[MS-RDPBCGR]:
Remote Desktop Protocol:
Basic Connectivity and Graphics Remoting Specification

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.1	Minor	Minor technical content changes.
07/20/2007	1.2	Minor	Made technical and editorial changes based on

Release: Thursday, February 7, 2008

Date	Revision History	Revision Class	Comments
			feedback.
08/10/2007	1.3	Minor	Updated content based on feedback.
09/28/2007	1.4	Minor	Made technical and editorial changes based on feedback.
10/23/2007	1.4.1	Editorial	Revised and edited the technical content.
11/30/2007	1.5	Minor	Made technical and editorial changes based on feedback.
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification is designed to facilitate user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input from the user to the remote computer, where it may be injected locally. RDP also provides an extensible transport mechanism which allows specialized communication to take place between components on the user computer and components running on the remote computer.

1.1 Glossary

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

Stock Keeping Unit (SKU)

The following terms are specific to this document:

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

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-CSSP] Microsoft Corporation, "Credential Security Support Provider (CredSSP) Protocol Specification", March 2007.

[MS-RDPEA] Microsoft Corporation, "Remote Desktop Protocol: Audio Output Virtual Channel Extension", September 2007.

[MS-RDPEGDI] Microsoft Corporation, "Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions", June 2007.

[MS-RDPELE] Microsoft Corporation, "Remote Desktop Protocol: Licensing Extension", September 2007.

[MS-RDPERP] Microsoft Corporation, "Remote Desktop Protocol: Remote Programs Virtual Channel Extension", July 2007.

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

[RFC2118] Pall, G., "Microsoft Point-To-Point Compression (MPPC) Protocol," RFC 2118, March 1997, http://www.ietf.org/rfc/rfc2118.txt

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

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[SSL3] Netscape, "SSL 3.0 Specification", http://wp.netscape.com/eng/ssl3/

If you have any trouble finding [SSL3], please check here.

[T123] ITU-T, "Network-Specific Data Protocol Stacks for Multimedia Conferencing", Recommendation T.123, May 1999, http://www.itu.int/rec/T-REC-T.123/en

Note There is a charge to download the specification.

[T124] ITU-T, "Generic Conference Control", Recommendation T.124, February 1998, http://www.itu.int/rec/T-REC-T.124/en

Note There is a charge to download the specification.

[T125] ITU-T, "Multipoint Communication Service Protocol Specification", Recommendation T.125, February 1998, http://www.itu.int/rec/T-REC-T.125-199802-I/en

Note There is a charge to download the specification.

[T128] ITU-T, "Multipoint Application Sharing", Recommendation T.128, February 1998, http://www.itu.int/rec/T-REC-T.128-199802-I/en

Note There is a charge to download the specification.

[X224] ITU-T, "Information technology - Open Systems Interconnection - Protocol for Providing the Connection-Mode Transport Service", Recommendation X.224, November 1995, http://www.itu.int/rec/T-REC-X.224-199511-I/en

Note There is a charge to download the specification.

1.2.2 Informative References

[ERRTRANS] Microsoft Corporation, "How to Translate NTSTATUS Error Codes to Message Strings", June 2005, http://support.microsoft.com/kb/259693

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

[MSDN-CP] Microsoft Corporation, "Code Page Identifiers", http://msdn2.microsoft.com/en-us/library/ms776446.aspx

If you have any trouble finding [MSDN-CP], please check here.

[MSDN-SCHANNEL] Microsoft Corporation, "Creating a Secure Connection Using Schannel", http://msdn2.microsoft.com/en-us/library/aa374782.aspx

[MSFT-SDLBTS] Microsoft Corporation, "Session Directory and Load Balancing Using Terminal Server", September 2002, http://www.microsoft.com/windowsserver2003/techinfo/overview/sessiondirectory.mspx

1.3 Protocol Overview (Synopsis)

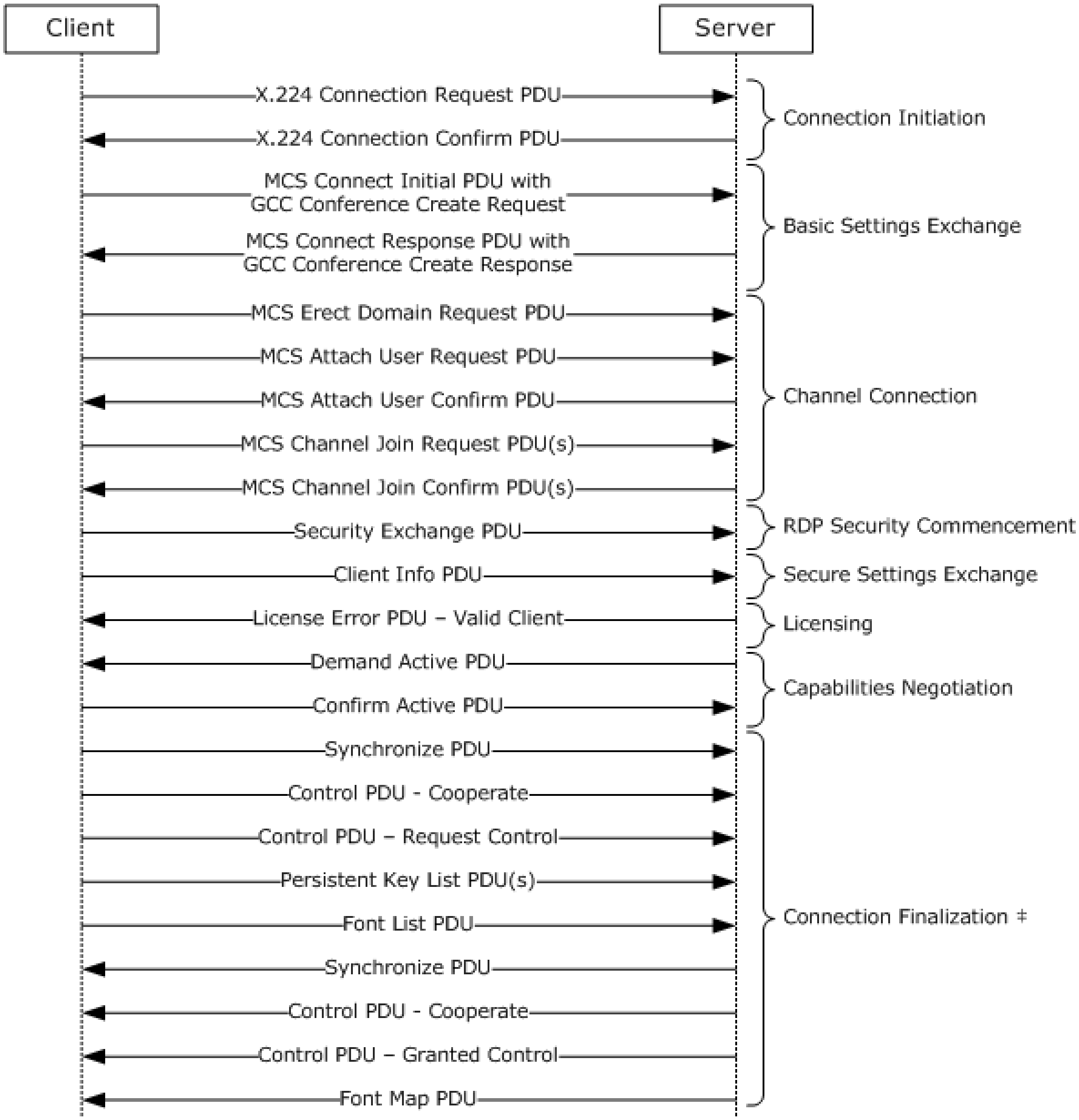
The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting is designed to facilitate user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input from the user to the remote computer, where it may be injected locally. This protocol also provides an extensible transport mechanism which allows specialized communication to take place between components on the user computer and components running on the remote computer.

