STATE_NEW 0x10000000	Indicates that the Windowing Alternate Secondary Drawing Order contains information for a new window. If this flag is not set, the order contains information for an existing window.
WINDOW_ORDER_CACHEDICON 0x80000000	Indicates that the order contains cached icon information for the window. This flag MUST be set.
WINDOW_ORDER_FIELD_ICON_BIG 0x00002000	Indicates that the large version of the icon is being referred to. If this flag is not present, the icon is a small icon. <7>

CachedIcon (3 bytes): Three bytes. TS_CACHED_ICON_INFO structure. Describes a cached icon on the client.

2.2.1.3.1.2.4 Deleted Window

The Deleted Window Information Order is generated by the server whenever an existing window is destroyed on the server.



Hdr (11 bytes): Eleven bytes. A TS_WINDOW_ORDER_HEADER structure. The **FieldsPresentFlags** field of the header MUST be constructed using the following values.

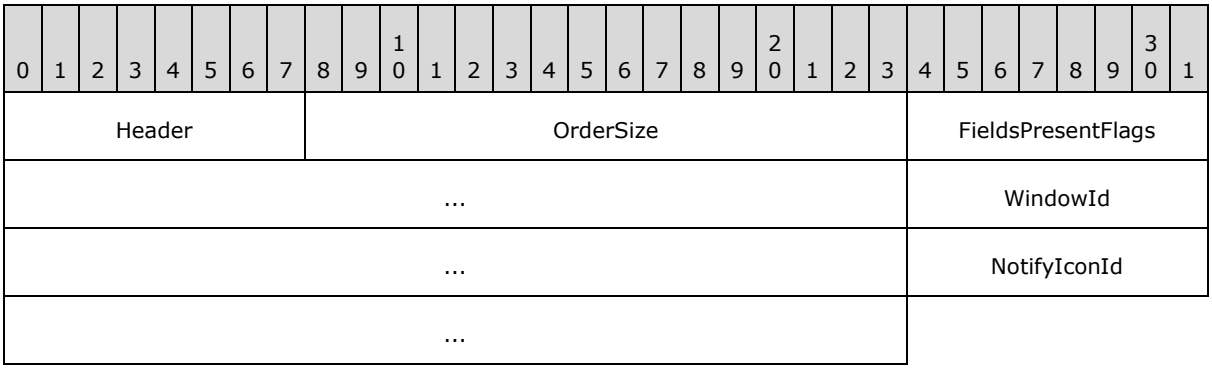
Value	Meaning
0x01000000 WINDOW_ORDER_TYPE_WINDOW	Indicates a Windowing Alternate Secondary Drawing Order describing a window. This flag MUST be set.
0x20000000 WINDOW_ORDER_STATE_DELETED	Indicates that the window is deleted. If this flag is set, the order MUST NOT contain any other information.

2.2.1.3.2 Notification Icon Information

Notification Icon Information orders specify the state of the notification icon on the server.

2.2.1.3.2.1 Common Header (TS_NOTIFYICON_ORDER_HEADER)

The TS_NOTIFYICON_ORDER_HEADER packet contains information common to every Windowing Alternate Secondary Drawing Order specifying a notification icon.



Header (1 byte): An unsigned 8-bit integer. An Alternate Secondary Order Header, as specified in [MS-RDPEGDI] section 2.2.2.2.1.3.1.1. The embedded **orderType** field MUST be set to 0x0B (TS_ALTSEC_WINDOW).

OrderSize (2 bytes): An unsigned 16-bit integer. The size, in bytes, of the entire packet.

FieldsPresentFlags (4 bytes): An unsigned 32-bit integer. The flags indicating which fields are present in the packet. See New or Existing Notification Icons.

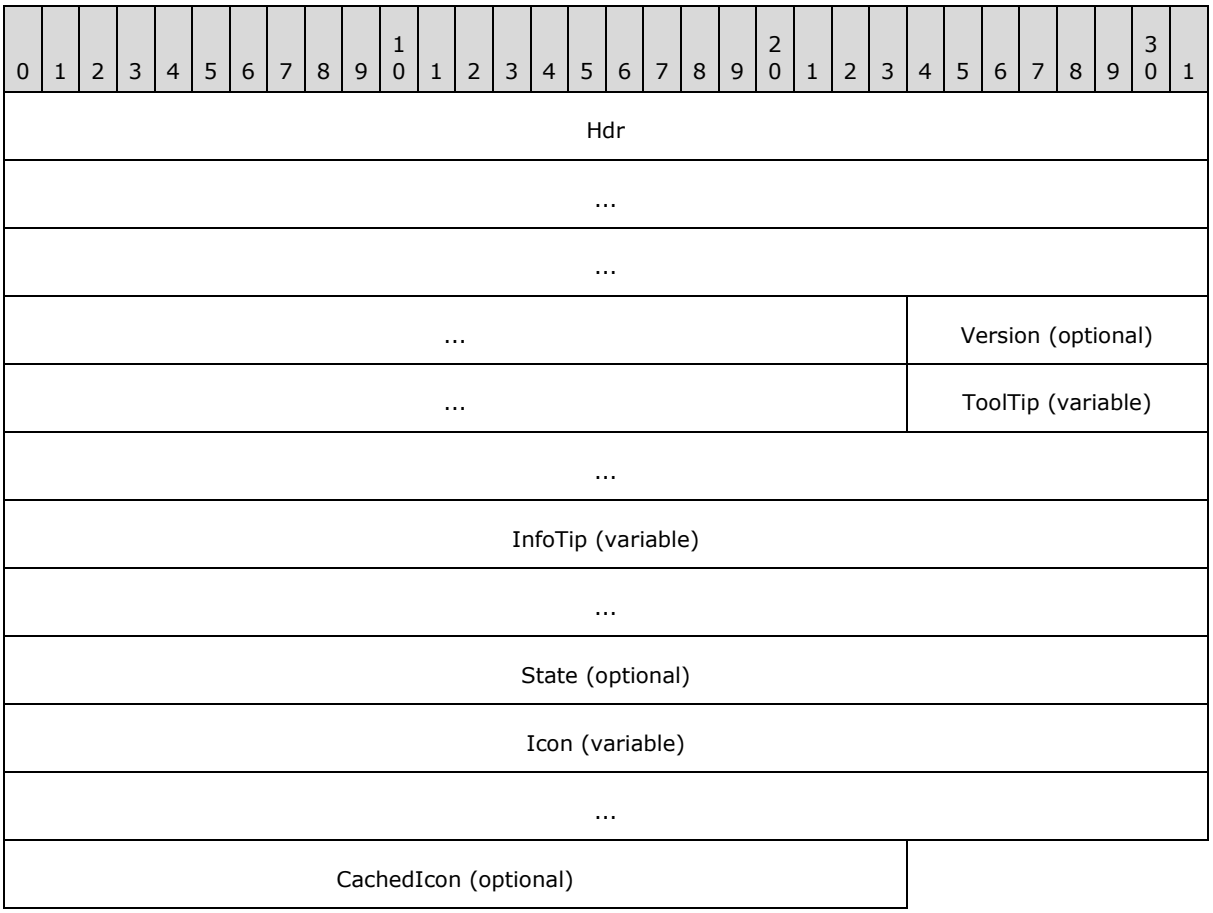
WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window owning the notification icon specified in the drawing order. The ID is generated by the server and is unique for every window in the session.

NotifyIconId (4 bytes): An unsigned 32-bit integer. The ID of the notification icon specified in the drawing order. The ID is generated by the application that owns the notification icon and SHOULD be unique for every notification icon owned by the application.

2.2.1.3.2.2 Orders

2.2.1.3.2.2.1 New or Existing Notification Icons

The Notification Icon Information Order packet is generated by the server whenever a new notification icon is created on the server or when an existing notification icon is updated.



Hdr (15 bytes): Fifteen bytes. A TS_NOTIFYICON_ORDER_HEADER structure. Common AltSec Order header. The **FieldsPresentFlags** field of the header MUST conform to the values defined as follows.

Value	Meaning
WINDOW_ORDER_TYPE_NOTIFY 0x02000000	Indicates a Windowing Alternate Secondary Drawing Order specifying a notification icon. This flag MUST be set.
WINDOW_ORDER_STATE_NEW 0x10000000	Indicates that the Windowing Alternate Secondary Drawing Order contains information for a new notification icon. If this flag is set, one of the **Icon** and **CachedIcon** fields MUST be present. If this flag is not set, the Windowing Alternate Secondary Drawing Order contains information for an existing notification icon.
WINDOW_ORDER_FIELD_NOTIFY_VERSION 0x00000008	Indicates that the **Version** field is present.
WINDOW_ORDER_FIELD_NOTIFY_TIP 0x00000001	Indicates that the **Tooltip** field is present.

Value	Meaning
WINDOW_ORDER_FIELD_NOTIFY_INFO_TIP 0x00000002	Indicates that the **InfoTip** field is present.
WINDOW_ORDER_FIELD_NOTIFY_STATE 0x00000004	Indicates that the **State** field is present.
WINDOW_ORDER_ICON 0x40000000	Indicates that the **Icon** field is present. Either the **Icon** or the **CachedIcon** field SHOULD be present, but not both.
WINDOW_ORDER_CACHED_ICON 0x80000000	Indicates that the **CachedIcon** field is present. Either the **Icon** or the **CachedIcon** field SHOULD be present, but not both. <8>

Version (4 bytes): An unsigned 32-bit integer. Specifies the behavior of the notification icons. This field is present only if the WINDOW_ORDER_FIELD_NOTIFY_VERSION flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER. This field MUST be set to one of the following values.

Value	Meaning
0	Use this value for applications designed for Windows NT 4.0.
3	Use the Windows 2000 notification icons behavior. Use this value for applications designed for Windows 2000 and Windows XP.
4	Use the current behavior. Use this value for applications designed for Windows Vista and Windows 7.

For more information about notification icons, see [MSDN-SHELLNOTIFY], the Remarks section.

ToolTip (variable): Variable length. UNICODE_STRING. Specifies the text of the notification icon **tooltip**. This structure is present only if the WINDOW_ORDER_FIELD_NOTIFY_TIP flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER.

InfoTip (variable): Variable length. A TS_NOTIFY_ICON_INFOTIP structure. Specifies the notify icon's **balloon tooltip**. This field SHOULD NOT be present for icons that follow Windows 95 behavior (Version = 0). This structure is present only if the WINDOW_ORDER_FIELD_NOTIFY_INFO_TIP flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER.

State (4 bytes): Unsigned 32-bit integer. Specifies the state of the notify icon. This field SHOULD NOT be present for icons that follow Windows 95 behavior (Version = 0).

This field is present only if the WINDOW_ORDER_FIELD_NOTIFY_STATE flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER.

Value	Meaning
1	The notify icon is hidden.

Icon (variable): Variable length. A TS_ICON_INFO structure. Specifies the notify icon's image. This structure is present only if the WINDOW_ORDER_ICON flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER.

A Notification Icon Order MUST NOT contain both an **Icon** field and a **CachedIcon** field. If the WINDOW_ORDER_STATE_NEW flag is set, either the **Icon** field or the **CachedIcon** field MUST be present.

CachedIcon (3 bytes): Three bytes. A TS_CACHED_ICON_INFO structure. Specifies the notify icon as a cached icon on the client.

This structure is present only if the WINDOW_ORDER_CACHEDICON flag is set in the **FieldsPresentFlags** field of TS_NOTIFYICON_ORDER_HEADER. Only one of **Icon** and **CachedIcon** fields SHOULD be present in the Notification Icon Order. If the WINDOW_ORDER_STATE_NEW flag is set, only one of **Icon** and **CachedIcon** fields MUST be present.

2.2.1.3.2.2.2 Deleted Notification Icons

The server generates a Notification Icon Information (section 2.2.1.3.2) order packet whenever an existing notification icon is deleted on the server.

Hdr (15 bytes): Fifteen bytes. A TS_NOTIFYICON_ORDER_HEADER (section 2.2.1.3.2.1) structure. The **FieldsPresentFlags** field of the header MUST be constructed using the following values.

Value	Meaning
0x02000000 WINDOW_ORDER_TYPE_NOTIFY	Indicates an order specifying a notification icon. This flag MUST be set.
0x20000000 WINDOW_ORDER_STATE_DELETED	Indicates that the window is deleted. This flag MUST be set, and the order MUST NOT contain any other information.

2.2.1.3.2.2.3 Notification Icon Balloon Tooltip (TS_NOTIFY_ICON_INFOTIP)

The TS_NOTIFY_ICON_INFOTIP structure specifies the balloon tooltip of a notification icon.

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
Timeout																															
InfoFlags																															
InfoTipText (variable)																															
...																															
Title (variable)																															
...																															

Timeout (4 bytes): An unsigned 32-bit integer. The timeout in milliseconds for the notify icon's balloon tooltip. After the specified timeout, the tooltip SHOULD be destroyed. <9>

InfoFlags (4 bytes): An unsigned 32-bit integer. The flags that can be set to add an icon to a balloon tooltip. It is placed to the left of the title. If the **InfoTipText** field length is zero-length, the icon is not shown.

Value	Meaning
NIIF_NONE 0x00000000	Do not show an icon.
NIIF_INFO 0x00000001	Show an informational icon next to the balloon tooltip text.
NIIF_WARNING 0x00000002	Show a warning icon next to the balloon tooltip text.
NIIF_ERROR 0x00000003	Show an error icon next to the balloon tooltip text.
NIIF_NOSOUND 0x00000010	Do not play an associated sound.
NIIF_LARGE_ICON 0x00000020	TShow the large version of the icon.

InfoTipText (variable): Variable length. A UNICODE_STRING specifying the text of the balloon tooltip. The maximum length of the tooltip text string is 510 bytes.

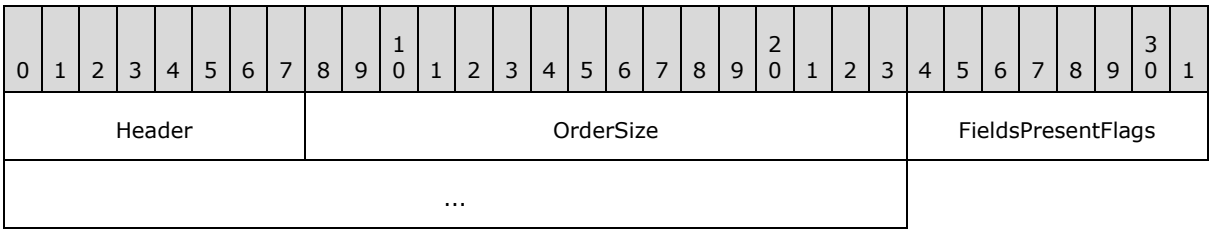
Title (variable): Variable length. A UNICODE_STRING specifying the title of the balloon tooltip. The maximum length of the tooltip title string is 126 bytes.

2.2.1.3.3 Desktop Information

Desktop Information Orders specify the state of the desktop on the server.

2.2.1.3.3.1 Common Header (TS_DESKTOP_ORDER_HEADER)

The TS_DESKTOP_ORDER_HEADER packet contains information common to every order specifying the desktop.



Header (1 byte): An unsigned 8-bit integer. An Alternate Secondary Order Header, as specified in [MS-RDPEGDI] section 2.2.2.2.1.3.1.1. The embedded **orderType** field MUST be set to 0x0B (TS_ALTSEC_WINDOW).

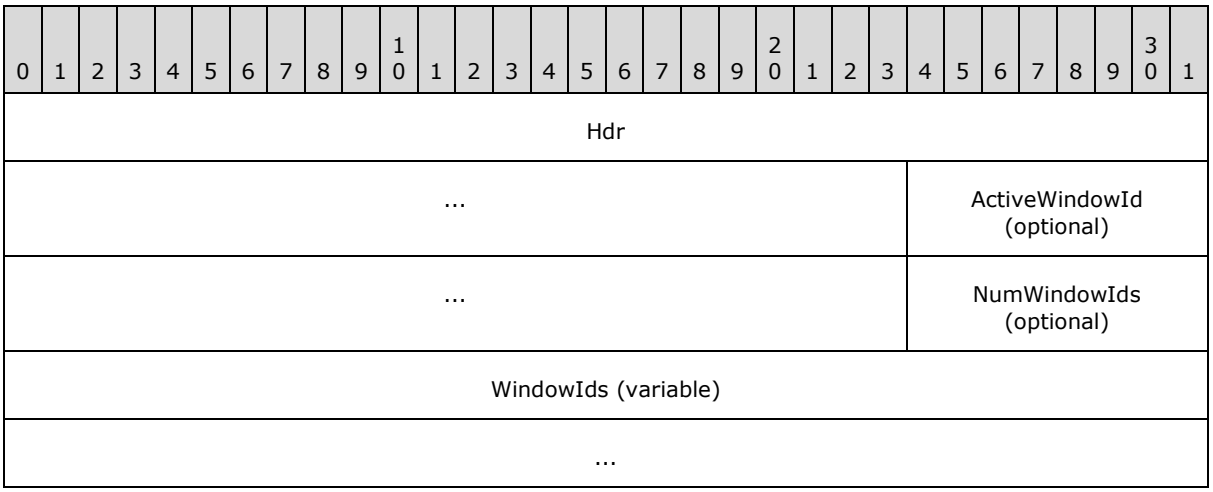
OrderSize (2 bytes): An unsigned 16-bit integer. The size of the entire packet in bytes.