The following subsections present overviews of the protocol operation as well as sequencing information.

1.3.1 Message Flows

1.3.1.1 Standard Connection Sequence

The goal of the standard connection sequence (Figure 1) is to exchange client and server settings and to negotiate common settings to use for the duration of the connection so that input, graphics and other data can be exchanged and processed between client and server.


Client
Server
X.224 Connection Request PDU
X.224 Connection Confirm PDU
MCS Connect Initial PDU with GCC Conference Create Request
MCS Connect Response PDU with GCC Conference Create Response
MCS Erect Domain Request PDU
MCS Attach User Request PDU
MCS Attach User Confirm PDU
MCS Channel Join Request PDU(s)
MCS Channel Join Confirm PDU(s)
Security Exchange PDU
Client Info PDU
License Error PDU – Valid Client
Demand Active PDU
Confirm Active PDU
Synchronize PDU
Control PDU - Cooperate
Control PDU – Request Control
Persistent Key List PDU(s)
Font List PDU
Synchronize PDU
Control PDU - Cooperate
Control PDU – Granted Control
Font Map PDU
Connection Initiation
Basic Settings Exchange
Channel Connection
RDP Security Commencement
Secure Settings Exchange
Licensing
Capabilities Negotiation
Connection Finalization ‡

‡All PDU's in the client-to-server direction must be sent in the specified order and all PDU's in the server-to-client direction must be sent in the specified order. However, there is no requirement that client-to-server PDU's be sent before server-to-client PDU's. PDU's may be sent concurrently as long as the sequencing in either direction is maintained.

Figure 1: Remote Desktop Protocol (RDP) connection initialization sequence

The connection sequence can be broken up into seven distinct phases:

1. Connection Initiation: The client initiates the connection by sending the server an X.224 Connection Request PDU (class 0). The server responds with an X.224 Connection Confirm PDU (class 0).

From this point, all subsequent data sent between client and server is wrapped in an *X.224 Data* Protocol Data Unit (PDU).

2. Basic Settings Exchange: Basic settings are exchanged between the client and server by using the MCS Connect Initial and MCS Connect Response PDUs. The Connect Initial PDU contains a GCC Conference Create Request, while the Connect Response PDU contains a GCC Conference Create Response.

   These two Generic Conference Control (GCC) packets contain concatenated blocks of settings data (such as core data, security data and network data) which are read by client and server.

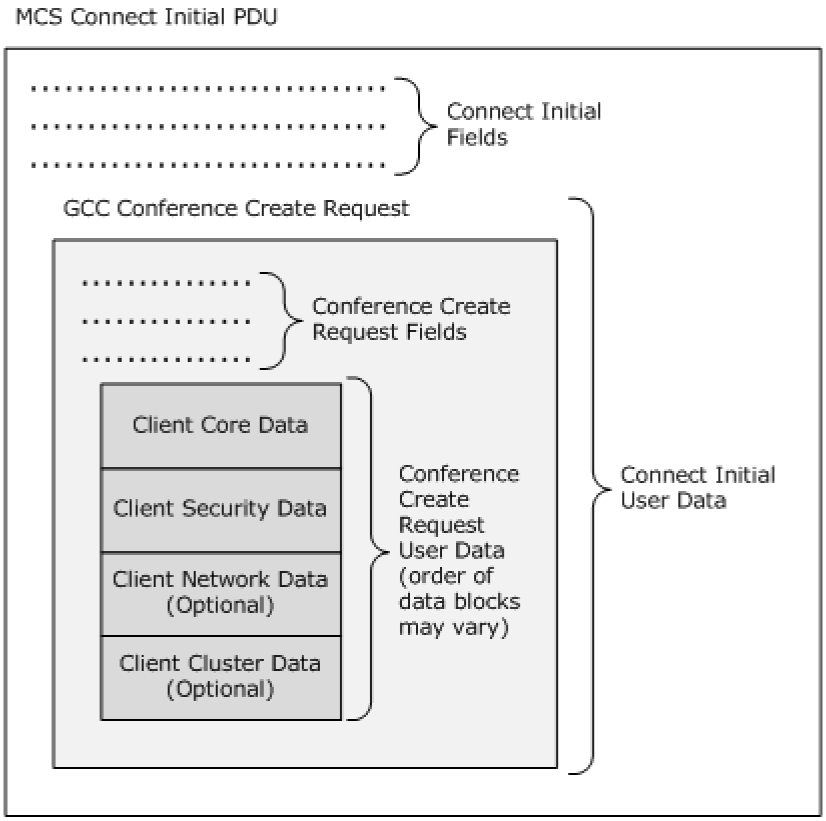

MCS Connect Initial PDU
Connect Initial Fields
GCC Conference Create Request
Conference Create Request Fields
Client Core Data
Client Security Data
Client Network Data (Optional)
Client Cluster Data (Optional)
Conference Create Request User Data (order of data blocks may vary)
Connect Initial User Data

Figure 2: Basic settings exchange PDUs

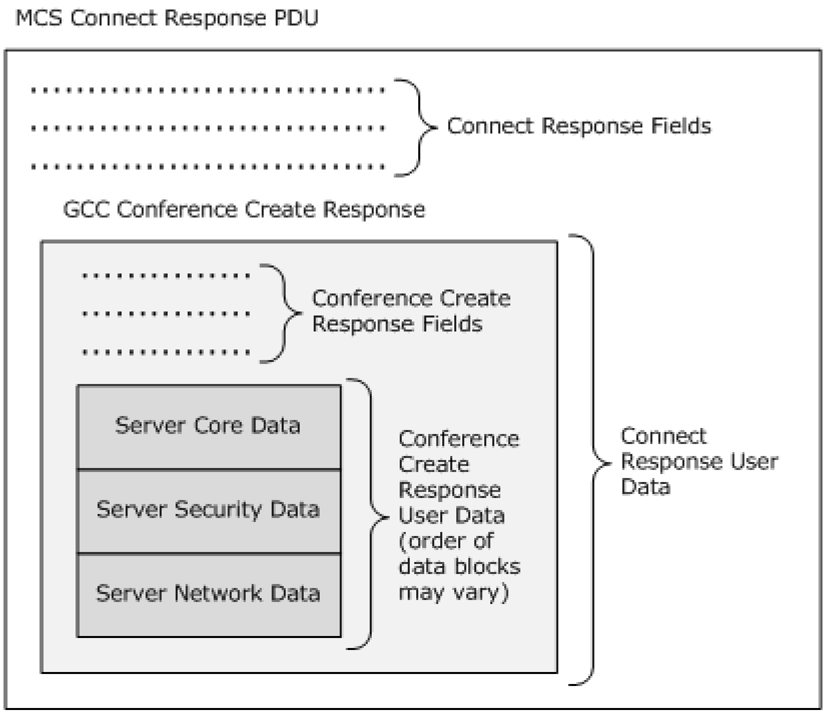

MCS Connect Response PDU
Connect Response Fields
GCC Conference Create Response
Conference Create Response Fields
Server Core Data
Server Security Data
Server Network Data
Conference Create Response User Data (order of data blocks may vary)
Connect Response User Data

Figure 3: MCS connect response PDU

3. Channel Connection: The client sends an MCS Erect Domain Request PDU, followed by an MCS Attach User Request PDU to attach the primary user identity to the MCS domain. The server responds with an MCS Attach User Response PDU containing the user channel ID. The client then proceeds to join the user channel, the input/output (I/O) channel and all of the static virtual channels (the I/O and static virtual channel IDs are obtained from the data embedded in the GCC packets) by using multiple MCS Channel Join Request PDUs. The server confirms each channel with an MCS Channel Join Confirm PDU. (The client only sends a Channel Join Request after it has received the Channel Join Confirm for the previously sent request.)

   From this point, all subsequent data sent from the client to the server is wrapped in an *MCS Send Data Request* PDU, while data sent from the server to the client is wrapped in an *MCS Send Data Indication* PDU. This is in addition to the data being wrapped by an *X.224 Data* PDU.

4. RDP Security Commencement: If standard RDP security methods are being employed and encryption is in force (this is determined by examining the data embedded in the GCC Conference Create Response packet) then the client sends a Security Exchange PDU containing an encrypted 32-byte random number to the server. This random number is encrypted with the public key of the server (the server's public key, as well as a 32-byte server-generated random number, are both obtained from the data embedded in the GCC Conference Create Response packet). The client and server then utilize the two 32-byte random numbers to generate session keys which are used to encrypt and validate the integrity of subsequent RDP traffic.

   From this point, all subsequent RDP traffic can be encrypted and a security header is included with the data if encryption is in force (the Client Info and licensing PDUs are an exception in that they always have a security header). The Security Header follows the X.224 and MCS Headers and indicates whether the attached data is encrypted. Even if encryption is in force server-to-client traffic may not always be encrypted, while client-to-server traffic will always be encrypted by Microsoft RDP implementations (encryption of licensing PDUs is optional, however).

5. Secure Settings Exchange: Secure client data (such as the username, password and auto-reconnect cookie) is sent to the server using the Client Info PDU.

6. Licensing: The goal of the licensing exchange is to transfer a license from the server to the client. The client should store this license and on subsequent connections send the license to the server for validation. However, in some situations the client may not be issued a license to store. In effect, the packets exchanged during this phase of the protocol depend on the licensing mechanisms employed by the server. Within the context of this document we will assume that the client will not be issued a license to store. For details regarding more advanced licensing scenarios that take place during the Licensing Phase, see [MS-RDPELE].

7. Capabilities Negotiation: The server sends the set of capabilities it supports to the client in a Demand Active PDU. The client responds with its capabilities by sending a Confirm Active PDU.

8. Connection Finalization: The client and server send PDUs to finalize the connection details. The client-to-server and server-to-client PDUs exchanged during this phase may be sent concurrently as long as the sequencing in either direction is maintained (there are no cross-dependencies between any of the client-to-server and server-to-client PDUs). After the client receives the Font Map PDU it can start sending mouse and keyboard input to the server, and upon receipt of the Font List PDU the server can start sending graphics output to the client.

Besides input and graphics data, other data that can be exchanged between client and server after the connection has been finalized includes connection management information and virtual channel messages (exchanged between client-side plug-ins and server-side applications).

1.3.1.2 Security-Enhanced Connection Sequence

The standard connection sequence does not provide any mechanisms which ensure that the identity of the server is authenticated, and as a result it is vulnerable to man-in-the-middle attacks (these attacks can compromise the confidentiality of the data sent between client and server).

The goal of the security-enhanced connection sequence is to provide an extensible mechanism within RDP so that well-known and proven security protocols (such as Secure Socket Layer (SSL) or Kerberos) can be used to provide server authentication and to wrap RDP traffic. There are two variations of the security-enhanced connection sequence. The negotiation-based approach aims to provide backward-compatibility with previous RDP implementations, while the Direct Approach favors more rigorous security over interoperability.

Negotiation-Based Approach: The client advertises the security packages which it supports (by appending a negotiation request structure to the X.224 Connection Request PDU) and the server selects the package to use (by appending a negotiation response structure to the X.224 Connection Confirm PDU). After the client receives the X.224 Connection Confirm PDU the negotiated security package is executed and used to secure all subsequent RDP traffic.

Direct Approach: Instead of negotiating a security package, the client and server immediately execute a pre-determined security protocol (for example, the CredSSP Protocol) prior to any RDP traffic being exchanged on the wire. This approach results in all RDP traffic being secured using the hard-coded security package. However, it also has the disadvantage of not working with servers that only utilize the standard connection sequence, as those servers expect an X.224 Connection Request PDU as the first packet.

For more details about Enhanced RDP Security, see section 5.4.

1.3.1.3 Deactivation-Reactivation Sequence

After the connection sequence has run to completion, the server may determine that the client needs to be connected to a waiting, disconnected session. To accomplish this task the server signals the client with a Deactivate All PDU. A Deactivate All PDU implies that the connection will be dropped or that a capability renegotiation will occur. If a capability renegotiation needs to be performed then the server will re-execute the connection sequence, starting with the Demand Active PDU (the Capability Negotiation and Connection Finalization phases as described in section 1.3.1.1) but excluding the Persistent Key List PDU.

1.3.1.4 Disconnection Sequences

1.3.1.4.1 User-Initiated on Client

The user can initiate a client-side disconnect by closing the RDP client application. To implement this type of disconnection the client sends the server a Shutdown Request PDU. The server will deny this request and send the client a Shutdown Request Denied PDU. At this point the client behavior is implementation-dependent. The Microsoft RDP client displays a dialog box specifying that the session will be disconnected. If the user chooses to disconnect, the client sends the server an MCS Disconnect Provider Ultimatum PDU (with the reason code set to "user requested") and closes the connection.

1.3.1.4.2 User-Initiated on Server

The user can initiate a server-side disconnect by ending the remote session (or application) running on the server. To implement this type of disconnection, the server first sends the client a Deactivate All PDU to indicate that the share is being disabled. This PDU is followed by an MCS Disconnect Provider Ultimatum PDU (with the reason code set to "user requested"). At this point the server closes the connection.

1.3.1.4.3 Administrator-Initiated on Server

The administrator of a server can force a user to be logged off from their session or disconnect sessions outside of the user's control. To implement this type of disconnection, the server first sends the client a Deactivate All PDU to indicate that the share is being disabled. This PDU is followed by an MCS Disconnect Provider Ultimatum PDU (with the reason code set to "provider initiated"). At this point the server closes the connection.

1.3.1.5 Automatic Reconnection

The automatic reconnection feature allows a client to reconnect to an existing session (after a short-term network failure has occurred) without having to resend the user's credentials to the server.

After a successful log on, the server sends the client an "auto-reconnect cookie" in the Save Session Info PDU. This cookie is bound to the current user's session and is stored by the client. In the case of a disconnection due to a network error, the client can try to automatically reconnect to the server. If it can connect, it sends a cryptographically modified version of the cookie to the server in the Client Info PDU (the Secure Settings Exchange phase of the connection sequence, as specified in section 1.3.1.1). The server uses the modified cookie to confirm that the client requesting auto-reconnection is the last client that was connected to the session. If this check passes, then the client is automatically connected to the desired session upon completion of the connection sequence.

The auto-reconnect cookie associated with a given session is flushed and regenerated whenever a client connects to the session or the session is reset. This ensures that if a different client connects to the session, then any previous clients that were connected can no longer use the auto-reconnect mechanism to connect. Furthermore, the server invalidates and updates the cookie at hourly intervals, sending the new cookie to the client in the Save Session Info PDU.