FieldsPresentFlags (4 bytes): An unsigned 32-bit integer. The flags indicating which fields are present in the packet. See Actively Monitored Desktop for values and use.

2.2.1.3.3.2 Orders

2.2.1.3.3.2.1 Actively Monitored Desktop

The Actively Monitored Desktop packet contains information about the actively monitored desktop.



Hdr (7 bytes): Seven bytes. A TS_DESKTOP_ORDER_HEADER header. The **FieldsPresentFlags** field of the header MUST be constructed using the following values.

Value	Meaning
0x04000000 WINDOW_ORDER_TYPE_DESKTOP	Indicates an order specifying a desktop. This flag MUST be set.
0x00000002	Indicates that the server will be sending

Value	Meaning
WINDOW_ORDER_FIELD_DESKTOP_HOOKED	information for the server's current input desktop.
0x00000008 WINDOW_ORDER_FIELD_DESKTOP_ARC_BEGAN	Indicates that the server is beginning to synchronize information with the client after the client has auto-reconnected or the server has just begun monitoring a new desktop. If this flag is set, the WINDOW_ORDER_FIELD_DESKTOP_HOOKED flag MUST also be set.
0x00000004 WINDOW_ORDER_FIELD_DESKTOP_ARC_COMPLETED	Indicates that the server has finished synchronizing data after the client has auto-reconnected or the server has just begun monitoring a new desktop. The client SHOULD assume that any window or shell notify icon not received during the synchronization is discarded. This flag MUST only be combined with the WINDOW_ORDER_TYPE_DESKTOP flag.
0x00000020 WINDOW_ORDER_FIELD_DESKTOP_ACTIVEWND	Indicates that the **ActiveWindowId** field is present.
0x00000010 WINDOW_ORDER_FIELD_DESKTOP_ZORDER	Indicates that the **NumWindowIds** field is present. If the **NumWindowIds** field has a value greater than 0, the **WindowIDs** field MUST also be present.

ActiveWindowId (4 bytes): Optional. An unsigned 32-bit integer. The ID of the currently active window on the server. This field is present if and only if the WINDOW_ORDER_FIELD_DESKTOP_ACTIVEWND flag is set in the **FieldsPresentFlags** field of the TS_DESKTOP_ORDER_HEADER packet (section 2.2.1.3.3.1).

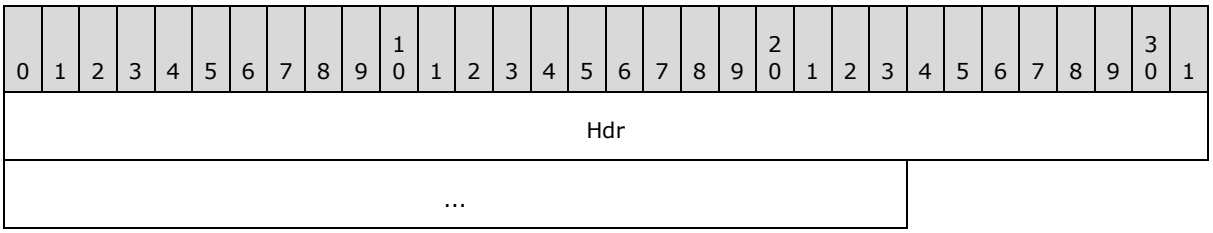
NumWindowIds (1 byte): Optional. An unsigned 8-bit integer. The number of top-level windows on the server. This field is present if and only if the WINDOW_ORDER_FIELD_DESKTOP_ZORDER flag is set in the **FieldsPresentFlags** field of the TS_DESKTOP_ORDER_HEADER packet (section 2.2.1.3.3.1).

WindowIds (variable): Variable length. An array of 4-byte window IDs, corresponding to the IDs of the top-level windows on the server, ordered by their **Z-order** on the server. The number of window IDs in the array is equal to the value of the **NumWindowIds** field.

This field is present if and only if the **NumWindowIds** field is greater than 0 and the WINDOW_ORDER_FIELD_DESKTOP_ZORDER flag is set in the **FieldsPresentFlags** field of the TS_DESKTOP_ORDER_HEADER packet (section 2.2.1.3.3.1).

2.2.1.3.3.2.2 Non-Monitored Desktop

The Non-Monitored Desktop packet is generated by the server when it is not actively monitoring the current desktop on the server.



Hdr (7 bytes): Seven bytes. A TS_DESKTOP_ORDER_HEADER header. The **FieldsPresentFlags** field of the header MUST be constructed using the following values.

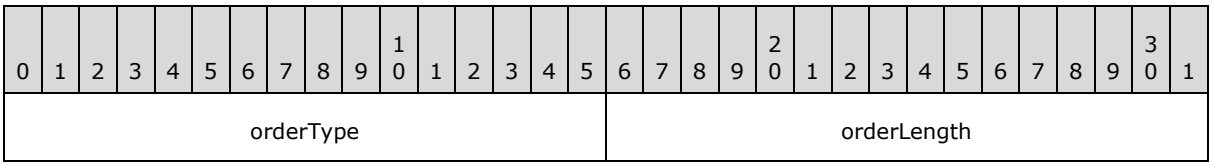
Value	Meaning
0x04000000 WINDOW_ORDER_TYPE_DESKTOP	Indicates an order specifying a desktop. This flag MUST be set.
0x00000001 WINDOW_ORDER_FIELD_DESKTOP_NONE	Indicates that the server will not be sending information for the server's current input desktop. This flag MUST be set.

2.2.2 Static Virtual Channel Protocol

The RAIL Static Virtual Channel is responsible for communicating non–RDP specific data between the RAIL client and server. The following sections outline the messages that are transmitted over the virtual channel.

2.2.2.1 Common Header (TS_RAIL_PDU_HEADER)

The TS_RAIL_PDU_HEADER packet contains information common to every RAIL Virtual Channel PDU.



orderType (2 bytes): An unsigned 16-bit integer. The type of the Virtual Channel message. MUST be one of the following values.

Value	Meaning
TS_RAIL_ORDER_EXEC 0x0001	Indicates a Client Execute PDU from client to server.
TS_RAIL_ORDER_ACTIVATE 0x0002	Indicates a Client Activate PDU from client to server.
TS_RAIL_ORDER_SYSPARAM 0x0003	Indicates a Client System Parameters Update PDU from client to server or a Server System Parameters Update PDU from server to client.
TS_RAIL_ORDER_SYSCOMMAND 0x0004	Indicates a Client System Command PDU from client to server.

Value	Meaning
TS_RAIL_ORDER_HANDSHAKE 0x0005	Indicates a bi-directional Handshake PDU.
TS_RAIL_ORDER_NOTIFY_EVENT 0x0006	Indicates a Client Notify Event PDU from client to server.
TS_RAIL_ORDER_WINDOWMOVE 0x0008	Indicates a Client Window Move PDU from client to server.
TS_RAIL_ORDER_LOCALMOVESIZE 0x0009	Indicates a Server Move/Size Start PDU and a Server Move/Size End PDU from server to client.
TS_RAIL_ORDER_MINMAXINFO 0x000a	Indicates a Server Min Max Info PDU from server to client.
TS_RAIL_ORDER_CLIENTSTATUS 0x000b	Indicates a Client Information PDU from client to server.
TS_RAIL_ORDER_SYSMENU 0x000c	Indicates a Client System Menu PDU from client to server.
TS_RAIL_ORDER_LANGBARINFO 0x000d	Indicates a Server Language Bar Information PDU from server to client, or a Client Language Bar Information PDU from client to server.
TS_RAIL_ORDER_EXEC_RESULT 0x0080	Indicates a Server Execute Result PDU from server to client.
TS_RAIL_ORDER_GET_APPID_REQ 0x000E	Indicates a Client Get Application ID PDU from client to server.
TS_RAIL_ORDER_GET_APPID_RESP 0x000F	Indicates a Server Get Application ID Response PDU from server to client.

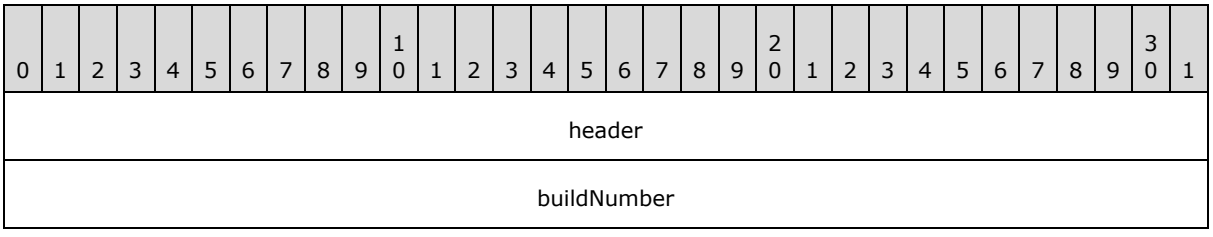
orderLength (2 bytes): An unsigned 16-bit integer. The length of the Virtual Channel PDU, in bytes.

2.2.2.2 Initialization Messages

Initialization messages are exchanged between client and server at the start of a RAIL session.

2.2.2.2.1 Handshake PDU (TS_RAIL_ORDER_HANDSHAKE)

The Handshake PDU is exchanged between the server and the client to establish that both endpoints are ready to begin RAIL mode. The server sends the Handshake PDU and the client responds with the Handshake PDU.

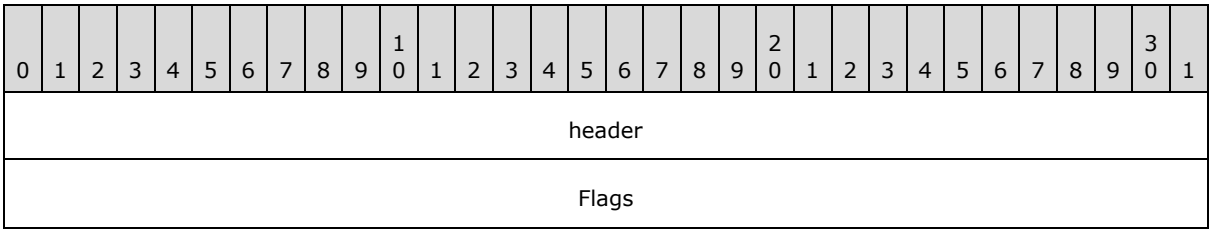


header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of the header MUST be set to 0x0005 (TS_RAIL_ORDER_HANDSHAKE).

buildNumber (4 bytes): An unsigned 32-bit integer. The build or version of the sending party.

2.2.2.2.2 Client Information PDU (TS_RAIL_ORDER_CLIENTSTATUS)

The Client Information PDU is sent from client to server and contains information about RAIL client state and features supported by the client.



header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of header MUST be set to 0x000b (TS_RAIL_ORDER_CLIENTSTATUS).

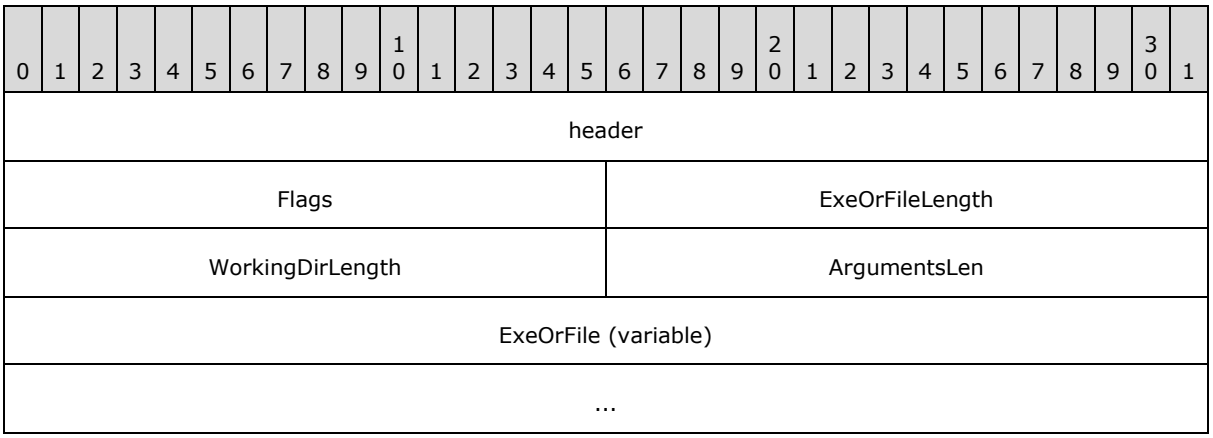
Flags (4 bytes): An unsigned 32-bit integer. RAIL features that are supported by the client. MUST be set to one of the following.

Value	Meaning
TS_RAIL_CLIENTSTATUS_ALLOWLOCALMOVESIZE 0x00000001	Indicates that the client supports the local move/size RAIL feature.
TS_RAIL_CLIENTSTATUS_AUTORECONNECT 0x00000002	Indicates that the client is auto-reconnecting to the server after an unexpected disconnect of the session.

2.2.2.3 Program Launching Messages

2.2.2.3.1 Client Execute PDU (TS_RAIL_ORDER_EXEC)

The Client Execute PDU is sent from a client to a server to request that a remote application launch on the server.



WorkingDir (variable)
...
Arguments (variable)
...

header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of the header MUST be set to 0x0001 (TS_RAIL_ORDER_EXEC).

Flags (2 bytes): An unsigned 16-bit integer. Specifies a bitfield of flags that indicate modifications to the Client Execute PDU fields.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0

Where the bits are defined as:

Value	Description
A TS_RAIL_EXEC_FLAG_EXPAND_WORKINGDIRECTORY	The environment variables in the WorkingDir field MUST be expanded on the server.
B TS_RAIL_EXEC_FLAG_TRANSLATE_FILES	The drive letters in the file path MUST be converted to corresponding mapped drives on the server. This flag MUST NOT be set if the TS_RAIL_EXEC_FLAG_FILE (0x0004) flag is not set.
C TS_RAIL_EXEC_FLAG_FILE	If this flag is set, the **ExeOrFile** field refers to a file path. If it is not set, the **ExeOrFile** field refers to an executable.
D TS_RAIL_EXEC_FLAG_EXPAND_ARGUMENTS	The environment variables in the **Arguments** field MUST be expanded on the server.

ExeOrFileLength (2 bytes): An unsigned 16-bit integer. Specifies the length of the **ExeOrFile** field in bytes. The length MUST be nonzero. The maximum length is 520 bytes.

WorkingDirLength (2 bytes): An unsigned 16-bit integer. Specifies the length of the WorkingDir field in bytes. The minimum length is 0, and the maximum length is 520 bytes.

ArgumentsLen (2 bytes): An unsigned 16-bit integer. Specifies the length of the Arguments field in bytes. The minimum length is 0, and the maximum length is 16000 bytes. <10>

ExeOrFile (variable): UNICODE_STRING. Variable length. Specifies the executable or file path to be launched on the server, as a non-null-terminated UNICODE_STRING. This field MUST be

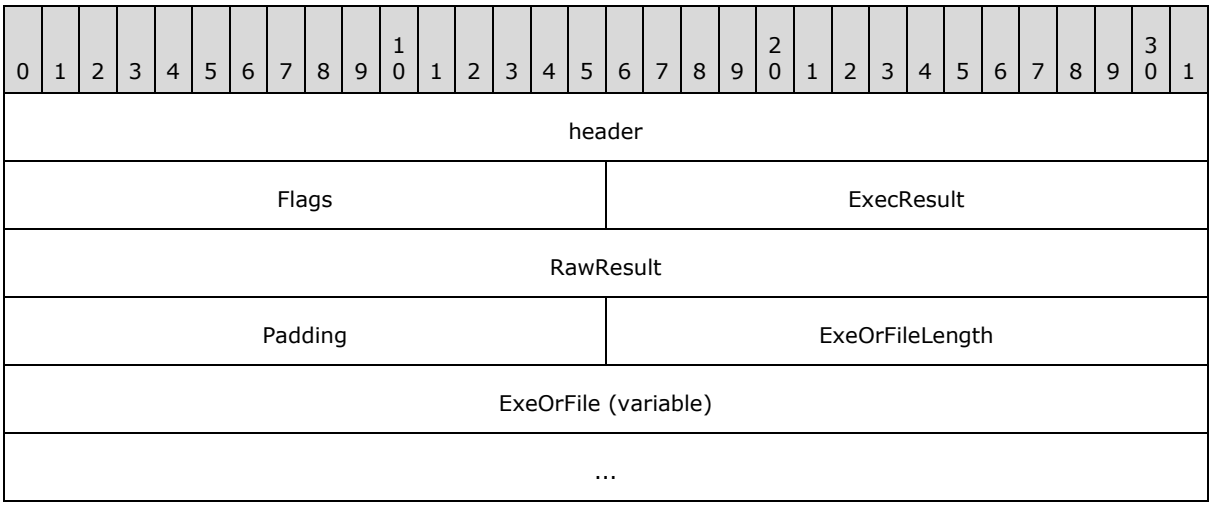
present. The maximum length of this field, including file path translations (see TS_RAIL_EXEC_FLAG_TRANSLATE_FILES mask of Flags field) is 520 bytes.

WorkingDir (variable): Optional UNICODE_STRING. Variable length. Specifies the working directory of the launched **ExeOrFile** field, as a non-null-terminated UNICODE_STRING. If the **WorkingDirLength** field is 0, this field MUST NOT be present; otherwise, it MUST be present. The maximum length of this field, including expanded environment variables (see TS_RAIL_EXEC_FLAG_EXPAND_WORKINGDIRECTORY mask of Flags field) is 520 bytes.

Arguments (variable): Optional UNICODE_STRING. Variable length. Specifies the arguments to the **ExeOrFile** field, as a non-null-terminated UNICODE_STRING. If the ArgumentsLength field is 0, this field MUST NOT be present; otherwise, it MUST be present. The maximum length of this field, including expanded environment variables (see TS_RAIL_EXEC_FLAG_EXPAND_ARGUMENTS mask of Flags field) is 16000 bytes.

2.2.2.3.2 Server Execute Result PDU (TS_RAIL_ORDER_EXEC_RESULT)

The Server Execute Result PDU is sent from server to client in response to a Client Execute PDU request, and contains the result of the server's attempt to launch the requested executable.



header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_EXEC_RESULT (0x0080).

Flags (2 bytes): An unsigned 16-bit integer. Identical to the Flags field of the Client Execute PDU. The server sets this field to enable the client to match the Client Execute PDU with the Server Execute Result PDU.

ExecResult (2 bytes): An unsigned 16-bit integer. The result of the Client Execute PDU. This field MUST be set to one of the following values.

Value	Meaning
RAIL_EXEC_S_OK 0x0000	The Client Execute request was successful and the requested application or file has been launched.
RAIL_EXEC_E_HOOK_NOT_LOADED 0x0001	The Client Execute request could not be satisfied because the server is not monitoring the current input desktop.

Value	Meaning
RAIL_EXEC_E_DECODE_FAILED 0x0002	The Execute request could not be satisfied because the request PDU was malformed.
RAIL_EXEC_E_NOT_IN_ALLOWLIST 0x0003	The Client Execute request could not be satisfied because the requested application was blocked by policy from being launched on the server.
RAIL_EXEC_E_FILE_NOT_FOUND 0x0005	The Client Execute request could not be satisfied because the application or file path could not be found.
RAIL_EXEC_E_FAIL 0x0006	The Client Execute request could not be satisfied because an unspecified error occurred on the server.
RAIL_EXEC_E_SESSION_LOCKED 0x0007	The Client Execute request could not be satisfied because the remote session is locked.

RawResult (4 bytes): An unsigned 32-bit integer. Contains an operating system-specific return code for the result of the Client Execute request.<11>

Padding (2 bytes): An unsigned 16-bit integer. Not used.

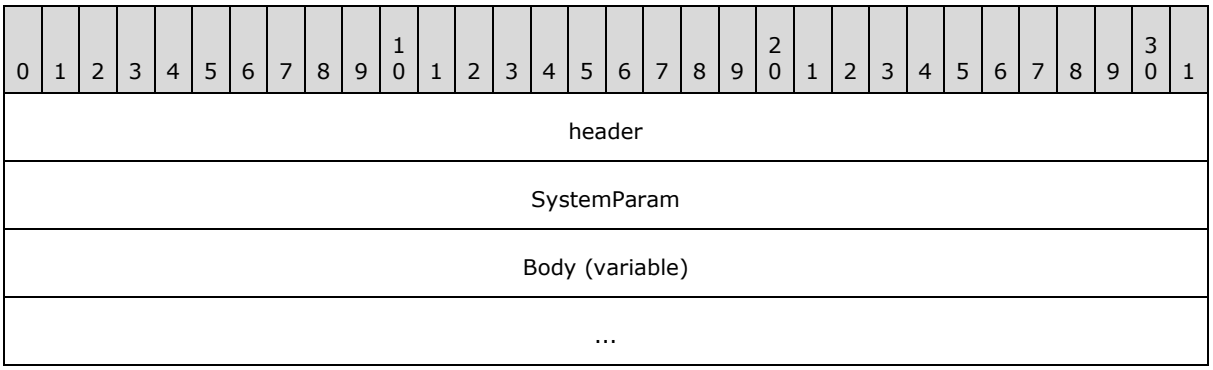
ExeOrFileLength (2 bytes): An unsigned 16-bit integer. Specifies the length of the **ExeOrFile** field in bytes. The length MUST be nonzero. The maximum length is 520 bytes.

ExeOrFile (variable): The executable or file that was attempted to be launched. This field is copied from the **ExeOrFile** field of the Client Execute PDU. The server sets this field to enable the client to match the Client Execute PDU with the Server Execute Result PDU.

2.2.2.4 Local Client System Parameters Update Messages

2.2.2.4.1 Client System Parameters Update PDU (TS_RAIL_ORDER_SYSPARAM)

The Client System Parameters Update PDU is sent from the client to the server to synchronize system parameters on the server with those on the client.



header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of header MUST be set to TS_RAIL_ORDER_SYSPARAM(0x0003).

SystemParam (4 bytes): An unsigned 32-bit integer. The type of system parameter being transmitted. The field MUST be set to one of the following values.

Value	Meaning
SPI_SETDRAGFULLWINDOWS 0x00000025	The system parameter for full-window drag.
SPI_SETKEYBOARDCUES 0x0000100B	The system parameter to determine whether menu access keys are always underlined.
SPI_SETKEYBOARDPREF 0x00000045	The system parameter specifying a preference for the keyboard instead of the mouse.
SPI_SETWORKAREA 0x0000002F	The system parameter to set the size of the work area. The work area is the portion of the screen not obscured by the system taskbar or by **application desktop toolbars**.
RAIL_SPI_DISPLAYCHANGE 0x0000F001	The system parameter for display resolution.
SPI_SETMOUSEBUTTONSWAP 0x00000021	The system parameter to swap or restore the meaning of the left and right mouse buttons.
RAIL_SPI_TASKBARPOS 0x0000F000	The system parameter to indicate the size of the client taskbar.
SPI_SETHIGHCONTRAST 0x00000043	The system parameter to set the parameters of the HighContrast accessibility feature.

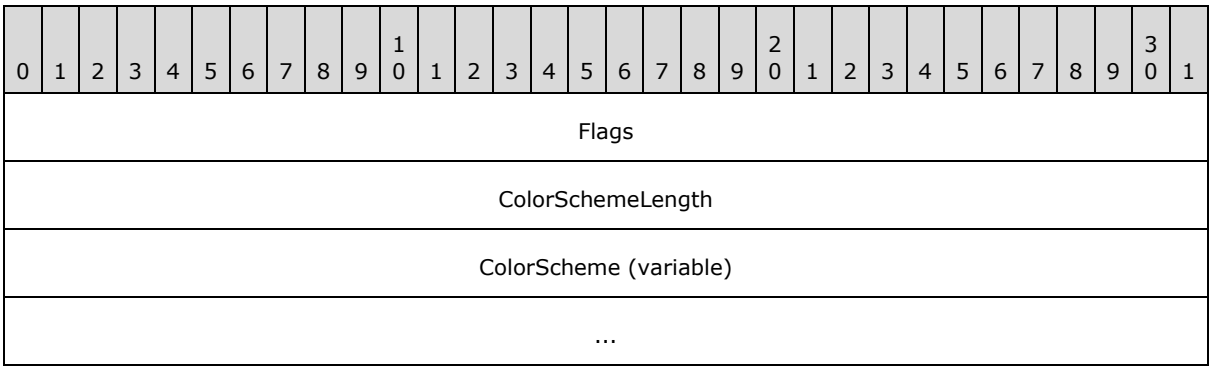
Body (variable): The contents of this field depend on the **SystemParameter** field. The following table outlines the valid values of the **SystemParameter** field (Value column) and corresponding values of the **Body** field (Meaning column).

Value	Meaning
SPI_SETDRAGFULLWINDOWS 0x0025	Size of Body field: 1 byte. 0 (FALSE): Full Window Drag is disabled. Nonzero (TRUE): Full Window Drag is enabled.
SPI_SETKEYBOARDCUES 0x100B	Size of Body field: 1 byte. 0 (FALSE): Menu Access Keys are underlined only when the menu is activated by the keyboard. Nonzero (TRUE): Menu Access Keys are always underlined.
SPI_SETKEYBOARDPREF 0x0045	Size of Body field: 1 byte. 0 (FALSE): The user does not prefer the keyboard over mouse. Nonzero (TRUE): The user prefers the keyboard over mouse. This causes applications to display keyboard interfaces that would otherwise be hidden.
SPI_SETMOUSEBUTTONSWAP 0x0021	Size of Body field: 1 byte. 0 (FALSE): Restores the meaning of the left and right mouse buttons to their original meanings. Nonzero (TRUE): Swaps the meaning of the left and right mouse buttons.
SPI_SETWORKAREA 0x002F	Size of Body field: 8 bytes. The body is a TS_RECTANGLE_16 structure that defines the work area in virtual screen coordinates. In a system with multiple

Value	Meaning
	display monitors, the work area is that of the monitor that contains the specified rectangle. For more information about virtual screen coordinates, see [MSDN-VIRTUALSCR].
RAIL_SPI_DISPLAYCHANGE 0xF001	Size of Body field: 8 bytes. The body is a TS_RECTANGLE_16 structure that indicates the new display resolution in virtual screen coordinates. For more information about virtual screen coordinates, see [MSDN-VIRTUALSCR].
RAIL_SPI_TASKBARPOS 0xF000	Size of Body field: 8 bytes. The body is a TS_RECTANGLE_16 structure that indicates the size of the client taskbar.
SPI_SETHIGHCONTRAST 0x0043	Size of Body field: Variable number of bytes. The body is a TS_HIGHCONTRAST structure.

2.2.2.4.2 High Contrast System Information Structure (TS_HIGHCONTRAST)

The TS_HIGHCONTRAST packet defines parameters for the high-contrast accessibility feature.



Flags (4 bytes): An unsigned 32-bit integer. This field is opaque to RAIL. It is transmitted from the client to the server and used by the server to set the High Contrast parameters.<12>

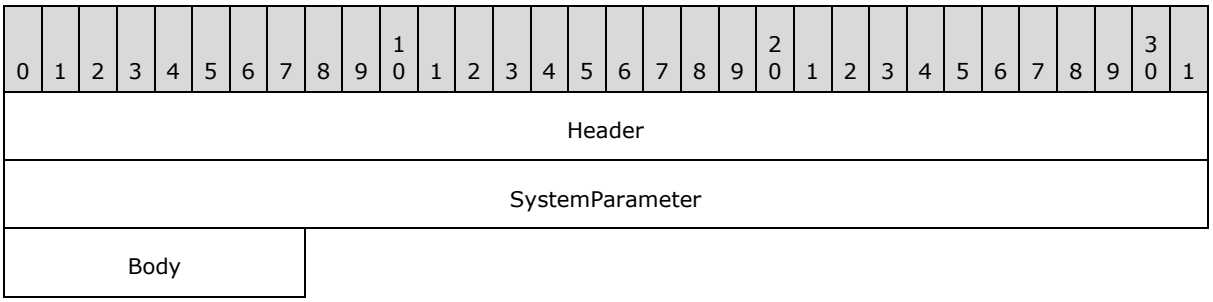
ColorSchemeLength (4 bytes): An unsigned 32-bit integer. The length, in bytes, of the ColorScheme field.

ColorScheme (variable): UNICODE_STRING. Variable length. The Windows-specific name of the High Contrast Color Scheme, specified as a null-terminated UNICODE_STRING.<13>

2.2.2.5 Server System Parameters Update Messages

2.2.2.5.1 Server System Parameters Update PDU (TS_RAIL_ORDER_SYSPARAM)

The Server System Parameters Update PDU is sent from the server to client to synchronize system parameters on the client with those on the server.



Header (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of header MUST be set to 0x03 (TS_RAIL_ORDER_SYSPARAM).

SystemParameter (4 bytes): An unsigned 32-bit integer. The type of system parameter being transmitted. This field MUST be set to one of the following values.

Value	Meaning
SPI_SETSCREENSAVEACTIVE 0x00000011	The system parameter indicating whether the screen saver is enabled.
SPI_SETSCREENSAVESECURE 0x00000077	The system parameter indicating whether the desktop should be locked after switching out of screen saver mode (that is, after the screen saver starts due to inactivity, then stops due to activity).<14>

Body (1 byte): The content of this field depends on the SystemParameter field. The following table outlines the valid values of the SystemParameter field (Value column) and corresponding values of the Body field (Meaning column).

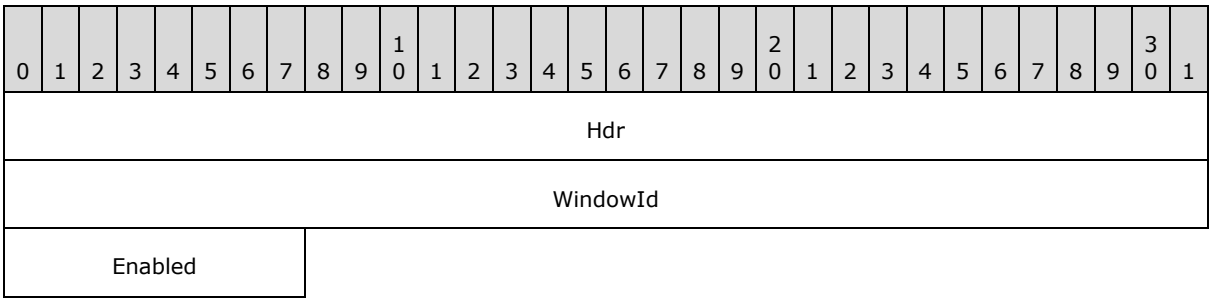
Value	Meaning
SPI_SETSCREENSAVEACTIVE 0x00000011	Size of Body field: 1 byte. 0 (FALSE): Screen saver is not enabled. Nonzero (TRUE): Screen Saver is enabled.
SPI_SETSCREENSAVESECURE 0x00000077	Size of Body field: 1 byte. 0 (FALSE): Do not lock the desktop when switching out of screen saver mode. Nonzero (TRUE): Lock the desktop when switching out of screen saver mode.

2.2.2.6 Local Client Event Messages

These messages are generated by the client whenever a window or notification icon event occurs on the client side that is not communicated via the RDP channel.

2.2.2.6.1 Client Activate PDU (TS_RAIL_ORDER_ACTIVATE)

The Client Activate PDU is sent from client to server when a local RAIL window on the client is activated or deactivated.



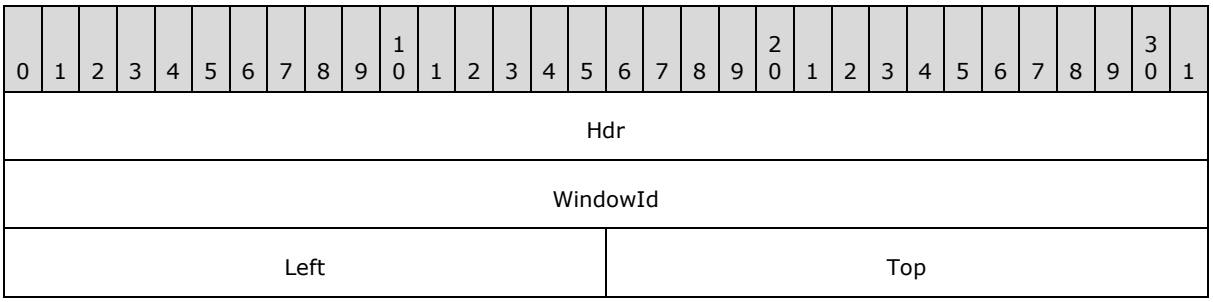
Hdr (4 bytes): A TS_RAIL_PDU_HEADER structure. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_ACTIVATE (0x0002).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the associated window on the server that should be activated or deactivated.

Enabled (1 byte): An unsigned 8-bit integer. Indicates whether the window should be activated (value = nonzero) or deactivated (value = 0).

2.2.2.6.2 Client System Menu PDU (TS_RAIL_ORDER_SYSMENU)

The Client System Menu PDU packet is sent from the client to the server when a local RAIL window on the client receives a command to display its **System menu**. This command is forwarded to the server via the System menu PDU.



Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_SYSMENU (0x000C).