1.3.2 Server Error Reporting

A server can send detailed error codes to a client by using the Set Error Info PDU (the client must indicate during the Basic Settings Exchange phase of the connection sequence, as specified in section 1.3.1.1, that it supports this PDU). This PDU can be sent when a phase in the connection sequence fails or when the client is about to be disconnected. These error codes allow the client to give much clearer failure explanations to the user.

1.3.3 Static Virtual Channels

Static virtual channels allow lossless communication between client and server components over the main RDP data connection. Virtual channel data is application-specific and opaque to RDP. A maximum of 30 static virtual channels can be created at connection time.

The list of desired virtual channels is requested and confirmed during the Basic Settings Exchange phase of the connection sequence (as specified in section 1.3.1.1) and the endpoints are joined during the Channel Connection phase (as specified in section 1.3.1.1). Once joined, the client and server endpoints should be prevented from exchanging data until the connection sequence has completed.

Static virtual channel data must be broken up into chunks of up to 1600 bytes in size before being transmitted (this size does not include RDP headers). Each virtual channel acts as an independent data stream. The client and server examine the data received on each virtual channel and route the data stream to the appropriate endpoint for further processing. A particular client or server implementation can decide whether to pass on individual chunks of data as they are received, or to assemble the separate chunks of data into a complete block before passing it on to the endpoint.

1.3.4 Data Compression

RDP uses a bulk compressor to compress virtual channel data and some data in PDUs sent from server to client. Capability advertising for various versions of the bulk compressor occurs in the Client Info PDU (the Secure Settings Exchange phase of the connection sequence, as specified in section 1.3.1.1).

One version of the bulk compressor is based directly on the Microsoft Point-To-Point Compression (MPPC) Protocol and uses an 8 KB history buffer. A more advanced version of the compressor is derived from the same MPPC Protocol, but uses a 64 KB history buffer and modified Huffman-style encoding rules.

Besides employing bulk compression for generic data, RDP also uses variations of run length encoding (RLE) rules to implement compression of bitmap data sent from server to client. All clients should be able to decompress compressed bitmap data—this capability is not negotiable.

1.3.5 Keyboard and Mouse Input

The client sends mouse and keyboard input PDUs in two flavors: Slow-Path and Fast-Path. Slow-Path is similar to T.128 input formats for input PDUs, with some modifications for RDP input requirements. Fast-Path was introduced to take advantage of the fact that in RDP there are no extended Multipoint Communication Services (MCS) topologies, just one top-level node and one leaf-node per socket. Fast-Path also uses reduced or removed headers and alternate bytestream-orientated encoding formats to reduce bandwidth and CPU cycles for encode and decode.

Client-to-server Input Event PDUs convey keyboard and mouse data to the server so that it can inject input as needed. The client can also periodically synchronize the state of the toggle keys (that is, NUM LOCK and CAPS LOCK) using the Synchronize Event PDU. This is necessary when the client loses input focus and then later gets the focus back (possibly with new toggle key states). In a similar vein, the server can also force an update of the local keyboard toggle keys or the local Input Method Editor (IME) being used to ensure that synchronization with the remote session is maintained.

1.3.6 Basic Server Output

In a similar style to input-related PDUs (as specified in section 1.3.5), server output-related PDUs come in two flavors: Slow-Path and Fast-Path. Fast-Path output uses reduced or removed headers to save bandwidth and reduce encoding and decoding latency by reducing the required CPU cycles. Slow-Path output is similar to T.128 output and is not optimized in any way.

The most fundamental output that a server can send to a connected client is bitmap images of the remote session using the Update Bitmap PDU. This allows the client to render the working space and enables a user to interact with the session running on the server. The global palette information for a session is sent to the client in the Update Palette PDU.

The client can choose to render the mouse cursor locally (if it is not included in the graphics' updates sent by the server). In this case the server should send updates of the current cursor image to ensure that it can be drawn with the correct shape (the Pointer Update PDUs should be used to accomplish this task). Furthermore, if the mouse is programmatically moved in the remote session the server should inform the client of the new position using the Pointer Position PDU.

Other basic output which a server sends to a connected client includes the Play Sound PDU, which instructs a client to play rudimentary sounds (by specifying a frequency and its duration) and Connection Management PDUs, as specified in section 2.2.10.

1.3.7 Controlling Server Graphics Output

A client connected to a server and displaying graphics data may need to request that the server resend the graphics data for a collection of rectangular regions of the session screen area, or stop sending graphics data for a period of time (perhaps when the client is minimized). These two tasks are accomplished by having the client send the Refresh Rect PDU and Suppress Output PDUs respectively.

1.4 Relationship to Other Protocols

The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification is based on the ITU (International Telecommunication Union) T.120 series of protocols. The T.120 standard is composed of a suite of communication and application-layer protocols that enable developers to create compatible products and services for real-time, multipoint data connections and conferencing.

1.5 Prerequisites/Preconditions

This protocol assumes that the system already has an IP address and is thus able to communicate on the network. It also assumes that the initiator (or "client") has already obtained the IP address of the server, that the server has registered a port, and that the server is actively listening for client connections on that port.<1>

All multiple-byte fields within a message are assumed to contain data in little-endian byte ordering, unless otherwise specified.

1.6 Applicability Statement

This protocol is applicable in scenarios where interactions with a remote session or remote application are required. In this context, the graphical user interface of a session or application running on a remote machine is transmitted to the client machine. The client, in turn, sends keyboard and mouse input to be processed by the server allowing the client to interact with the remote session or application.

In scenarios in which more specialized communication between client and server components is needed, Virtual Channels (see section 1.3.3) provide an extensible transport mechanism. Examples of more specialized communication include redirection of client-side devices (for example, printers, drives, smart card readers, or Plug and Play devices) and synchronization of client-side and remote session clipboards.

1.7 Versioning and Capability Negotiation

Capability negotiation for RDP is essentially the same as for T.128. The server advertises its capabilities in a Demand Active PDU sent to the client, and the client advertises its capabilities in the follow-up Confirm Active PDU (see the Capability Negotiation phase in section 1.3.1.1). Capability sets are packaged in a combined capability set structure. This structure contains a count of the number of capability sets, followed by the contents of the individual capability sets.

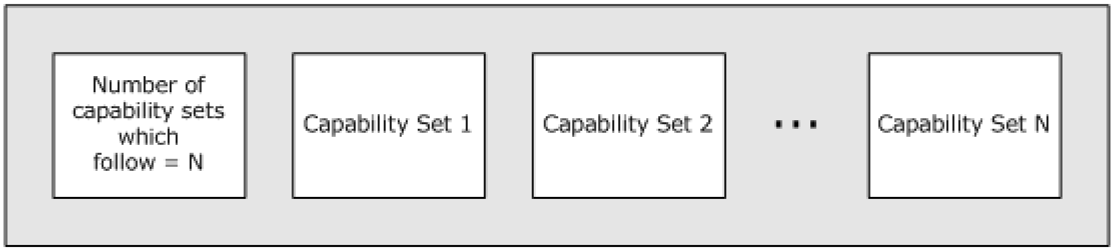


Figure 4: Combined Capability Set Structure

Information exchanged in the capability sets includes data such as supported PDUs and drawing orders, desktop dimensions and allowed color depths, input device support, cache structures and feature support. When the capability sets are received, the client and server should each perform a "merge" operation between their capabilities and the peer capabilities so that all RDP traffic on the wire is consistent with negotiated expectations and can be processed by each party.

Early capability information (in the form of a bitmask) is advertised by the client as part of the data which it sends to the server during the Basic Settings Exchange phase. This information is intended for capabilities that need to be advertised prior to the actual Capability Negotiation phase. For example, support for the Set Error Info PDU needs to be established before the Licensing phase of the connection sequence, which occurs before to the Capability Negotiation phase (see section 1.3.1.1). This is necessary because the server needs to be aware of how errors can be communicated back to the client.

The client and server data exchanged during the Basic Settings Exchange phase in the connection sequence (see section 1.3.1.1) includes an RDP version number (consisting of a major and minor field). However, this version information does not accurately reflect the version of RDP being used (for example, RDP 4.0 clients advertise a minor version field of "1", while all later client versions advertise the same value of "4").

The build number of the client is also available as part of the data the client sends to the server during the Basic Settings Exchange phase. However, this value is implementation dependent and is not necessarily consistent across the spectrum of RDP clients manufactured by different vendors.

1.8 Vendor-Extensible Fields

This protocol contains no vendor-extensible fields.

1.9 Standards Assignments

This protocol makes no standards assignments.

2 Messages

The following sections specify how Remote Desktop Protocol: Basic Connectivity and Graphics Remoting messages are transported and message syntax.

2.1 Transport

The Remote Desktop Protocol: Basic Connectivity and Graphics Remoting packets are encapsulated in TCP. The TCP packets MUST be encapsulated in version 4 of the IP protocol.

There is no officially assigned TCP port for the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification, but protocol servers listen by default on TCP port 3389 for client requests.

2.2 Message Syntax

2.2.1 Normal Connection Sequence

2.2.1.1 Client X.224 Connection Request PDU

The X.224 Connection Request PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Initiation phase (see section 1.3.1.1).

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
tpktHeader																															
x224Crq																															
...																								routingToken (variable)							
...																															
rdpNegData (optional)																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Crq (7 bytes): An X.224 Class 0 Connection Request TPDU, as specified in [X224] section 13.3.

routingToken (variable): Optional and variable-length routing token bytes used for load balancing terminated by a carriage-return (CR) and line-feed (LF) ANSI sequence. For more information, see [MSFT-SDLBTS]. The length of the routing token and CR+LF sequence is included in the **X.224 Connection Request Length Indicator** field.

rdpNegData (8 bytes): An optional RDP Negotiation Request (section 2.2.1.1.1) structure. The length of this negotiation structure is included in the **X.224 Connection Request Length Indicator** field.

2.2.1.1.1 RDP Negotiation Request (RDP_NEG_REQ)

The RDP Negotiation Request structure is used by a client to advertise the security protocols which it supports.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">type</td><td colspan="8">flags</td><td colspan="16">length</td></tr>
<tr><td colspan="32">requestedProtocols</td></tr>
</table>

type (1 byte): An 8-bit unsigned integer. Negotiation packet type. This field MUST be set to 0x01 (TYPE_RDP_NEG_REQ) to indicate that the packet is a Negotiation Request.

flags (1 byte): An 8-bit unsigned integer. Negotiation packet flags. There are currently no defined flags so the field MUST be set to 0x00.

length (2 bytes): A 16-bit unsigned integer. Indicates the packet size. This field MUST be set to 0x0008 (8 bytes).

requestedProtocols (4 bytes): A 32-bit unsigned integer. Flags indicating the supported security protocols.

Flag	Meaning
PROTOCOL_RDP_FLAG 0x00000000	Legacy RDP encryption
PROTOCOL_SSL_FLAG 0x00000001	TLS 1.0 (section 5.4.5.1)
PROTOCOL_HYBRID_FLAG 0x00000002	CredSSP (section 5.4.5.2). If this flag is set, then the PROTOCOL_SSL_FLAG (0x00000001) SHOULD also be set, as TLS (section 5.4.5.1) is a subset of CredSSP.

2.2.1.2 Server X.224 Connection Confirm PDU

The X.224 Connection Confirm PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Connection Initiation phase (see section 1.3.1.1). It is sent as a response to the X.224 Connection Request PDU (section 2.2.1.1).

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
tpktHeader																															
x224Ccf																															
...																								rdpNegData (optional)							
...																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Ccf (7 bytes): An X.224 Class 0 Connection Confirm TPDU, as specified in [X224] section 13.4.

rdpNegData (8 bytes): Optional RDP Negotiation Response (section 2.2.1.2.1) structure or an optional RDP Negotiation Failure (section 2.2.1.2.2) structure. The length of the negotiation structure is included in the **X.224 Connection Confirm Length Indicator** field.

2.2.1.2.1 RDP Negotiation Response (RDP_NEG_RSP)

The RDP Negotiation Response structure is used by a server to inform the client of the security protocol which it has selected to use for the connection.

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
type								flags								length															
selectedProtocol																															

type (1 byte): An 8-bit unsigned integer. Negotiation packet type. This field MUST be set to 0x02 (TYPE_RDP_NEG_RSP) to indicate that the packet is a Negotiation Response.

flags (1 byte): An 8-bit unsigned integer. Negotiation packet flags. There are currently no defined flags so the field MUST be set to 0x00.

length (2 bytes): A 16-bit unsigned integer. Indicates the packet size. This field MUST be set to 0x0008 (8 bytes)

selectedProtocol (4 bytes): A 32-bit unsigned integer. Field indicating the selected security protocol.

Value	Meaning
PROTOCOL_RDP	Legacy RDP encryption

Value	Meaning
0x00000000	
PROTOCOL_SSL 0x00000001	TLS 1.0 (section 5.4.5.1)
PROTOCOL_HYBRID 0x00000002	CredSSP (section 5.4.5.2)

2.2.1.2.2 RDP Negotiation Failure (RDP_NEG_FAILURE)

The RDP Negotiation Failure structure is used by a server to inform the client of a failure that has occurred while preparing security for the connection.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
type								flags								length															
failureCode																															

type (1 byte): An 8-bit unsigned integer. Negotiation packet type. This field MUST be set to 0x03 (TYPE_RDP_NEG_FAILURE) to indicate that the packet is a Negotiation Failure.

flags (1 byte): An 8-bit unsigned integer. Negotiation packet flags. There are currently no defined flags so the field MUST be set to 0x00.

length (2 bytes): A 16-bit unsigned integer. Indicates the packet size. This field MUST be set to 0x0008 (8 bytes).

failureCode (4 bytes): A 32-bit unsigned integer. Field containing the failure code.

Value	Meaning
SSL_REQUIRED_BY_SERVER 0x00000001	The server requires that the client support Enhanced RDP Security (section 5.4) with either TLS 1.0 (section 5.4.5.1) or CredSSP (section 5.4.5.2). If only CredSSP was requested then the server only supports TLS.
SSL_NOT_ALLOWED_BY_SERVER 0x00000002	The server is configured to only use RDP Standard Security (section 5.3) and does not support any External Security Protocols (section 5.4.5).
SSL_CERT_NOT_ON_SERVER 0x00000003	The server does not possess a valid server authentication certificate and cannot initialize the External Security Protocol Provider (see section 5.4.5).
INCONSISTENT_FLAGS 0x00000004	The list of requested security protocols is not consistent with the current security protocol in effect. This error is only possible when the Direct Approach (see sections 5.4.2.2 and 1.3.1.2) is used and an External Security Protocol (section 5.4.5) is already being used.
HYBRID_REQUIRED_BY_SERVER	The server requires that the client support Enhanced RDP

Value	Meaning
0x00000005	Security (section 5.4) with CredSSP (section 5.4.5.2).