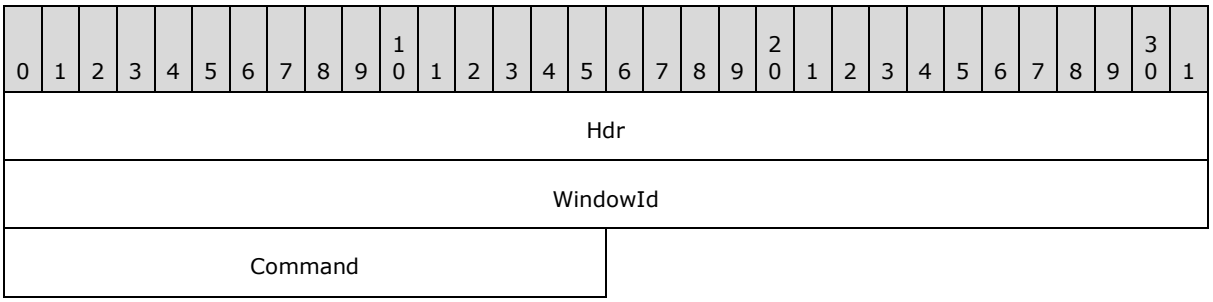
WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server that SHOULD display its System menu.

Left (2 bytes): An unsigned 16-bit integer. The x-coordinate of the top-left corner at which the System menu should be displayed. Specified in screen coordinates.

Top (2 bytes): An unsigned 16-bit integer. The y-coordinate of the top-left corner at which the System menu should be displayed. Specified in screen coordinates.

2.2.2.6.3 Client System Command PDU (TS_RAIL_ORDER_SYSCOMMAND)

The Client System Command PDU packet is sent from the client to the server when a local RAIL window on the client receives a command to perform an action on the window, such as minimize or maximize. This command is forwarded to the server via the **System Command** Menu PDU.



Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_SYSCOMMAND (0x0004).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server to activate or deactivate.

Command (2 bytes): An unsigned 16-bit integer. Specifies the type of command. The field MUST be one of the following values.

Value	Meaning
SC_SIZE 0xF000	Resize the window.
SC_MOVE 0xF010	Move the window.
SC_MINIMIZE 0xF020	Minimize the window.
SC_MAXIMIZE 0xF030	Maximize the window.
SC_CLOSE 0xF060	Close the window.
SC_KEYMENU 0xF100	The ALT + SPACE key combination was pressed; display the window's system menu.
SC_RESTORE 0xF120	Restore the window to its original shape and size.
SC_DEFAULT 0xF160	Perform the default action of the window's system menu.

2.2.2.6.4 Client Notify Event PDU (TS_RAIL_ORDER_NOTIFY_EVENT)

The Client Notify Event PDU packet is sent from a client to a server when a local **RAIL Notification Icon** on the client receives a keyboard or mouse message from the user. This notification is forwarded to the server via the Notify Event PDU.

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
Hdr																															
WindowId																															
NotifyIconId																															
Message																															

Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_NOTIFY_EVENT (0x0006).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the associated window on the server that owns the notification icon being specified in the PDU.

NotifyIconId (4 bytes): An unsigned 32-bit integer. The ID of the associated notification icon on the server that SHOULD receive the keyboard or mouse interaction.

Message (4 bytes): An unsigned 32-bit integer. The message being sent to the notification icon on the server.

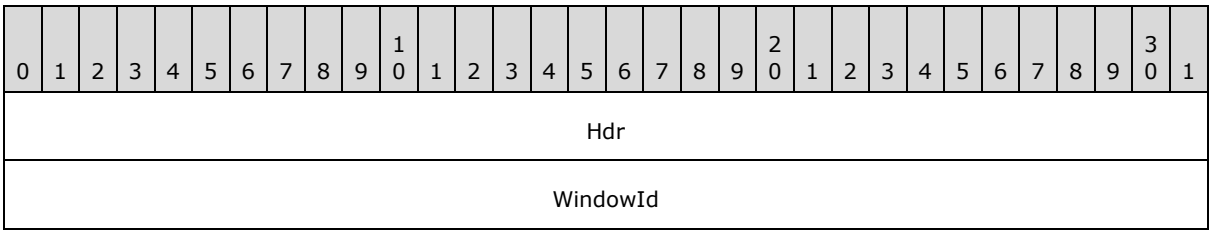
Value	Meaning
WM_LBUTTONDOWN 0x00000201	The user pressed the left mouse button in the client area of the notification icon.
WM_LBUTTONUP 0x00000202	The user released the left mouse button while the cursor was in the client area of the notification icon.
WM_RBUTTONDOWN 0x00000204	The user pressed the right mouse button in the client area of the notification icon.
WM_RBUTTONUP 0x00000205	The user released the right mouse button while the cursor was in the client area of the notification icon.
WM_CONTEXTMENU 0x0000007B	The user selected a notify icon's shortcut menu with the keyboard. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).
WM_LBUTTONDBLCLK 0x00000203	The user double-clicked the left mouse button in the client area of the notification icon.
WM_RBUTTONDBLCLK 0x00000206	The user double-clicked the right mouse button in the client area of the notification icon.
NIN_SELECT 0x00000400	The user selected a notify icon with the mouse and activated it with the ENTER key. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).
NIN_KEYSELECT 0x00000401	The user selected a notify icon with the keyboard and activated it with the SPACEBAR or ENTER key. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section

Value	Meaning
	2.2.1.3.2.2.1).
NIN_BALLOONSHOW 0x00000402	The user passed the mouse pointer over an icon with which a balloon tooltip is associated (see **InfoTip** field in section 2.2.1.3.2.2.1), and the balloon tooltip was shown. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).
NIN_BALLOONHIDE 0x00000403	The icon's balloon tooltip disappeared because, for example, the icon was deleted. This message is not sent if the balloon is dismissed because of a timeout or mouse click by the user. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).
NIN_BALLOONTIMEOUT 0x00000404	The icon's balloon tooltip was dismissed because of a timeout. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).
NIN_BALLOONUSERCLICK 0x00000405	User dismissed the balloon by clicking the mouse. This message is sent only for notification icons that follow Windows 2000 behavior (see **Version** field in section 2.2.1.3.2.2.1).

2.2.2.6.5 Client Get Application ID PDU (TS_RAIL_ORDER_GET_APPID_REQ)

The Client Get Application ID PDU is sent from a client to a server. This PDU requests information from the server about the Application ID that the window SHOULD <15> have on the client.

The server MAY ignore this PDU.



Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_GET_APPID_REQ (0x000E).

WindowId (4 bytes): An unsigned 32-bit integer specifying the ID of the associated window on the server that requires needs an Application ID.

2.2.2.7 Window Move Messages

2.2.2.7.1 Server Min Max Info PDU (TS_RAIL_ORDER_MINMAXINFO)

The Server Min Max Info PDU is sent from a server to a client when a window move or resize on the server is being initiated. This PDU contains information about the minimum and maximum extents to which the window can be moved or sized.

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
Hdr																															
WindowId																															
MaxWidth																MaxHeight															
MaxPosX																MaxPosY															
MinTrackWidth																MinTrackHeight															
MaxTrackWidth																MaxTrackHeight															

Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_MINMAXINFO (0x000A).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server that is being moved or resized.

MaxWidth (2 bytes): An unsigned 16-bit integer. The width of the maximized window.

MaxHeight (2 bytes): An unsigned 16-bit integer. The height of the maximized window.

MaxPosX (2 bytes): An unsigned 16-bit integer. The x-coordinate of the top-left corner of the maximized window.

MaxPosY (2 bytes): An unsigned 16-bit integer. The y-coordinate of the top-left corner of the maximized window.

MinTrackWidth (2 bytes): An unsigned 16-bit integer. The minimum width to which the window can be resized.

MinTrackHeight (2 bytes): An unsigned 16-bit integer. The minimum height to which the window can be resized.

MaxTrackWidth (2 bytes): An unsigned 16-bit integer. The maximum width to which the window can be resized.

MaxTrackHeight (2 bytes): An unsigned 16-bit integer. The maximum height to which the window can be resized.

2.2.2.7.2 Server Move/Size Start PDU (TS_RAIL_ORDER_LOCALMOVESIZE)

The Server Move/Size Start PDU packet is sent by the server when a window on the server is beginning a move or resize. The client uses this information to initiate a local move or resize of the corresponding local window.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Hdr																															
WindowId																															
IsMoveSizeStart																MoveSizeType															
PosX																PosY															

Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_LOCALMOVESIZE (0x0009).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server that is being moved or resized.

IsMoveSizeStart (2 bytes): An unsigned 16-bit integer. Indicates that the move/size is beginning. MUST be set to a nonzero value.

MoveSizeType (2 bytes): An unsigned 16-bit integer. Indicates the type of the move/size. This value determines the meaning of the fields **PosX** and **PosY**.

Value	Meaning
RAIL_WMSZ_LEFT 0x0001	The left edge of the window is being sized.
RAIL_WMSZ_RIGHT 0x0002	The right edge of the window is being sized.
RAIL_WMSZ_TOP 0x0003	The top edge of the window is being sized.
RAIL_WMSZ_TOPLEFT 0x0004	The top-left corner of the window is being sized.
RAIL_WMSZ_TOPRIGHT 0x0005	The top-right corner of the window is being sized.
RAIL_WMSZ_BOTTOM 0x0006	The bottom edge of the window is being sized.
RAIL_WMSZ_BOTTOMLEFT 0x0007	The bottom-left corner of the window is being sized.
RAIL_WMSZ_BOTTOMRIGHT 0x0008	The bottom-right corner of the window is being sized.
RAIL_WMSZ_MOVE 0x0009	The window is being moved by using the mouse.
RAIL_WMSZ_KEYMOVE 0x000A	The window is being moved by using the keyboard.

Value	Meaning
RAIL_WMSZ_KEYSIZE 0x000B	The window is being resized by using the keyboard.

PosX (2 bytes): An unsigned 16-bit integer. The meaning of this field depends upon the value of the **MoveSizeType** field.

Value	Meaning
RAIL_WMSZ_LEFT 0x0001	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_RIGHT 0x0002	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_TOP 0x0003	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_TOPLEFT 0x0004	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_TOPRIGHT 0x0005	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOM 0x0006	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOMLEFT 0x0007	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOMRIGHT 0x0008	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_MOVE 0x0009	The horizontal offset between the window's top-left edge and the current mouse position.
RAIL_WMSZ_KEYMOVE 0x000A	The x-coordinate of the last mouse button-down.
RAIL_WMSZ_KEYSIZE 0x000B	The x-coordinate of the last mouse button-down.

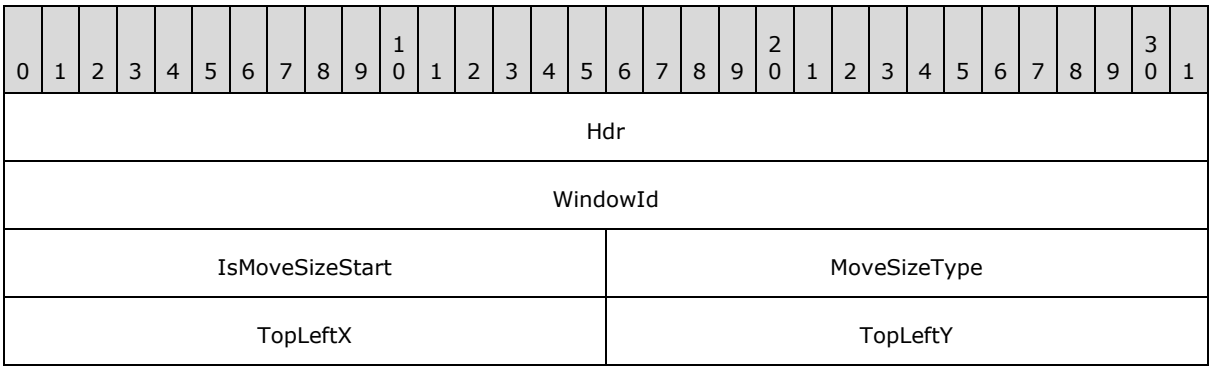
PosY (2 bytes): An unsigned 16-bit integer. The meaning of this field depends on the value of the **MoveSizeType** field.

Value	Meaning
RAIL_WMSZ_LEFT 0x0001	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_RIGHT 0x0002	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_TOP 0x0003	The y-coordinate of the last mouse button-down.

Value	Meaning
RAIL_WMSZ_TOPLEFT 0x0004	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_TOPRIGHT 0x0005	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOM 0x0006	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOMLEFT 0x0007	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_BOTTOMRIGHT 0x0008	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_MOVE 0x0009	The vertical offset between the window's top-left edge and the current mouse position.
RAIL_WMSZ_KEYMOVE 0x000A	The y-coordinate of the last mouse button-down.
RAIL_WMSZ_KEYSIZE 0x000B	The y-coordinate of the last mouse button-down.

2.2.2.7.3 Server Move/Size End PDU (TS_RAIL_ORDER_LOCALMOVESIZE)

The Server Move/Size End PDU is sent by the server when a window on the server is completing a move or resize. The client uses this information to end a local move/resize of the corresponding local window.



Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_LOCALMOVESIZE (0x0009).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server that is being moved or resized.

IsMoveSizeStart (2 bytes): An unsigned 16-bit integer. Indicates the move or resize is ending. This field MUST be set to 0.

MoveSizeType (2 bytes): An unsigned 16-bit integer. Indicates the type of the move/size.

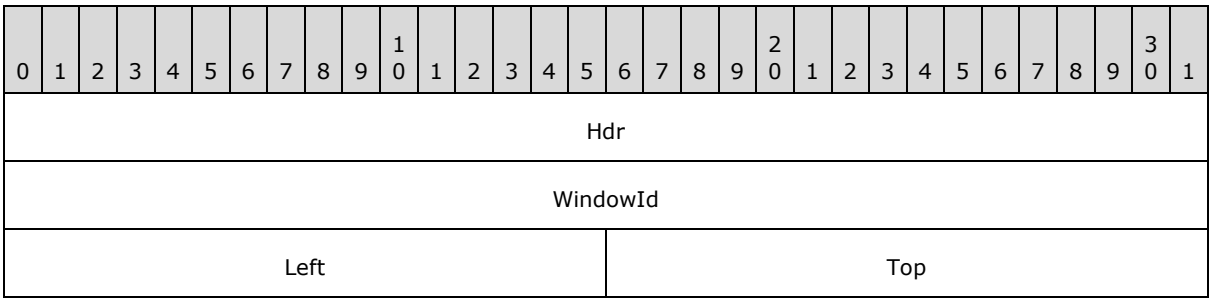
Value	Meaning
RAIL_WMSZ_LEFT 0x0001	The left edge of the window is being sized.
RAIL_WMSZ_RIGHT 0x0002	The right edge of the window is being sized.
RAIL_WMSZ_TOP 0x0003	The top edge of the window is being sized.
RAIL_WMSZ_TOPLEFT 0x0004	The top-left corner of the window is being sized.
RAIL_WMSZ_TOPRIGHT 0x0005	The top-right corner of the window is being sized.
RAIL_WMSZ_BOTTOM 0x0006	The bottom edge of the window is being sized.
RAIL_WMSZ_BOTTOMLEFT 0x0007	The bottom-left corner of the window is being sized.
RAIL_WMSZ_BOTTOMRIGHT 0x0008	The bottom-right corner of the window is being sized.
RAIL_WMSZ_MOVE 0x0009	The window is being moved by using the mouse.
RAIL_WMSZ_KEYMOVE 0x000A	The window is being moved by using the keyboard.
RAIL_WMSZ_KEYSIZE 0x000B	The window is being resized by using the keyboard.

TopLeftX (2 bytes): An unsigned 16-bit integer. The x-coordinate of the moved or resized window's top-left corner.

TopLeftY (2 bytes): An unsigned 16-bit integer. The y-coordinate of the moved or resized window's top-left corner.

2.2.2.7.4 Client Window Move PDU (TS_RAIL_ORDER_WINDOWMOVE)

The Client Window Move PDU packet is sent from the client to the server when a local window is ending a move or resize. The client communicates the locally moved or resized window's position to the server by using this packet. The server uses this information to reposition its window.



Right	Bottom

Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_WINDOWMOVE (0x0008).

WindowId (4 bytes): An unsigned 32-bit integer. The ID of the window on the server corresponding to the local window that was moved or resized.

Left (2 bytes): An unsigned 16-bit integer. The x-coordinate of the top-left corner of the window's new position.

Top (2 bytes): An unsigned 16-bit integer. The y-coordinate of the top-left corner of the window's new position.

Right (2 bytes): An unsigned 16-bit integer. The x-coordinate of the bottom-right corner of the window's new position.

Bottom (2 bytes): An unsigned 16-bit integer. The y-coordinate of the bottom-right corner of the window's new position.

2.2.2.8 Server Application ID Response

2.2.2.8.1 Server Get Application ID Response PDU (TS_RAIL_ORDER_GET_APPID_RESP)

The Server Get Application ID Response PDU is sent from a server to a client. This PDU MAY be sent to the client as a response to a Client Get Application ID PDU.