2.2.1.3 Client MCS Connect Initial PDU with GCC Conference Create Request

The MCS Connect Initial PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Basic Settings Exchange phase (see Section 1.3.1.1). It is sent after receiving the X.224 Connection Confirm PDU. The MCS Connect Initial PDU encapsulates a GCC Conference Create Request, which encapsulates concatenated blocks of settings data. A basic high-level overview of the nested structure for the Client MCS Connect Initial PDU is illustrated in Figure 2. Note that the order of the settings data blocks is allowed to vary from that shown in Figure 2 and the message syntax layout which follows. This is possible because each data block is identified by a User Data Header structure (see section 2.2.1.3.1).

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
tpktHeader																															
x224Data																								mcsCi (variable)							
...																															
gccCCrq (variable)																															
...																															
clientCoreData																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(clientCoreData cont'd for 46 rows)																															

clientSecurityData
...
...
clientNetworkData (variable)
...
clientClusterData (optional)
...
...

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsCi (variable): Variable-length BER-encoded MCS Connect Initial PDU (using definite-length encoding) as described in [T125] (the ASN.1 structure definition is detailed in [T125] section 7, part 2). The **userData** field of the MCS Connect Initial PDU contains the GCC Conference Create Request data. The maximum allowed size of this user data is 1024 bytes, which implies that the combined size of the **gccCCrq**, **clientCoreData**, **clientSecurity**, **clientNetworkData** and **clientClusterData** fields must be less than 1024 bytes.

gccCCrq (variable): Variable-length PER-encoded GCC Conference Create Request as described in [T124] (the ASN.1 structure definition is detailed in [T124] section 8.7) appended as user data to the MCS Connect Initial PDU (using the format described in [T124] sections 9.5 and 9.6). The **userData** field of the GCC Conference Create Request contains concatenated client data blocks.

clientCoreData (216 bytes): Client Core Data (section 2.2.1.3.2) structure.

clientSecurityData (12 bytes): Client Security Data (section 2.2.1.3.3) structure.

clientNetworkData (variable): Optional and variable length Client Network Data (section 2.2.1.3.4) structure.

clientClusterData (12 bytes): Optional Client Cluster Data (section 2.2.1.3.5) structure.

2.2.1.3.1 User Data Header (TS_UD_HEADER)

The TS_UD_HEADER precedes all data blocks in the client and server GCC user data.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">type</td><td colspan="16">length</td></tr>
</table>

type (2 bytes): A 16-bit unsigned integer. The type of the data block that this header precedes.

Value	Meaning
CS_CORE 0xC001	The data block that follows contains Client Core Data (section 2.2.1.3.2).
CS_SECURITY 0xC002	The data block that follows contains Client Security Data (section 2.2.1.3.3).
CS_NET 0xC003	The data block that follows contains Client Network Data (section 2.2.1.3.4).
CS_CLUSTER 0xC004	The data block that follows contains Client Cluster Data (section 2.2.1.3.5).
SC_CORE 0x0C01	The data block that follows contains Server Core Data (section 2.2.1.4.2).
SC_SECURITY 0x0C02	The data block that follows contains Server Security Data (section 2.2.1.4.3).
SC_NET 0x0C03	The data block that follows contains Server Network Data (section 2.2.1.4.4).

length (2 bytes): A 16-bit unsigned integer. The size in bytes of the data block, including this header.

2.2.1.3.2 Client Core Data (TS_UD_CS_CORE)

The TS_UD_CS_CORE data block contains core client connection-related information.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">header</td></tr>
<tr><td colspan="32">version</td></tr>
<tr><td colspan="16">desktopWidth</td><td colspan="16">desktopHeight</td></tr>
<tr><td colspan="16">colorDepth</td><td colspan="16">SASSequence</td></tr>
<tr><td colspan="32">keyboardLayout</td></tr>
</table>

clientBuild
clientName
...
...
...
...
...
...
...
keyboardType
keyboardSubType
keyboardFunctionKey
imeFileName
...
...
...
...
...
...
...
(imeFileName cont'd for 8 rows)

<table>
<tr><td>postBeta2ColorDepth</td><td>clientProductId</td></tr>
<tr><td colspan="2">serialNumber (optional)</td></tr>
<tr><td>highColorDepth (optional)</td><td>supportedColorDepths (optional)</td></tr>
<tr><td>earlyCapabilityFlags (optional)</td><td>clientDigProductId (optional)</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">(clientDigProductId (optional) cont'd for 8 rows)</td></tr>
<tr><td>...</td><td>pad2octets (optional)</td></tr>
<tr><td colspan="2">serverSelectedProtocol (optional)</td></tr>
</table>

header (4 bytes): GCC user data block header, as specified in section 2.2.1.3.1. The User Data Header **type** field MUST be set to CS_CORE (0xC001).

version (4 bytes): A 32-bit unsigned integer. Client version number for the Remote Desktop Protocol (RDP). The major version number is stored in the high 2 bytes, while the minor version number is stored in the low 2 bytes.

Value	Meaning
0x00080001	RDP 4.0 clients
0x00080004	RDP 5.0, 5.1, 5.2 and 6.0 clients

desktopWidth (2 bytes): A 16-bit unsigned integer. The requested desktop width in pixels (up to a maximum value of 4096 pixels).

desktopHeight (2 bytes): A 16-bit unsigned integer. The requested desktop height in pixels (up to a maximum value of 2048 pixels).

colorDepth (2 bytes): A 16-bit unsigned integer. The requested color depth. This field MUST be set to RNS_UD_COLOR_8BPP (0xCA01) for historical reasons.

SASSequence (2 bytes): A 16-bit unsigned integer. Secure access sequence. This field SHOULD be set to RNS_UD_SAS_DEL (0xAA03).

keyboardLayout (4 bytes): A 32-bit unsigned integer. Keyboard layout (active input locale identifier). For a list of possible input locales, see [MSDN-MUI].

clientBuild (4 bytes): A 32-bit unsigned integer. Build number of the client.

clientName (32 bytes): Name of the client computer. This field contains up to 15 Unicode characters plus a null terminator.

keyboardType (4 bytes): A 32-bit unsigned integer. Keyboard type.

Value	Meaning
1	IBM PC/XT or compatible (83-key) keyboard
2	Olivetti "ICO" (102-key) keyboard
3	IBM PC/AT (84-key) and similar keyboards
4	IBM enhanced (101- or 102-key) keyboard
5	Nokia 1050 and similar keyboards
6	Nokia 9140 and similar keyboards
7	Japanese keyboard

keyboardSubType (4 bytes): A 32-bit unsigned integer. The keyboard subtype (an original equipment manufacturer-dependent value).

keyboardFunctionKey (4 bytes): A 32-bit unsigned integer. The number of function keys on the keyboard.

imeFileName (64 bytes): A 64-byte field. The Input Method Editor (IME) file name associated with the input locale. This field contains up to 31 Unicode characters plus a null terminator.

postBeta2ColorDepth (2 bytes): A 16-bit unsigned integer. The requested color depth examined by RDP 4.0 and 5.0 version servers.

Value	Meaning
RNS_UD_COLOR_4BPP 0xCA00	4 bits-per-pixel
RNS_UD_COLOR_8BPP 0xCA01	8 bits-per-pixel
RNS_UD_COLOR_16BPP_555 0xCA02	15-bit 555 red, green, blue (RGB) mask (5 bits for red, 5 bits for green, and 5 bits for blue)
RNS_UD_COLOR_16BPP_565 0xCA03	16-bit 565 RGB mask (5 bits for red, 6 bits for green, and 5 bits for blue)

Value	Meaning
RNS_UD_COLOR_24BPP 0xCA04	24-bit RGB mask (8 bits for red, 8 bits for green, and 8 bits for blue)

If the **highColorDepth** field is being used and the required color depth is greater than or equal to 8 bits-per-pixel, then this field MUST be set to RNS_UD_COLOR_8BPP (0xCA01).

clientProductId (2 bytes): A 16-bit unsigned integer. The client product ID. This field SHOULD be initialized to 1.

serialNumber (4 bytes): A 32-bit unsigned integer. Serial number. This field SHOULD be initialized to 0. If this field is not present, then none of the subsequent fields MUST be present.

highColorDepth (2 bytes): A 16-bit unsigned integer. The requested color depth examined by RDP 5.1, 5.2, and 6.0 servers.

Value	Meaning
4	4 bits-per-pixel
8	8 bits-per-pixel
15	15-bit 555 red, green, blue (RGB) mask (5 bits for red, 5 bits for green, and 5 bits for blue)
16	16-bit 565 RGB mask (5 bits for red, 6 bits for green, and 5 bits for blue)
24	24-bit RGB mask (8 bits for red, 8 bits for green, and 8 bits for blue)

If this field is present, then all of the preceding fields MUST also be present. If this field is not present, then none of the subsequent fields MUST be present.

supportedColorDepths (2 bytes): A 16-bit unsigned integer. Specifies the high color depths that the client is capable of supporting (examined by RDP 5.1 and later servers).

Flag	Meaning
RNS_UD_24BPP_SUPPORT 0x0001	24-bit RGB mask (8 bits for red, 8 bits for green, and 8 bits for blue)
RNS_UD_16BPP_SUPPORT 0x0002	16-bit 565 RGB mask (5 bits for red, 6 bits for green, and 5 bits for blue)
RNS_UD_15BPP_SUPPORT 0x0004	15-bit 555 red, green, blue (RGB) mask (5 bits for red, 5 bits for green, and 5 bits for blue)
RNS_UD_32BPP_SUPPORT 0x0008	32-bit RGB mask (8 bits for the alpha channel, 8 bits for red, 8 bits for green, and 8 bits for blue)

If this field is present, then all of the preceding fields MUST also be present. If this field is not present, then none of the subsequent fields MUST be present.

earlyCapabilityFlags (2 bytes): A 16-bit unsigned integer. It specifies capabilities early in the connection sequence.

Flag	Meaning
RNS_UD_CS_SUPPORT_ERRINFO_PDU 0x0001	Client can support the Set Error Info PDU (section 2.2.5.1) from the server.
RNS_UD_CS_WANT_32BPP_SESSION 0x0002	Indicates that the client is requesting a session color depth of 32 bits-per-pixel. This flag is necessary as the **highColorDepth** field does not support a value of 32. If this flag is set, the **highColorDepth** field SHOULD be set to 24 to provide an acceptable fallback for the scenario where the server does not support 32 bpp color.
RNS_UD_CS_STRONG_ASYMMETRIC_KEYS 0x0008	Indicates that the client supports asymmetric keys larger than 512-bits for use with the Server Certificate (see Section 2.2.1.4.3.1) sent in the Server Security Data block (see section 2.2.1.4.3).

If this field is present, then all of the preceding fields MUST also be present. If this field is not present, then none of the subsequent fields MUST be present.

clientDigProductId (64 bytes): Contains a value which uniquely identifies the client. If this field is present, then all of the preceding fields MUST also be present. If this field is not present, then none of the subsequent fields MUST be present.

pad2octets (2 bytes): A 16-bit unsigned integer. Padding to align the **serverSelectedProtocol** field on the correct byte boundary. If this field is present, then all of the preceding fields MUST also be present. If this field is not present, then none of the subsequent fields MUST be present.

serverSelectedProtocol (4 bytes): A 32-bit unsigned integer. It contains the value returned by the server in the **selectedProtocol** field of the RDP Negotiation Response (section 2.2.1.2.1) structure. In the event that an RDP Negotiation Response structure was not sent, this field MUST be initialized to PROTOCOL_RDP (0). If this field is present, then all of the preceding fields MUST also be present.

2.2.1.3.3 Client Security Data (TS_UD_CS_SEC)

The TS_UD_CS_SEC data block contains security-related information used to advertise client cryptographic support. This information is only relevant when Standard RDP Security (section 5.3) is in effect, as opposed to Enhanced RDP Security (section 5.4)). See section 3 (in particular, section 5.3.2) for a detailed discussion of how this information is used.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">header</td></tr>
<tr><td colspan="32">encryptionMethods</td></tr>
<tr><td colspan="32">extEncryptionMethods</td></tr>
</table>

header (4 bytes): GCC user data block header as described in User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to CS_SECURITY (0xC002).

encryptionMethods (4 bytes): A 32-bit unsigned integer. Cryptographic methods supported by the client and used in conjunction with Standard RDP Security (see section 5.3.2).

Value	Meaning
40BIT_ENCRYPTION_FLAG 0x00000001	40-bit session keys should be used to encrypt data (with RC4) and generate message authentication codes (MAC).
128BIT_ENCRYPTION_FLAG 0x00000002	128-bit session keys should be used to encrypt data (with RC4) and generate MACs.
56BIT_ENCRYPTION_FLAG 0x00000008	56-bit session keys should be used to encrypt data (with RC4) and generate MACs.
FIPS_ENCRYPTION_FLAG 0x00000010	All encryption and message authentication code generation routines should be FIPS 140-1 compliant.

extEncryptionMethods (4 bytes): A 32-bit unsigned integer. This field is used exclusively for the French locale. In French locale clients, **encryptionMethods** MUST be set to 0 and **extEncryptionMethods** MUST be set to the value to which **encryptionMethods** would have been set. For non-French locale clients, this field MUST be set to 0.

2.2.1.3.4 Client Network Data (TS_UD_CS_NET)

The TS_UD_CS_NET packet contains a list of requested virtual channels.

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

header (4 bytes): A 32-bit unsigned integer. GCC user data block header, as specified in User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to CS_NET (0xC003).

channelCount (4 bytes): A 32-bit unsigned integer. The number of requested static virtual channels (the maximum allowed is 30).

channelDefArray (variable): A variable length array containing the information for requested static virtual channels encapsulated in CHANNEL_DEF (section 2.2.1.3.4.1) structures. The number of CHANNEL_DEF structures which follows is given by the **channelCount** field.

2.2.1.3.4.1 Channel Definition Structure (CHANNEL_DEF)

The CHANNEL_DEF packet contains information for a particular static virtual channel.

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
name																															
...																															
options																															

name (8 bytes): An 8-byte array containing a unique 7-character ANSI channel name and a null terminator.

options (4 bytes): A 32-bit unsigned integer. Channel option flags.

Flag	Meaning
INITIALIZED 0x80000000	Absence of this flag indicates that this channel is a placeholder and that the server should not actually set it up.
ENCRYPT_RDP 0x40000000	Channel data should be encrypted in the same way as RDP input and output data.
ENCRYPT_SC	Server-to-client data should be encrypted. This flag is ignored if

Flag	Meaning
0x20000000	ENCRYPT_RDP is set.
ENCRYPT_CS 0x10000000	Client-to-server data should be encrypted. This flag is ignored if ENCRYPT_RDP is set.
PRI_HIGH 0x08000000	Channel data should be sent with high MCS priority.
PRI_MED 0x04000000	Channel data should be sent with medium MCS priority.
PRI_LOW 0x02000000	Channel data should be sent with low MCS priority.
COMPRESS_RDP 0x00800000	Virtual channel data should be compressed if RDP data is being compressed.
COMPRESS 0x00400000	Virtual channel data should be compressed, regardless of RDP compression.
SHOW_PROTOCOL 0x00200000	Server virtual channel add-ins should be shown the full virtual channel packet header on receipt of data. If this option is not set, the server add-ins receive only the data stream without headers.
REMOTE_CONTROL_PERSISTENT 0x00100000	Channel is persistent across remote control transactions.