This PDU specifies the Application ID that the specified window SHOULD <16> have on the client. The client MAY ignore this PDU.

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
Hdr																															
WindowId																															
ApplicationId																															
...																															
...																															
...																															
...																															
...																															

...
...
(ApplicationId cont'd for 56 rows)

Hdr (4 bytes): A TS_RAIL_PDU_HEADER header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_GET_APPID_RESP (0x000F).

WindowId (4 bytes): An unsigned 32-bit integer specifying the ID of the associated window on the server whose Application ID is being sent to the client.

ApplicationId (256 bytes): A null-terminated string of **Unicode characters** specifying the Application ID that the Client SHOULD associate with its window, if it supports using the Application ID for identifying and grouping windows.

2.2.2.9 Language Bar Messages

2.2.2.9.1 Language Bar Information PDU (TS_RAIL_ORDER_LANGBARINFO)

The Language Bar Information PDU is used to set the language bar status. It is sent from a client to a server or a server to a client, but only when both support the Language Bar docking capability (TS_RAIL_LEVEL_DOCKED_LANGBAR_SUPPORTED). This PDU contains information about the language bar status.

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
Hdr																															
LanguageBarStatus																															

Hdr (4 bytes): A TS_RAIL_PDU_HEADER (section 2.2.2.1) header. The **orderType** field of the header MUST be set to TS_RAIL_ORDER_LANGBARINFO (0x000D).

LanguageBarStatus (4 bytes): An unsigned 32-bit integer. The possible values are indicated in the table below. The server sends the **LanguageBarStatus** it retrieves from the local language bar:

Value	Meaning
TF_SFT_SHOWNORMAL 0x00000001	Display the language bar as a floating window. This constant cannot be combined with the TF_SFT_DOCK, TF_SFT_MINIMIZED, TF_SFT_HIDDEN, or TF_SFT_DESKBAND constants.
TF_SFT_DOCK 0x00000002	Dock the language bar in its own task pane. This constant cannot be combined with the TF_SFT_SHOWNORMAL, TF_SFT_MINIMIZED, TF_SFT_HIDDEN, or TF_SFT_DESKBAND constants. <17>

Value	Meaning
TF_SFT_MINIMIZED 0x00000004	Display the language bar as a single icon in the system tray. This constant cannot be combined with the TF_SFT_SHOWNORMAL, TF_SFT_DOCK, TF_SFT_HIDDEN, or TF_SFT_DESKBAND constants.
TF_SFT_HIDDEN 0x00000008	Hide the language bar. This constant cannot be combined with the TF_SFT_SHOWNORMAL, TF_SFT_DOCK, TF_SFT_MINIMIZED, or TF_SFT_DESKBAND constants.
TF_SFT_NOTRANSPARENCY 0x00000010	Make the language bar opaque.
TF_SFT_LOWTRANSPARENCY 0x00000020	Make the language bar partially transparent. <18>
TF_SFT_HIGHTRANSPARENCY 0x00000040	Make the language bar highly transparent. <19>
TF_SFT_LABELS 0x00000080	Display text labels next to language bar icons.
TF_SFT_NOLABELS 0x00000100	Hide language bar icon text labels.
TF_SFT_EXTRAICONSONMINIMIZED 0x00000200	Display text service icons on the taskbar when the language bar is minimized.
TF_SFT_NOEXTRAICONSONMINIMIZED 0x00000400	Hide text service icons on the taskbar when the language bar is minimized.
TF_SFT_DESKBAND 0x00000800	Dock the language bar in the system task bar. This constant cannot be combined with the TF_SFT_SHOWNORMAL, TF_SFT_DOCK, TF_SFT_MINIMIZED, or TF_SFT_HIDDEN constants. <20>

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

3.1.1.1 Server State Machine

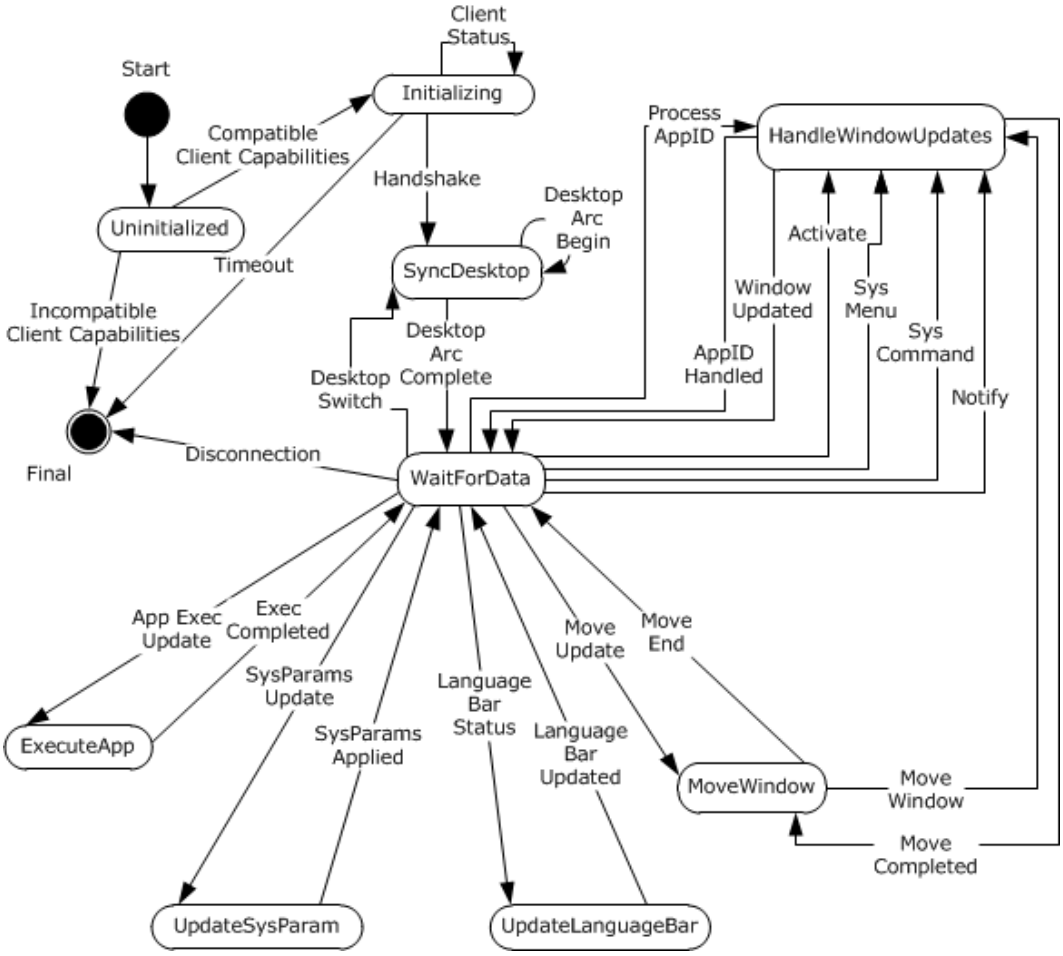


Figure 4: Server State Machine Diagram

State	Description
Uninitialized	This is the initial state of the server. In this state, the server waits for the Remote Programs Capability Set (section 2.2.1.1.1) and the Window List Capability Set (section 2.2.1.1.2) from the client. On receiving these capability sets, the server processes them as specified in section 3.3.5.1.5. If the server finds compatible settings, it transitions to the Initializing state. Otherwise, the connection is terminated (see [MS-RDPBCGR] section 1.3.1.4.2).
Initializing	In this state, the server examines the Handshake PDU (section 2.2.2.2.1) and

State	Description
	the Client Information PDU (section 2.2.2.2.2). On receiving these, the server processes the Handshake PDU as specified in section 3.1.5.2 and the Client Information PDU as specified in section 3.3.5.2.1.1, and transitions to the SyncDesktop state. If the server does not receive the Handshake PDU within a specified time, it may disconnect the connection (see section 3.1.2).
SyncDesktop	In this state, the server syncs its desktop with that of the client. The server transitions to this state either during the RAIL connection synchronization (see section 1.3.2.3) or on detection of a desktop switch (see section 3.3.5.1.8). After completion of the desktop sync (as specified in section 3.3.5.1.8), the server transitions to the WaitForData state.
WaitForData	In this state, the server waits for all non-initialization messages received on the static virtual channel (see section 3.3.5.2). On receiving a Client Execute PDU (section 2.2.2.3.1), the server transitions to the ExecuteApp state. On receiving a Client System Parameters Update PDU (section 2.2.2.4.1), the server transitions to the UpdateSysParam state. On receiving a Language Bar Information PDU (section 2.2.2.9.1), the server transitions to the UpdateLanguageBar state. On receiving a Client Window Move PDU (section 2.2.2.7.4), the server transitions to the MoveWindow state. On receiving a Client Activate PDU (section 2.2.2.6.1), a Client System Menu PDU (section 2.2.2.6.2), a Client System Command PDU (section 2.2.2.6.3), a Client Notify Event PDU (section 2.2.2.6.4), or a Client Get Application ID PDU (section 2.2.2.6.5), the server transitions to the HandleWindowUpdates state. On detecting a desktop switch, the server transitions to the SyncDesktop state.
ExecuteApp	In this state, the server processes the Client Execute PDU, as specified in section 3.3.5.2.2.1, and sends the Server Execute Result PDU (section 2.2.2.3.2), as specified in section 3.3.5.2.2.2. The server then transitions back to the WaitForData state.
UpdateSysParam	In this state, the server processes the Client System Parameters Update PDU, as specified in section 3.3.5.2.3.1, and transitions back to the WaitForData state.
UpdateLanguageBar	In this state, the server processes the Language Bar Information PDU, as specified in section 3.3.5.2.5.5, and transitions back to the WaitForData state.
MoveWindow	In this state, the server processes the Client Window Move PDU, as specified in section 3.3.5.2.6.3. If the window ID is valid, the server transitions to the HandleWindowUpdates state. Once the window is moved, the server transitions back to the MoveWindow state. If applicable, the server also sends a Server Move/Size End PDU (section 2.2.2.7.3) to the client, as specified in section 3.3.5.2.6.4. After all processing is complete, the server transitions back to the WaitForData state.
HandleWindowUpdates	In this state, the server processes local client events relevant to individual windows: the Client Activate PDU, as specified in section 3.3.5.2.5.1; the Client System Menu PDU, as specified in section 3.3.5.2.5.2; the Client System Command PDU, as specified in section 3.3.5.2.5.3; and the Client Notify Event PDU, as specified in section 3.3.5.2.5.4. The server also processes the Client Get Application ID PDU, as specified in section 3.3.5.2.7.1, and sends the Server Get Application ID Response PDU (section 2.2.2.8.1), as specified in section 3.3.5.2.7.2. After all processing is complete, the server transitions back to the WaitForData state.

3.1.1.2 Icon Cache Support

If the implementation supports icon caching, then the following state is negotiated between the client and server as part of the Window List Capability Set order (section 2.2.1.1.2), and thereafter maintained on both client and server.

NumIconCaches: the number of discrete caches for icons maintained on client and server.

NumIconCacheEntries: the number of entries allocated in each icon cache.

Once an icon cache capability is established, individual entries in the cache are identified by a Cached Icon Info packet (section 2.2.1.2.4), containing a pair of index values designating the specific icon cache and the entry within that cache.

3.1.2 Timers

A handshake timer MAY<21> be used by the client and/or server to wait for the Handshake PDU from the sending party.

3.1.3 Initialization

The static virtual channel between the client and the server MUST be established before protocol operations can commence (see section 1.3.2.1 for an overview).

The Handshake PDU (as specified in section 2.2.2.2.1) is exchanged between the server and the client to establish that both endpoints are ready to begin RAIL mode.

The Client Information PDU (as specified in section 2.2.2.2.2) is sent from a client to a server and contains information about RAIL client state and features supported by the client.

3.1.4 Higher-Layer Triggered Events

No higher-layer triggered events are used.

3.1.5 Message Processing Events and Sequencing Rules

The following sections describe construction and processing of common messages.

3.1.5.1 Constructing Handshake PDU

The Handshake PDU is constructed during initialization of the remote applications integrated locally (RAIL) virtual channel. The buildNumber field SHOULD be initialized to the build or version of the sending party. This PDU MUST be sent before any other PDU on the virtual channel.

3.1.5.2 Processing Handshake PDU

The receiving party SHOULD check the **buildNumber** field to verify compatibility of the receiver with the sender.<22>

The receiving party MUST NOT process any other virtual channel PDUs unless the Handshake PDU has been received.

3.1.6 Timer Events

Upon the expiration of the handshake timer (as specified in section 3.1.2), the receiving party SHOULD drop the connection.

3.1.7 Other Local Events

No additional events are used.

3.2 Client Details

3.2.1 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 what is described in this document.

Note: It is possible to implement the following conceptual data by using a variety of techniques as long as the implementation produces external behavior that is consistent with what is described in this document.

3.2.1.1 Windowing Support Level

The windowing support level determines whether the server is capable of supporting Windowing Alternate Secondary Drawing Orders and the following flags: WINDOW_ORDER_FIELD_CLIENTAREASIZE, WINDOW_ORDER_FIELD_RPCONTENT, and WINDOW_ORDER_FIELD_ROOTPARENT. This is communicated to the client by the **WndSupportLevel** field, as part of the Window List Capability Set (section 2.2.1.1.2).

3.2.2 Timers

No timers are used.

3.2.3 Initialization

3.2.4 Higher-Layer Triggered Events

There are no higher-layer triggered events.

3.2.5 Message Processing Events and Sequencing Rules

The following sections describe construction and processing of client messages.

3.2.5.1 Updates to RDP Core Protocol

3.2.5.1.1 Constructing Client MCS Connect Initial PDU

The Client MCS Connect Initial PDU is constructed by the client during the connection establishment phase, as specified in [MS-RDPBCGR] section 3.2.5.3.3.

For remote applications integrated locally (RAIL) clients, the **clientNetworkData** field (as specified in [MS-RDPBCGR] section 2.2.1.3) MUST be present and MUST contain a CHANNEL_DEF structure in channelDefArray for the RAIL virtual channel. This informs the server that the client wants to use a static virtual channel for communicating RAIL virtual channel messages.<23>

3.2.5.1.2 Processing Server MCS Connect Response PDU

This PDU is sent by the server in response to the Client MCS Connect Initial PDU. It is processed by the client, as specified in [MS-RDPBCGR] section 3.2.5.3.4.

3.2.5.1.3 Constructing Client Info PDU

The Client Info PDU (as specified in [MS-RDPBCGR] section 2.2.1.11) is constructed by the client during the connection establishment phase (as specified in [MS-RDPBCGR] section 3.2.5.3.11).

For remote applications integrated locally (RAIL) clients, the **flags** field of the Info Packet (as specified in [MS-RDPBCGR], section 2.2.1.11.1.1) MUST have the INFO_RAIL (0x00008000) flag set. This informs the server that the client wants to create a RAIL session.

3.2.5.1.4 Constructing Confirm Active PDU

The Confirm Active PDU is constructed by the client in response to the Demand Active PDU, as specified in [MS-RDPBCGR] section 3.2.5.3.13.2.

Remote applications integrated locally (RAIL) clients MUST populate this PDU with two RAIL-specific capabilities in the **capabilitySets** field: the Remote Programs Capability Set, as specified in section 2.2.1.1.1, and the Window List Capability Set, as specified in section 2.2.1.1.2.

The NumIconCaches and NumIconCacheEntries of the Window List Capability Set SHOULD be reported as the minimum of the corresponding values supported by the client, and those reported by the server in the Demand Active PDU. The values MUST not exceed those reported by the server in the Demand Active PDU.

3.2.5.1.5 Processing Demand Active PDU

The Demand Active PDU is processed by the client during the connection establishment phase, as specified in [MS-RDPBCGR] section 3.2.5.3.13.1.

Remote applications integrated locally (RAIL) clients MUST verify that this PDU contains two RAIL-specific capabilities in the **capabilitySets** field: the Remote Programs Capability Set, as specified in section 2.2.1.1.1, and the Window List Capability Set, as specified in section 2.2.1.1.2. If it does not contain these capability sets, or if the RailSupportLevel of the Remote Programs Capability Set is not set to at least TS_RAIL_LEVEL_SUPPORTED, or the WndSupportLevel of the Window List Capability Set is TS_WINDOW_LEVEL_NOT_SUPPORTED (0), the client MUST drop the connection.

The client SHOULD use the NumIconCaches and NumIconCacheEntries of the Window List Capability Set to determine the values of NumIconCaches and NumIconCacheEntries reported by it in the Confirm Active PDU, as specified in section 3.2.5.1.4.

3.2.5.1.6 Processing Window Information Orders

Window Information Orders (section 2.2.1.3.1) inform the client of the following types of window events on the server:

- Creation of a new window.
- Updates on window properties for a new or existing window.
- Updates on icons for a new or existing window.
- Deletion of an existing window.

Upon receipt of a Window Information Order for a new window (the **FieldsPresentFlags** field of the **Hdr** contains the WINDOW_ORDER_STATE_NEW (0x10000000) flag, as specified in section 2.2.1.3.1.2.1), the client SHOULD create a new RAIL window locally. The client SHOULD store an association of the **WindowId** reported in the **Hdr** field with the local RAIL window.

Upon receipt of a Window Information Order for an existing window (the **FieldsPresentFlags** field of **Hdr** does not contain the WINDOW_ORDER_STATE_NEW (0x10000000) flag, as specified in section 2.2.1.3.1.2.1), the client SHOULD locate the local RAIL window that corresponds to the **WindowId** reported in the **Hdr** field and apply the specified updates to the RAIL window. If no such window can be found, the client SHOULD ignore the order.

Upon receipt of a Window Information Order for an icon or cached icon, as specified in sections 2.2.1.3.1.2.2 and 2.2.1.3.1.2.3, the client SHOULD locate the local RAIL window that corresponds to the **WindowId** reported in the **Hdr** field and apply the icon updates to the RAIL window. If no such window can be found, the client SHOULD ignore the order.

Upon receipt of a Window Information Order for a deleted window, as specified in section 2.2.1.3.1.2.4, the client SHOULD locate the local RAIL window that corresponds to the **WindowId** reported in the **Hdr** field and destroy it. If no such window can be found, the client SHOULD ignore the order.

3.2.5.1.7 Processing Notification Icon Orders

Notification Icon Information Orders (section 2.2.1.3.2) inform the client of the following types of notification icon events on the server:

- Creation of a new notification icon.
- Updates on properties for a new or existing notification icon.
- Deletion of an existing notification icon.

Upon receipt of a Notification Icon Order for a new notify icon (the **FieldsPresentFlags** field of **Hdr** contains the WINDOW_ORDER_STATE_NEW (0x10000000) flag, as specified in section 2.2.1.3.2.2.1), the client SHOULD create a new RAIL notification icon locally. The client SHOULD store an association of the **WindowId** and **NotifyIconId** reported in the **Hdr** field with the local notify icon.

Upon receipt of a Notification Icon Order for an existing notify icon (the FieldsPresentFlags field of **Hdr** does not contain the WINDOW_ORDER_STATE_NEW (0x10000000) flag, as specified in section 2.2.1.3.2.2.1), the client SHOULD locate the RAIL notify icon that corresponds to the WindowId and NotifyIconId reported in the **Hdr** field, and then apply the specified updates to the RAIL notify icon. If no such icon can be found, the client SHOULD ignore the Order.

Upon receipt of a Notification Icon Order for a deleted icon, as specified in section 2.2.1.3.2.2.2, the client SHOULD locate the local RAIL notify icon that corresponds to the WindowId and NotifyIconId reported in the **Hdr** field and destroy it. If no such icon can be found, the client SHOULD ignore the Order.

3.2.5.1.8 Processing Desktop Information Orders

Desktop Information Orders inform the client of events on the server that are not confined to a single window or notification icon. Processing of these orders is indicated as follows:

- Upon receipt of a Desktop Information Order, as specified in section 2.2.1.3.3.2.1, with the WINDOW_ORDER_FIELD_DESKTOP_ARC_BEGAN (0x00000008) and the

WINDOW_ORDER_FIELD_DESKTOP_HOOKED (0x00000002) flags set in the **Hdr** field, the client SHOULD discard all of the existing RAIL windows and Notify Icons and prepare for Window Orders (see sections 2.2.1.3.1.2.1 and 2.2.1.3.1.2.4) and Notify Icon Orders (see sections 2.2.1.3.1.2.2and 2.2.1.3.1.2.3) from the server.

- Upon receipt of a Desktop Information Order for a non-monitored desktop, as specified in section 2.2.1.3.3.2.2, the client SHOULD discard all of the existing RAIL windows and Notify Icons.
- Upon receipt of a Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_HOOKED (0x00000002) flag set in the **Hdr** field, the client SHOULD prepare for Window and Notify Icon Orders from the server.
- Upon receipt of a Desktop Information Order with the **NumWindowIds** and **WindowIds** fields present, the client SHOULD apply the specified Z-order of the server's windows to its local RAIL windows.
- Upon receipt of a Desktop Information Order with the **ActiveWindowId** field present, the client SHOULD activate the corresponding local RAIL window.

3.2.5.2 Static Virtual Channel Protocol

3.2.5.2.1 Initialization Messages

3.2.5.2.1.1 Sending Client Information PDU

The client information PDU is initialized as specified in section 2.2.2.2.2.

3.2.5.2.2 Program Launching Messages

3.2.5.2.2.1 Sending Execute PDU

As specified in section 2.2.2.3.1, the client SHOULD store the execute request to match execute requests with Execute Result PDUs from the server. For Server Execute Result PDU, see section 2.2.2.3.2.

3.2.5.2.2.2 Processing Execute Result PDU

The client SHOULD match the Execute Result PDU with a previously sent Execute PDU and report the results to the user.

3.2.5.2.3 Local Client System Parameters Update Messages

3.2.5.2.3.1 Sending System Parameters Update PDU

Initialized as specified in section 2.2.2.4.1, this PDU SHOULD be sent at the start of every remote applications integrated locally (RAIL) connection or reconnection and when a system parameter on the client changes its value.

3.2.5.2.4 Server System Parameters Update Messages

3.2.5.2.4.1 Processing Server System Parameters Update PDU

On receipt of this PDU, the client SHOULD update its system parameters to those reported by the server. This helps to maintain consistency between local client and remote server settings, which is an important aspect of the seamless windows experience.

3.2.5.2.5 Local Client Event Messages

Local Client Event Messages are Virtual Channel PDUs sent from the client to the server specifying user interactions with RAIL windows and notifications that cannot be captured and sent over the regular RDP channel.

3.2.5.2.5.1 Sending Activate PDU

The Activate PDU is sent by the client when a RAIL window is activated by a means other than clicking it, such as by pressing ALT+TAB.

Note Mouse clicks on the RAIL window are forwarded to the server via the RDP core protocol. The PDU is initialized as specified in section 2.2.2.6.1.

The **WindowId** field SHOULD be initialized to the ID of an existing window on the server that is associated with the local RAIL window being activated. The RAIL client SHOULD create this association during processing of the Window Information Order for new windows, as specified in section 2.2.1.3.1.2.1.

3.2.5.2.5.2 Sending System Menu PDU

The System Menu PDU is sent by the client when a RAIL window receives a command to display its system menu by a means other than clicking it, such as by right-clicking the **taskbar** icon for the window.

Note Mouse clicks in the RAIL window are forwarded to the server via the RDP core protocol. The PDU is initialized as specified in section 2.2.2.6.2.

The **WindowId** field SHOULD be initialized to the ID of an existing window on the server that is associated with the local RAIL window. The RAIL client SHOULD create this association during processing of the Window Information Order for new windows, as specified in section 2.2.1.3.1.2.1.

3.2.5.2.5.3 Sending System Command PDU

The System Command PDU is sent by the client when a RAIL window receives a system command by a means other than clicking it (for example, by pressing the Windows logo key+M to minimize the window, by clicking the Show Desktop button in the taskbar, or by selecting the system menu by pressing ALT+SPACE).

Note Mouse clicks in the RAIL window are forwarded to the server via the RDP core protocol. The PDU is initialized as specified in section 2.2.2.6.3.

The **WindowId** field SHOULD be initialized to the ID of an existing window on the server that is associated with the local RAIL window. The RAIL client SHOULD create this association during processing of the Window Information Order for new windows, as specified in section 2.2.1.3.1.2.1.

3.2.5.2.5.4 Sending Notify Event PDU

The Notify Event PDU is sent by the client when a remote applications integrated locally (RAIL) notify icon receives any user interaction via the keyboard or mouse. The PDU is initialized as specified in section 2.2.2.6.4.

The **WindowId** and **NotifyIconId** fields SHOULD be initialized to the ID of an existing Window and notification icon (respectively) on the server and associated with the local RAIL notification icon. The RAIL client SHOULD create this association during processing of the Notification Icon Information Order for new notify icons, as specified in section 2.2.1.3.2.2.1.

3.2.5.2.6 Language Bar Information PDUs

3.2.5.2.6.1 Sending Language Bar Information PDU

After initialization (as specified in section 2.2.2.9.1), this PDU SHOULD be sent from a client to a server just after sending the RAIL handshake (see section 2.2.2.2.1). This enables the server synchronize its language bar state with the client's.

This PDU MUST NOT be sent if the server does not support the Docked Language Bar RAIL capability (TS_RAIL_LEVEL_DOCKED_LANGBAR_SUPPORTED).

3.2.5.2.6.2 Processing Language Bar Information PDU

Upon receipt of this PDU, the client SHOULD update the status of its language using the Language Bar Information PDU.

3.2.5.2.7 Window Move Messages

Window Move Messages are generated by the server and client to enable the local move/size feature of RAIL.

3.2.5.2.7.1 Processing Min Max Info PDU

On receipt of the Min Max Info PDU, if the client supports local move/size, it SHOULD locate the local RAIL window that corresponds to the **WindowId** field and apply the specified window extents (**MaxWidth**, **MaxHeight**, **MaxPosX**, **MaxPosY**, **MinTrackWidth**, **MinTrackHeight**, **MaxTrackWidth**, and **MaxTrackHeight** fields) to it.

If no such RAIL window can be found, the client SHOULD ignore this PDU.

If the client does not support local move/size, it SHOULD ignore this PDU.

3.2.5.2.7.2 Processing Move-Size Start PDU

On receipt of the Move-Size Start PDU, if the client supports local move/size features, it SHOULD locate the local RAIL window that corresponds to the **WindowId** field and initiate a move/size of the local RAIL window by using the local Window Manager based on the **MoveSizeType** field. The client SHOULD also suppress forwarding of keyboard/mouse events to the server to maintain a local-only move/size of the RAIL window.

If no RAIL window can be found corresponding to **WindowId**, the client SHOULD ignore this PDU.

If the client does not support local move/size, it SHOULD ignore this PDU.

3.2.5.2.7.3 Sending Window Move PDU

If the client supports local move/size, it SHOULD send the Window Move PDU upon receiving a notification from the local window manager that a local move/size of a RAIL window has ended. The PDU is sent for keyboard–based moves and all resizes, and it is initialized as specified in section 2.2.2.7.4.

The **WindowId** field SHOULD be initialized to the ID of an existing window on the server that is associated with the local RAIL window. The RAIL client SHOULD create this association during processing of the Window Information Order for new windows, as specified in section 2.2.1.3.1.2.1.

If the client suppressed forwarding of keyboard/mouse events to the server during processing of the Move-Size Start PDU, it MUST resume the forwarding of these events to the server to allow the server to detect a move/size end of the remote window.

3.2.5.2.7.4 Processing Move-Size End PDU

Upon receipt of the Move-Size End PDU, if the client supports local move/size features, it SHOULD locate the local RAIL window that corresponds to the **WindowId** field and move it to the coordinates specified by the **TopLeftX** and **TopLeftY** fields. This ensures synchronization between the final positions of the corresponding moved/resized windows on the server and client.

If no RAIL window can be found corresponding to **WindowId**, the client SHOULD ignore this PDU.

If the client does not support local move/size, it SHOULD ignore this PDU.

3.2.5.2.8 Application ID Messages

3.2.5.2.8.1 Sending Client Get Application ID PDU

After being initialized as specified in section 2.2.2.6.5, this PDU MAY be sent from a client to a server after receiving a Windows Information Order containing the WINDOW_ORDER_STATE_NEW (0x10000000) flag.

3.2.5.2.8.2 Processing Server Get Application ID Response PDU

Upon receipt of this PDU, the client MAY <24>update the Application ID string of the Window matching the Windows ID received from the server.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Server Details

3.3.1 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 what is described in this document.

Note: It is possible to implement the following conceptual data by using a variety of techniques as long as the implementation produces external behavior that is consistent with what is described in this document.

3.3.1.1 Client Local Move/Size Ability Store

The Client Local Move/Size Ability store determines whether the client has the ability to support Local Move/Size in RAIL. This is communicated to the server by the TS_RAIL_CLIENTSTATUS_ALLOWLOCALMOVESIZE flag as part of Client Information PDU (see section 2.2.2.2.2).

3.3.1.2 Windowing Support Level

The windowing support level determines whether the client is capable of supporting Windowing Alternate Secondary Drawing Orders and the following flags: WINDOW_ORDER_FIELD_CLIENTAREASIZE, WINDOW_ORDER_FIELD_RPCONTENT, and WINDOW_ORDER_FIELD_ROOTPARENT. This is communicated to the server by the **WndSupportLevel** field, as part of the Window List Capability Set (section 2.2.1.1.2).

3.3.2 Timers

No timers are used.

3.3.3 Initialization

None.

3.3.4 Higher-Layer Triggered Events

No higher-layer triggered events are used.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Updates to RDP Core Protocol

3.3.5.1.1 Processing Client MCS Connect Initial PDU

The Client MCS Connect Initial PDU is processed by the server during the connection establishment phase, as specified in [MS-RDPBCGR].

3.3.5.1.2 Constructing Server MCS Connect Response PDU

This PDU is sent by the server in response to the Client MCS Connect Initial PDU, as specified in [MS-RDPBCGR].

3.3.5.1.3 Processing Client Info PDU

The Client Info PDU is processed by the server during the connection establishment phase, as specified in [MS-RDPBCGR].

If the flags field of the Info Packet (as specified in [MS-RDPBCGR], section 2.2.1.11.1.1) has the INFO_RAIL (0x00008000) flag set, it indicates that the client wants to start a remote applications integrated locally (RAIL) connection. If the server supports RAIL, it SHOULD indicate this by using the Demand Active PDU (see section 3.3.5.1.4).

3.3.5.1.4 Constructing Demand Active PDU

The Demand Active PDU is constructed by the server during the connection establishment phase, as specified in [MS-RDPBCGR] section 3.3.5.3.13.1.

If the client has requested support for remote applications integrated locally (RAIL) in the Client Info PDU (as specified in [MS-RDPBCGR] section 2.2.1.11), and the server supports RAIL, the server MUST specify two RAIL–specific capabilities in the capabilitySets field of the PDU: the Remote Programs Capability Set (section 2.2.1.1.1) and the Window List Capability Set (section 2.2.1.1.2).

The server MUST specify the number of icon caches supported by using the NumIconCaches and NumIconCacheEntries of the Window List Capability Set.

3.3.5.1.5 Processing Confirm Active PDU

The Confirm Active PDU is processed by the server, as specified in [MS-RDPBCGR] section 3.3.5.3.13.2.

If the client has requested support for remote applications integrated locally (RAIL) in the Client Info PDU (see section 3.2.5.1.3), and the server has indicated support for RAIL in the Demand Active PDU (see section 3.3.5.1.4), the server MUST verify that this PDU contains two RAIL-specific capabilities in the capabilitySets field: the Remote Programs Capability Set (section 2.2.1.1.1) and the Window List Capability Set (section 2.2.1.1.2). If it does not contain these capability sets, or the RailSupportLevel of the Remote Programs Capability Set is not set to at least TS_RAIL_LEVEL_SUPPORTED, or the WndSupportLevel of the Window List Capability Set is TS_WINDOW_LEVEL_NOT_SUPPORTED (0), the server MUST drop the connection.

The server MUST verify that the NumIconCaches and NumIconCacheEntries of the Window List Capability Set do not exceed the corresponding entries set by the server in the Demand Active PDU. <25> The server MUST also update its icon cache limits to those reported in NumIconCaches and NumIconCacheEntries.

3.3.5.1.6 Constructing Window Information Orders

The server generates Window Information Orders to inform the client of the following types of window events on the server:

- Creation of a new window.
- Updates on window properties for a new or existing window.
- Updates on icons for a new or existing window.
- Deletion of an existing window.

The Window Information Orders are constructed as specified in section 2.2.1.3.1.

3.3.5.1.7 Constructing Notification Icon Orders

The server generates Notification Icon Information Orders to inform the client of the following types of notification icon events on the server.

- Creation of a new notification icon.
- Updates on properties for a new or existing notification icon.
- Deletion of an existing notification icon.

The Notification Icon Orders are constructed as specified in section 2.2.1.3.2.

3.3.5.1.8 Constructing Desktop Information Orders

Desktop Information Orders are generated by the server to inform the client of events on the server that are not confined to a single window or notification icon. These events include the following:

- A client connects to the server that is actively monitoring a desktop. The server generates the following events in order:

  1. A Desktop Information Order (see section 2.2.1.3.3.2.1) with the WINDOW_ORDER_FIELD_DESKTOP_ARC_BEGAN (0x00000008) and the WINDOW_ORDER_FIELD_DESKTOP_HOOKED (0x00000002) flags set in the **Hdr** field to indicate that the synchronization has begun.

  2. After all orders specifying windows, icons, and the desktop are sent, the server generates a Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_ARC_COMPLETED (0x00000004) flag set to signal the end of synchronization data.

- A desktop switch occurred on the server causing the server to stop monitoring the current desktop and (optionally) start monitoring the new desktop. This is indicated by generating the following events in order.

  1. A Desktop Information Order for the non-monitored desktop (see section 2.2.1.3.3.2.2).

  2. A Desktop Information Order with the WINDOW_ORDER_FIELD_DESKTOP_HOOKED (0x00000002) flag set in the **Hdr** field. If the server is unable to monitor the new desktop, the server SHOULD NOT send this order.

- The number and/or Z-order of top-level windows on the server changes. This is indicated by generating a Desktop Information Order with the **NumWindowIds** and **WindowIds** fields present.
- The active window on the server changes. This is indicated by generating a Desktop Information Order with the **ActiveWindowId** field present.

3.3.5.2 Static Virtual Channel Protocol

3.3.5.2.1 Initialization Messages

3.3.5.2.1.1 Processing Client Information PDU

If the Flags field of the PDU contains the TS_RAIL_CLIENTSTATUS_ALLOWLOCALMOVESIZE(0x00000001) flag, the client supports Local Move/Size. If the server also supports Local Move/Size, it SHOULD record this fact and SHOULD send Move Messages to the client window when appropriate (see section 2.2.2.7.4).

3.3.5.2.2 Program Launching Messages

3.3.5.2.2.1 Processing Execute PDU

Upon receipt of this PDU, the server MUST start the application specified in the PDU on the server. The PDU is processed as specified in 2.2.2.3.2.

3.3.5.2.2.2 Sending Execute Result PDU

This PDU is sent in response to an Execute PDU from the client and is initialized as specified in section 2.2.2.3.2.

3.3.5.2.3 Local Client System Parameters Update Messages

3.3.5.2.3.1 Processing System Parameters Update PDU

Upon receipt of this PDU, the server SHOULD set its system parameters to those reported by the client. This helps applications running remotely to behave consistently with local user settings, which is an important aspect of the seamless Microsoft Windows® experience.

3.3.5.2.4 Server System Parameters Update Messages

3.3.5.2.4.1 Sending Server System Parameters Update PDU

This PDU is initialized as specified in section 2.2.2.5.1. This PDU SHOULD be sent at the start of every remote applications integrated locally (RAIL) connection/reconnection, and when a system parameter on the server changes its value.

3.3.5.2.5 Local Client Event Messages

3.3.5.2.5.1 Processing Activate PDU

Upon receipt of this PDU, the server SHOULD activate or deactivate the remote window whose ID is specified by **WindowId** and whose activation state is specified by the **Enabled** field.

If no such window exists, the server SHOULD ignore the PDU.

3.3.5.2.5.2 Processing System Menu PDU

On receipt of this PDU, the server SHOULD post a command to the remote window whose ID is specified by WindowId to display its system menu at the coordinates specified by the Left and Top fields.

If no such window exists, the server SHOULD ignore the PDU.

3.3.5.2.5.3 Processing System Command PDU

Upon receipt of this PDU, the server SHOULD post the system command specified by the Command field to the remote window whose ID is specified by **WindowId**.

If no such window exists, the server SHOULD ignore the PDU.

3.3.5.2.5.4 Processing Notify Event PDU

Upon receipt of this PDU, the server SHOULD post the message specified by the **Message** field to the remote notification icon specified by the **WindowId** and **NotifyIconId** fields.

If no such notify icon exists, the server SHOULD ignore the PDU.

3.3.5.2.5.5 Processing Language Bar Information PDU

Upon receipt of this PDU, the server MUST first send the status of its language bar to the client using the Language Bar Information PDU. The server MUST then adjust the server-side language bar to match the client's language bar status by making it either float or be docked.

3.3.5.2.6 Window Move Messages

The Window Move messages are generated by the server and client to enable the Local Move/Size feature of RAIL.

3.3.5.2.6.1 Sending Min Max Info PDU

This PDU is sent by the server when a user attempts to move or resize a local RAIL window and when the corresponding keyboard input or mouse input forwarded to the server causes the corresponding remote window to begin to move or resize. It is initialized as specified in section 2.2.2.7.1.

This PDU SHOULD be sent if the client and server both support local move/size features.

3.3.5.2.6.2 Sending Move/Size Start PDU

This PDU is sent by the server when a user attempts to move or resize a local RAIL window (for example, by dragging the window title with the mouse or resizing the window borders with the mouse), and the corresponding keyboard input or mouse input forwarded to the server causes the corresponding remote window to begin the move or resize. It is initialized as specified in section 2.2.2.7.2.

This PDU SHOULD be sent if the client and server both support local move/size features. It SHOULD be sent immediately after the Min Max Info PDU (see section 2.2.2.7.1).

3.3.5.2.6.3 Processing Window Move PDU

On receipt of the Client Window Move PDU section 2.2.2.7.4, the server SHOULD move the remote window specified by the WindowId field to the coordinates specified by the Left, Top, Right, and Bottom fields.

If no such Window exists, the server SHOULD ignore the PDU.

3.3.5.2.6.4 Sending Move/Size End PDU

This PDU is sent by the server when a user completes a move or resize of a local RAIL window (for example, by releasing the mouse button), and the corresponding keyboard input or mouse input forwarded to the server causes the corresponding remote window to complete the move or resize. It is initialized as specified in section 2.2.2.7.3.

This PDU SHOULD be sent if the client and server both support local move/size features.

3.3.5.2.7 Application ID Messages

3.3.5.2.7.1 Processing the Get Application ID PDU

Upon receipt of the Get Application ID PDU, the server MAY <26> retrieve the application id of the window whose window id is specified in the PDU.

If no such window exists, the server SHOULD ignore the PDU.

3.3.5.2.7.2 Sending the Get Application ID Response PDU

The Get Application ID Response PDU is sent in response to a Get Application ID PDU from the client and is initialized as specified in section 2.2.2.8.1.

3.3.6 Timer Events

No timer events are used.

3.3.7 Other Local Events

3.3.7.1 Sending Language Bar Information PDU

Upon receiving a notification from the server-side language bar indicating that its status was updated, the server MUST then send the updated status of its language bar to the client using the Language Bar Information PDU. This enables the client to stay in sync with the server.

4 Protocol Examples

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

4.1 Updates to the RDP Core Protocol

4.1.1 Windowing Alternate Secondary Drawing Orders

4.1.1.1 New or Existing Windows

The following is a network capture of a Window Information Order, sent when a new window is created on the server or when a property on a new or existing window is updated (as specified in 2.2.1.3.1.2.1).

```
00000000 2e 82 00 1e de 00 11 5e 00 03 00 00 00 00 00 .......^........
00000010 00 ef 34 00 03 04 00 02 36 00 43 00 3a 00 5c 00 ..4.....6.C.:.\.
00000020 57 00 69 00 6e 00 64 00 6f 00 77 00 73 00 5c 00 W.i.n.d.o.w.s.\.
00000030 73 00 79 00 73 00 74 00 65 00 6d 00 33 00 32 00 s.y.s.t.e.m.3.2.
00000040 5c 00 63 00 6d 00 64 00 2e 00 65 00 78 00 65 00 \.c.m.d...e.x.e.
00000050 00 00 00 00 98 04 00 00 00 00 00 00 98 04 00 00 ................
00000060 00 00 00 00 00 00 00 00 a0 00 00 00 18 00 00 00 ................
00000070 00 00 00 00 98 04 00 00 01 00 00 00 00 00 a0 00 ................
00000080 18 00

2e -> TS_WINDOW_ORDER_HEADER::Flags(1 Byte)
82 00 -> TS_WINDOW_ORDER_HEADER::OrderSize(2 Bytes)
1e de 00 11 -> TS_WINDOW_ORDER_HEADER::FieldsPresentFlags(4 Bytes)
5e 00 03 00 -> TS_WINDOW_ORDER_HEADER::WindowId(4 Bytes)
00 00 00 00 -> OwnerWindowId(4 Bytes)
00 00 ef 34 -> Style
00 03 04 00 -> ExtendedStyle
02 -> Show
00000010 00 ef 34 00 03 04 00 02 36 00 43 00 3a 00 5c 00 ..4.....6.C.:.\.
00000020 57 00 69 00 6e 00 64 00 6f 00 77 00 73 00 5c 00 W.i.n.d.o.w.s.\.
00000030 73 00 79 00 73 00 74 00 65 00 6d 00 33 00 32 00 s.y.s.t.e.m.3.2.
00000040 5c 00 63 00 6d 00 64 00 2e 00 65 00 78 00 65 00 \.c.m.d...e.x.e.
-> Title (C:\Windows\system32\cmd.exe)
00 00 00 00 -> ClientOffsetX (0)
98 04 00 00 -> ClientOffsetY (1176)
00 00 00 00 -> WindowOffsetX (0)
98 04 00 00 -> WindowOffsetY (1176)
00 00 00 00 -> WindowClientDeltaX (0)
00 00 00 00 -> WindowClientDeltaY (0)
a0 00 00 00 -> WindowWidth  (160)
18 00 00 00 -> WindowHeight (24)
00 00 00 00 -> VisibleOffsetX (0)
98 04 00 00 -> VisibleOffsetY (1176)
01 00 -> NumVisibilityRects (1)
00 00 00 00 a0 00 18 00 -> VisibilityRects (0,0,160,24)
```

4.1.1.2 Deleted Window

The following is a network capture of a Window Information Order, sent when an existing window is destroyed on the server (as specified in 2.2.1.3.1.2.4).

```
00000000 2e 0b 00 00 00 00 21 24 00 03 00......!$...

2e -> TS_WINDOW_ORDER_HEADER::Flags(1 Byte)
0b 00 -> TS_WINDOW_ORDER_HEADER::OrderSize(2 Bytes)
00 00 00 21 -> TS_WINDOW_ORDER_HEADER::FieldsPresentFlags(4 Bytes)
       (WINDOW_ORDER_TYPE_WINDOW | WINDOW_ORDER_STATE_DELETED )
24 00 03 00 -> WindowId(4 Bytes)
```

4.1.1.3 New or Existing Notification Icons

The following is a network capture of a Notification Icon Information Order, sent when a new notification icon is created on the server (as specified in 2.2.1.3.2.2.1).

```
00000000 2e 9d 04 01 00 00 52 8e 00 01 00 d2 9c 00 00 40 ......R........@
00000010 00 2a 20 0e 20 43 00 6f 00 6d 00 6d 00 75 00 6e .* . C.o.m.m.u.n
00000020 00 69 00 63 00 61 00 74 00 6f 00 72 00 20 00 2d .i.c.a.t.o.r. .-
00000030 00 20 00 4e 00 6f 00 74 00 20 00 73 00 69 00 67 . .N.o.t. .s.i.g
00000040 00 6e 00 65 00 64 00 20 00 69 00 6e 00 0e 20 2c .n.e.d. .i.n.. ,
00000050 20 00 00 02 20 10 00 10 00 40 00 00 04 fe 03 00  ... ....@......
00000060 00 fc 01 00 00 fc 01 00 00 c0 01 00 00 80 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 ................
00000090 00 80 01 00 00 f0 3f 00 00 f8 7f 00 00 00 00 00 ......?.........
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000b0 00 00 00 00 00 00 00 00 00 00 18 36 80 18 1e 38 7f ..........6...8.
000000c0 9c 19 35 96 ef 1a 3c b5 fe 1e 3e ad ee 15 34 8c ..5...<...>...4.
000000d0 8d 14 30 77 1b 00 00 00 00 00 00 00 00 00 00 00 ..0w............
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000f0 00 00 00 00 00 1c 38 85 17 1a 34 87 c6 17 31 9d ......8...4...1.
00000100 ff 0d 2c af ff 15 39 cd ff 1a 42 e3 ff 22 49 e0 ..,...9...B.."I.
00000110 fc 17 39 a4 c1 13 30 78 1b 00 00 00 00 00 00 00 ..9...0x........
……..(more icon data)


2e -> TS_NOTIFYICON_ORDER_HEADER::Flags(1 Byte)
9d 04 -> TS_NOTIFYICON_ORDER_HEADER::OrderSize(2 Bytes)
01 00 00 52 -> TS_NOTIFYICON_ORDER_HEADER::FieldsPresentFlags (4 Bytes)
     WINDOW_ORDER_TYPE_NOTIFY | WINDOW_ORDER_FIELD_NOTIFY_TIP |
     WINDOW_ORDER_STATE_NEW | WINDOW_ORDER_ICON)
8e 00 01 00 -> WindowId
d2 9c 00 00 -> NotifyIconId
0000000f 40
00000010 00 57 00 69 00 6e 00 64 00 6f 00 77 00 73 00 20
00000020 00 54 00 61 00 73 00 6b 00 20 00 4d 00 61 00 6e
00000030 00 61 00 67 00 65 00 72 00 00 00 02 10 10 00 10
00000040 00 6e 00 65 00 64 00 20 00 69 00 6e 00 0e 20 2c -> ToolTip (Communicator  - Not
signed in )

00000050 20 00 00 02 20 10 00 10 00 40 00 00 04 fe 03 00  ... ....@......
00000060 00 fc 01 00 00 fc 01 00 00 c0 01 00 00 80 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
00000080 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 ...............
00000090 00 80 01 00 00 f0 3f 00 00 f8 7f 00 00 00 00 ......?.........
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............
000000b0 00 00 00 00 00 00 00 00 00 18 36 80 18 1e 38 7f ..........6...8.
000000c0 9c 19 35 96 ef 1a 3c b5 fe 1e 3e ad ee 15 34 8c ..5...<...>...4.
000000d0 8d 14 30 77 1b 00 00 00 00 00 00 00 00 00 00 00 ..0w............
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000f0 00 00 00 00 00 1c 38 85 17 1a 34 87 c6 17 31 9d ......8...4...1.
00000100 ff 0d 2c af ff 15 39 cd ff 1a 42 e3 ff 22 49 e0 ..,...9...B.."I.
00000110 fc 17 39 a4 c1 13 30 78 1b 00 00 00 00 00 00 00 ..9...0x........    -> Icon
……..(more icon data)
```

Note The icon data is significantly large and accounts for the remainder of the order. For the sake of brevity, the icon information in the remaining bytes of the **orderSize** field has been truncated in this example.

4.1.1.4 Deleted Notification Icons

The following is a network capture of a Notification Icon Information Order, sent when an existing notification icon is deleted on the server (as specified in 2.2.1.3.2.2.2).

```
00000000 2e 0f 00 01 00 00 62 f4 01 03 00 00 00 00 00  .y....B........

2e -> TS_NOTIFYICON_ORDER_HEADER::Flags(1 Byte)
0f 00 -> TS_NOTIFYICON_ORDER_HEADER::OrderSize(2 Bytes)
01 00 00 62 -> TS_NOTIFYICON_ORDER_HEADER::FieldsPresentFlags (4 Bytes)
     WINDOW_ORDER_TYPE_NOTIFY | WINDOW_ORDER_STATE_DELETED |
     WINDOW_ORDER_FIELD_NOTIFY_TIP | WINDOW_ORDER_ICON)
f4 01 03 00 -> WindowId
00 00 00 00 -> NotifyIconId
```

4.1.1.5 Actively Monitored Desktop

The following is a network capture of an Actively Monitored Desktop packet (as specified in 2.2.1.3.3.2.1).

```
00000000 2e 14 00 30 00 00 04 a0 00 01 00 02 a0 00 01 00 ...0............


2e -> TS_DESKTOP_ORDER_HEADER::Flags
14 00 -> TS_DESKTOP_ORDER_HEADER::OrderSize
30 00 00 04 -> TS_DESKTOP_ORDER_HEADER::FieldsPresentFlags (0x4000030)
               (WINDOW_ORDER_TYPE_DESKTOP | WINDOW_ORDER_FIELD_DESKTOP_ZORDER
WINDOW_ORDER_FIELD_DESKTOP_ACTIVEWND )
a0 00 01 00 -> ActiveWindowId
02 -> NumWindowIds
66 00 02 00
a0 00 01 00 -> WindowIds
```

4.1.1.6 Non-monitored Desktop

The following is a network capture of a Non-Monitored Desktop packet (as specified in 2.2.1.3.3.2.2).

```
00000000 2e 07 00 01 00 00 04  ..........@.....

2e -> TS_DESKTOP_ORDER_HEADER::Flags
07 00 -> TS_DESKTOP_ORDER_HEADER::OrderSize
01 00 00 04 -> TS_DESKTOP_ORDER_HEADER::FieldsPresentFlags
               (WINDOW_ORDER_TYPE_DESKTOP | WINDOW_ORDER_FIELD_DESKTOP_NONE)
```

4.2 Initialization Messages

4.2.1 TS_RAIL_ORDER_HANDSHAKE

The following are network captures of the Filter Updated PDUs (TS_RAIL_ORDER_HANDSHAKE, as specified in 2.2.2.2.1).

Server to Client

```
00000000 05 00 08 00 71 17 00 00                         ....q...

05 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_HANDSHAKE (5)  (2 Bytes)
08 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 8                            (2 Bytes)
71 17 00 00 -> buildNumber                                            (4 Bytes)
```

Client to Server

```
00000000 05 00 08 00 71 17 00 00                         ....q...

05 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_HANDSHAKE (5)  (2 Bytes)
08 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 8                            (2 Bytes)
71 17 00 00 -> buildNumber                                              (4 Bytes)
```

4.2.2 TS_RAIL_ORDER_CLIENTSTATUS

The following is a network capture of the Client Caps PDU (TS_RAIL_ORDER_CLIENTSTATUS, as specified in 2.2.2.2.2).

```
00000000 0b 00 08 00 01 00 00 00                         ........

0b 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_CLIENTSTATUS (11)  (2 Bytes)
08 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 8                          (2 Bytes)
01 00 00 00 -> CLIENTCAPS                                             (4 Bytes)
```

4.3 Launching Messages

4.3.1 TS_RAIL_ORDER_EXEC

The following is a network capture of the Client Execute PDU (TS_RAIL_ORDER_EXEC, as specified in 2.2.2.3.1).

```
 00000000 01 00 5e 00 08 00 14 00 26 00 18 00 7c 00 7c 00 ..^.....&...|.|.
 00000010 69 00 65 00 78 00 70 00 6c 00 6f 00 72 00 65 00 i.e.x.p.l.o.r.e.
 00000020 66 00 3a 00 5c 00 77 00 69 00 6e 00 64 00 6f 00 f.:.\.w.i.n.d.o.
 00000030 77 00 73 00 5c 00 73 00 79 00 73 00 74 00 65 00 w.s.\.s.y.s.t.e.
 00000040 6d 00 33 00 32 00 77 00 77 00 77 00 2e 00 62 00 m.3.2.w.w.w...b.
 00000050 69 00 6e 00 67 00 2e 00 63 00 6f 00 6d 00 00 00 i.n.g...c.o.m...

Header:
01 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_EXEC (1) (2 Bytes)
5e 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 94                     (2 Bytes)
08 00 -> Flags            : TS_RAIL_EXEC_FLAG_EXPAND_ARGUMENTS  (2 Bytes)
14 00 -> ExeorFileLength  : 0x14                                (2 Bytes)
26 00 -> WorkingDirLength : 0x26                                (2 Bytes)
18 00 -> ArgumentsLength  : 0x18                                (2 Bytes)
7c 00 7c 00 69 00 65 00 78 00 70 00 6c 00 6f 00 72 00 65 00 -> ExeOrFile : ||iexplore  (20
Bytes)
66 00 3a 00 5c 00 77 00 69 00 6e 00 64 00 6f 00 77 00 73 00 5c 00 73 00 79 00 73 00 74 00 65
00 6d 00 33 00 32 00 -> WorkingDir: f:\windows\system32 (38 bytes)
77 00 77 00 77 00 2e 00 62 00 69 00 6e 00 67 00 2e 00 63 00 6f 00 6d 00 -> Arguments (24
bytes)
```

4.3.2 RAIL_ORDER_EXEC_RESULT

The following is a network capture of the Server Execute Result PDU (RAIL_ORDER_EXEC_RESULT, as specified in 2.2.2.3.2).

```
00000000 80 00 24 00 08 00 03 00 15 00 00 00 00 00 14 00 ..$.............
00000010 7c 00 7c 00 57 00 72 00 6f 00 6e 00 67 00 41 00 |.|.W.r.o.n.g.A.
00000020 70 00 70 00                                     p.p.

80 00 -> TS_RAIL_PDU_HEADER::orderType = RAIL_ORDER_EXEC_RESULT(128) (2 Bytes)
24 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 36                         (2 Bytes)
08 00 -> Flags           : TS_RAIL_EXEC_FLAG_EXPAND_ARGUMENTS  (2 Bytes)
03 00 -> ExecResult      : 3                                   (2 Bytes)
15 00 00 00 -> RawResult : 0x15                                (4 Bytes)
00 00 -> Padding         : 0                                   (2 Bytes)
14 00 -> ExeOrFileLength : 0x14       (2 Bytes)
7c 00 7c 00 57 00 72 00 6f 00 6e 00 67 00 41 00
70 00 70 00 : ExeOrFile : ||WrongApp                           (20 Bytes)
```

4.4 Local Client System Parameters Update Messages

4.4.1 TS_RAIL_ORDER_SYSPARAM

The following are network captures of the Client System Parameters Update PDU (TS_RAIL_ORDER_SYSPARAM, as specified in 2.2.2.4.1).

```
00000000 03 00 12 00 43 00 00 00 7e 00 00 00 02 00 00 00 ....C...~.......
00000010 00 00                                           ..

03 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_SYSPARAM(3) (2 Bytes)
12 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 18                        (2 Bytes)
43 00 00 00 -> SystemParam: SPI_SETHIGHCONTRAST                    (4 Bytes)
7e 00 00 00 -> Flags: 0x7e   (4 Bytes)
```

```
02 00 00 00 -> ColorSchemeLength: 2    (4 Bytes)
00 00 -> ColorScheme: 0                          (2 Bytes)
```

4.5 Local Client Event Messages

4.5.1 TS_RAIL_ORDER_ACTIVATE

The following is a network capture of the Client Activate PDU (TS_RAIL_ORDER_ACTIVATE, as specified in 2.2.2.6.1).

```
00000000 02 00 09 00 4e 01 01 00 01                      ....N....

02 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_ACTIVATE(2) (2 Bytes)
09 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 9                         (2 Bytes)
4e 01 01 00 -> WindowId:: 0x1014e    (4 Bytes)
01 -> Enabled      (1 Byte)
```

4.5.2 TS_RAIL_ORDER_SYSMENU

The following is a network capture of the Client System Menu PDU (TS_RAIL_ORDER_SYSMENU, as specified in 2.2.2.6.2).

```
00000000 0c 00 0c 00 22 01 09 00 a4 ff 4a 02             ....".....J.

0c 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_SYSMENU(12) (2 Bytes)
0c 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 12                        (2 Bytes)
22 01 09 00 -> WindowId:: 0x90122    (4 Bytes)
a4 ff -> Left      (2 Bytes)
4a 02 -> Top      (2 Bytes)
```

4.5.3 TS_RAIL_ORDER_SYSCOMMAND

The following is a network capture of the Client System Command PDU (TS_RAIL_ORDER_SYSCOMMAND, as specified in 2.2.2.6.3).

```
00000000 04 00 0a 00 52 00 02 00 20 f0                   ....R... .

04 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_SYSCOMMAND(4) (2 Bytes)
0a 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 10                        (2 Bytes)
52 00 02 00 -> WindowId:: 0x20052      (4 Bytes)
20 f0 -> Command      (2 Bytes)
```

4.5.4 TS_RAIL_ORDER_NOTIFY_EVENT

The following is a network capture of the Client Notify Event PDU (TS_RAIL_ORDER_NOTIFY_EVENT, as specified in 2.2.2.6.4).

```
00000000 06 00 10 00 aa 01 02 00 02 00 00 00 04 02 00 00 ................

06 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_NOTIFY_EVENT(6) (2 Bytes)
10 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 16                           (2 Bytes)
aa 01 02 00 -> WindowId               (4 Bytes)
02 00 00 00 -> NotifyIconId              (4 Bytes)
04 02 00 00 -> Message                      (4 Bytes)
```

4.5.5 TS_RAIL_ORDER_LANGBARINFO

The following is a network capture of the Language Bar Information PDU (TS_RAIL_ORDER_LANGBARINFO, as specified in 2.2.2.9.1).

```
0D 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_LANGBARINFO (13) (2 Bytes)
08 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 8                            (2 Bytes)
01 00 00 00 -> LanguageBarStatus:: 0x00000001                         (4 Bytes)
```

4.5.6 TS_RAIL_ORDER_GET_APPID_REQ

The following is a network capture of the Client Get Application ID PDU (TS_RAIL_ORDER_GET_APPID_REQ), as specified in section 2.2.2.6.5.

```
00000000 0E 00 08 00 52 00 02 00                    ....R...
0E 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_GET_APPID_REQ (14) (2 Bytes)
08 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 8                                (2 Bytes)
52 00 02 00 -> WindowId:: 0x20052                                         (4 Bytes)
```

4.5.7 TS_RAIL_ORDER_GET_APPID_RESP

The following is a network capture of the Server Get Application ID Response PDU (TS_RAIL_ORDER_GET_APPID_RESP), as specified in section 2.2.2.8.1.

```
00000000 0F 00 08 20 52 00 02 00 6d 00 69 00 63 00 72 00 ....R...m.i.c.r.
00000010 6f 00 73 00 6f 00 66 00 74 00 2e 00 77 00 69 00 o.s.o.f.t...w.i.
00000020 6e 00 64 00 6f 00 77 00 73 00 2e 00 6e 00 6f 00 n.d.o.w.s...n.o.
00000030 74 00 65 00 70 00 61 00 64 00 00 00 00 00 00 00 t.e.p.a.d.......
                  00000040 00 ...
00000200 00 00 00 00 00 00 00 00                        ........

0F 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_GET_APPID_RESP (15) (2 Bytes)
08 20 -> TS_RAIL_PDU_HEADER::cbOrder   = 520                              (2 Bytes)
52 00 02 00 -> WindowId:: 0x1014e                                         (4 Bytes)
6d 00 69 00 63 00 72 00 6f 00 73 00 6f 00 66 00 74 00 2e 00 77 00 69 00 6e 00 64 00
6f 00 77 00 73 00 2e 00 6e 00 6f 00 74 00 65 00 70 00 61 00 64 00 00 ... -> ApplicationId::
microsoft.windows.notepad      (512 Bytes)
```

4.6 Window Move Messages

4.6.1 TS_RAIL_ORDER_WINDOWMOVE

The following is a network capture of the Client Window Move PDU (TS_RAIL_ORDER_WINDOWMOVE, as specified in 2.2.2.7.4).

```
00000000 08 00 10 00 20 00 02 00 09 03 00 01 db 05 88 01 .... ............

08 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_WINDOWMOVE(8) (2 Bytes)
10 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 16                        (2 Bytes)
20 00 02 00 -> WindowId                    (4 Bytes)
09 03 -> Left(2 Bytes)
00 01 -> Top(2 Bytes)
db 05 -> Right(2 Bytes)
88 01 -> Bottom(2 Bytes)
```

4.6.2 TS_RAIL_ORDER_LOCALMOVESIZE

The following is a network capture of the Server Local Move-Size PDU (TS_RAIL_ORDER_LOCALMOVESIZE, as specified in 2.2.2.7.3).

```
00000000 09 00 10 00 94 00 01 00 01 00 08 00 2c 05 e9 03 ............,...

09 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_LOCALMOVESIZE(9) (2 Bytes)
10 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 16                        (2 Bytes)
94 00 01 00 -> WindowId                    (4 Bytes)
01 00 -> IsMoveSizeStart   (2 Bytes)
08 00 -> MoveSizeType   (2 Bytes)
2c 05 -> PosX   (2 Bytes)
e9 03 -> PosY   (2 Bytes)
```

4.6.3 TS_RAIL_ORDER_MINMAXINFO

The following is a network capture of the Server Min Max Info PDU (TS_RAIL_ORDER_MINMAXINFO, as specified in 2.2.2.7.1).

```
*00000000 0a 00 18 00 94 00 01 00 48 06 b8 04 00 00 00 00 ........H.......
*00000010 70 00 1b 00 4c 06 bc 04                         p...L...

0a 00 -> TS_RAIL_PDU_HEADER::orderType = TS_RAIL_ORDER_MINMAXINFO(10) (2 Bytes)
18 00 -> TS_RAIL_PDU_HEADER::cbOrder   = 24                        (2 Bytes)
94 00 01 00 -> WindowId                    (4 Bytes)
48 06 -> MaxWidth   (2 Bytes)
b8 04 -> MaxHeight   (2 Bytes)
00 00 -> MaxPosX   (2 Bytes)
00 00 -> MaxPosY   (2 Bytes)
70 00 -> MinTrackWidth   (2 Bytes)
1b 00 -> MinTrackHeight   (2 Bytes)
4c 06 -> MaxTrackWidth   (2 Bytes)
bc 04 -> MaxTrackHeight   (2 Bytes)
```

5 Security

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

5.1 Security Considerations for Implementers

There are no security considerations for Remote Desktop Protocol: Remote Programs Virtual Channel Extension messages because all traffic is secured by the underlying Remote Desktop Protocol core protocol. For an overview of the implemented security-related mechanisms, see [MS-RDPBCGR] section 5.

5.2 Index of Security Parameters

There are no security parameters in the Remote Desktop Protocol: Remote Programs Virtual Channel Extension.

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 Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

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

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

<1> Section 2.2.1.1.1: Microsoft implementations set TS_RAIL_LEVEL_SUPPORTED to 1 in the following versions of Windows: Windows Server 2008, Windows Server 2008 R2, Windows 7 Enterprise, Windows 7 Enterprise N, Windows 7 Ultimate, Windows 7 Ultimate N.

Microsoft implementations set TS_RAIL_LEVEL_DOCKED_LANGBAR_SUPPORTED to 1 in the following versions of Windows: Windows Server 2008 R2, Windows 7 Enterprise, Windows 7 Enterprise N, Windows 7 Ultimate, Windows 7 Ultimate N.

Microsoft implementations set TS_RAIL_LEVEL_SUPPORTED to 0 on other versions when that capability is sent and the server does not support Remote Programs.

<2> Section 2.2.1.1.2: Only Windows 7 and Windows Server 2008 R2 send the TS_WINDOW_LEVEL_SUPPORTED_EX value to the client instead of the TS_WINDOW_LEVEL_SUPPORTED value.

<3> Section 2.2.1.3.1.2.1: This flag is not set in any Windows server implementation.

<4> Section 2.2.1.3.1.2.1: This flag is not set in any Windows server implementation.

<5> Section 2.2.1.3.1.2.1: This flag is not set in any Windows server implementation.

<6> Section 2.2.1.3.1.2.2: Windows applications display large icons in elements such as the Alt-Tab dialog box and on the desktop, and place small icons in elements such as the window's title bar and taskbar buttons.

<7> Section 2.2.1.3.1.2.3: Windows applications display large icons in elements such as the Alt-Tab dialog box and on the desktop, and place small icons in elements such as the window's title bar and taskbar buttons.

<8> Section 2.2.1.3.2.2.1: The WINDOW_ORDER_CACHED_ICON flag is not set in Windows 7 and Windows Server 2008 R2 implementations.

<9> Section 2.2.1.3.2.2.3: Microsoft implementations set minimum value to 10000 (10 seconds) and the maximum value to 30000 (30 seconds).

<10> Section 2.2.2.3.1: The length of the **ArgumentsLen** field is set to a maximum of 520 bytes on Windows Vista and Windows Server 2008, and to a maximum of 16000 bytes on Windows 7 and Windows Server 2008 R2.

<11> Section 2.2.2.3.2: This contains a Win32 error code. For more information, see [MS-ERREF].

<12> Section 2.2.2.4.2: Sets the High-Contrast parameters using the Win32 API. For more information, see [MSDN-HIGHCONTRAST].

<13> Section 2.2.2.4.2: Uses the Windows–specific name of the color scheme.

<14> Section 2.2.2.5.1: This system parameter is supported only on the following versions of Windows: Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2. For more information, see ([MSDN-SysParamsInfo]).

<15> Section 2.2.2.6.5: Only Windows 7 and Windows Server 2008 R2 use the Application ID string to identify and group windows.

<16> Section 2.2.2.8.1: Only Windows 7 and Windows Server 2008 R2 use the Application ID string to identify and group windows.

<17> Section 2.2.2.9.1: This option is available on Windows 7 and Windows Server 2008 R2 only.

<18> Section 2.2.2.9.1: This option is available on Windows 7 and Windows Server 2008 R2 only.

<19> Section 2.2.2.9.1: This option is available on Windows Server 2008 R2 only.

<20> Section 2.2.2.9.1: This option is available on Windows Server 2008 R2 only.

<21> Section 3.1.2: Microsoft implementations use 30 seconds as the time-out value.

<22> Section 3.1.5.2: Windows implementations ignore any incompatibility resulting from checking the **buildNumber** field between the sender and the receiver.

<23> Section 3.2.5.1.1: Windows implementations use RAIL as the name of the virtual channel.

<24> Section 3.2.5.2.8.2: Only Windows 7 and Windows Server 2008 R2 use the Application ID string to identify and group windows.

<25> Section 3.3.5.1.5: In Windows implementations the **NumIconCaches** and **NumCacheEntries** fields for each cache are set to 0 if the values in the Windows List Capability Sets exceed the corresponding entries set in the server cache.

<26> Section 3.3.5.2.7.1: Windows 7 and Windows Server 2008 R2 use the Application ID string to identify and group windows.

7 Change Tracking

This section identifies changes that were made to the [MS-RDPERP] protocol document between the May 2011 and June 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
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

8 Index

T

U

V

W