2.2.1.3.5 Client Cluster Data (TS_UD_CS_CLUSTER)

The TS_UD_CS_CLUSTER data block is sent by the client to the server either to advertise that it can support the Server Redirection PDU (see [MS_RDPEGDI] section 2.2.3.1) or to request a connection to a given session identifier.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
Flags																															
RedirectedSessionID																															

header (4 bytes): GCC user data block header, as specified in User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to CS_CLUSTER (0xC004).

Flags (4 bytes): A 32-bit unsigned integer. Cluster information flags.

Flag	Meaning
REDIRECTION_SUPPORTED 0x00000001	The client can receive server session redirection packets. If this flag is set, the

Flag	Meaning
	ServerSessionRedirectionVersionMask will contain the server session redirection version that the client supports.
ServerSessionRedirectionVersionMask 0x0000003C	The server session redirection version that the client supports. See the discussion which follows this table for more information.
REDIRECTED_SESSIONID_FIELD_VALID 0x00000002	The **RedirectedSessionID** field contains a valid session identifier to which the client wants to connect.
REDIRECTED_SMARTCARD 0x00000040	The client logged on with a smart card.

The ServerSessionRedirectionVersionMask is a 4-bit enumerated value containing the server session redirection version supported by the client. Only one version can be supported; the version values cannot be combined. Possible version values are:

Value	Meaning
REDIRECTION_VERSION3 0x08	If REDIRECTION_SUPPORTED is set, server session redirection version 3 is supported by the client.
REDIRECTION_VERSION4 0x0C	If REDIRECTION_SUPPORTED is set, server session redirection version 4 is supported by the client.

RedirectedSessionID (4 bytes): A 32-bit unsigned integer. If the REDIRECTED_SESSIONID_FIELD_VALID flag is set in the **Flags** field, then the **RedirectedSessionID** field contains a valid session identifier to which the client wants to connect.

2.2.1.4 Server MCS Connect Response PDU with GCC Conference Create Response

The MCS Connect Response PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Basic Settings Exchange phase (see section 1.3.1.1). It is sent as a response to the MCS Connect Initial PDU. The MCS Connect Response PDU encapsulates a GCC Conference Create Response, which encapsulates concatenated blocks of settings data. A basic high-level overview of the nested structure for the Server MCS Connect Response PDU is illustrated in Figure 2. Note that the order of the settings data blocks is allowed to vary from that shown in Figure 2 and the message syntax layout that follows. This is possible because each data block is identified by a User Data Header structure (see section 2.2.1.4.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsCrsp (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">gccCCrsp (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">serverCoreData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">serverSecurityData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">serverNetworkData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsCrsp (variable): Variable-length BER-encoded MCS Connect Response PDU (using definite-length encoding) as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 2). The **userData** field of the MCS Connect Response PDU contains the GCC Conference Create Response data.

gccCCrsp (variable): Variable-length PER-encoded GCC Conference Create Request PDU as described in [T124] (the ASN.1 structure definition is specified in [T124] section 8.7) appended as user data to the MCS Connect Initial PDU (using the format specified in [T124] sections 9.5 and 9.6). The **userData** field of the GCC Conference Create Response contains concatenated server data blocks.

serverCoreData (12 bytes): Server Core Data (section 2.2.1.4.2) structure.

serverSecurityData (variable): Variable-length Server Security Data (section 2.2.1.4.3) structure.

serverNetworkData (variable): Variable-length Server Network Data (section 2.2.1.4.4) structure.

2.2.1.4.1 User Data Header (TS_UD_HEADER)

See section 2.2.1.3.1 for a description of the User Data Header.

2.2.1.4.2 Server Core Data (TS_UD_SC_CORE)

The TS_UD_SC_CORE data block contains core server connection-related information.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
version																															
clientRequestedProtocols (optional)																															

header (4 bytes): GCC user data block header, as specified in User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to SC_CORE (0x0C01).

version (4 bytes): A 32-bit unsigned integer. The server version number for the Remote Desktop Protocol (RDP). The major version number is stored in the high 2 bytes, while the minor version number is stored in the low 2 bytes.

Value	Meaning
0x00080001	RDP 4.0 clients
0x00080004	RDP 5.0, 5.1, 5.2 and 6.0 clients

clientRequestedProtocols (4 bytes): A 32-bit unsigned integer. Contains the flags sent by the client in the **requestedProtocols** field of the RDP Negotiation Request (section 2.2.1.1.1) structure. In the event that an RDP Negotiation Request structure was not sent, this field MUST be initialized to PROTOCOL_RDP (0).

2.2.1.4.3 Server Security Data (TS_UD_SC_SEC1)

The TS_UD_SC_SEC1 data block returns negotiated security-related information to the client. See section 3 (in particular section 5.3.2) for a detailed discussion of how this information is used.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">header</td></tr>
<tr><td colspan="32">encryptionMethod</td></tr>
<tr><td colspan="32">encryptionLevel</td></tr>
<tr><td colspan="32">serverRandomLen</td></tr>
<tr><td colspan="32">serverCertLen</td></tr>
<tr><td colspan="32">serverRandom (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">serverCertificate (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

header (4 bytes): GCC user data block header, as specified in User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to SC_SECURITY (0x0C02).

encryptionMethod (4 bytes): A 32-bit unsigned integer. The selected cryptographic method to use for the session. When Enhanced RDP Security (section 5.4) is being used, this field MUST be set to ENCRYPTION_METHOD_NONE (0).

Value	Meaning
ENCRYPTION_METHOD_NONE 0x00000000	No encryption and message authentication codes will be used.
ENCRYPTION_METHOD_40BIT 0x00000001	40-bit session keys will be used to encrypt data (with RC4) and generate message authentication codes (MAC).
ENCRYPTION_METHOD_128BIT 0x00000002	128-bit session keys will be used to encrypt data (with RC4) and generate MACs.
ENCRYPTION_METHOD_56BIT 0x00000008	56-bit session keys will be used to encrypt data (with RC4) and generate MACs.
ENCRYPTION_METHOD_FIPS 0x00000010	All encryption and message authentication code generation routines will be FIPS 140-1 compliant.

encryptionLevel (4 bytes): A 32-bit unsigned integer. It describes the encryption behavior to use for the session. When Enhanced RDP Security (section 5.4) is being used, this field MUST be set to ENCRYPTION_LEVEL_NONE (0).

Name	Value
ENCRYPTION_LEVEL_NONE	0x00000000
ENCRYPTION_LEVEL_LOW	0x00000001
ENCRYPTION_LEVEL_CLIENT_COMPATIBLE	0x00000002
ENCRYPTION_LEVEL_HIGH	0x00000003
ENCRYPTION_LEVEL_FIPS	0x00000004

See section 5.3.1 for a description of each of the low, client-compatible, high and FIPS encryption levels.

serverRandomLen (4 bytes): A 32-bit unsigned integer. The size in bytes of the **serverRandom** field. This field MUST be set to 32 bytes.

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

serverRandom (variable): The variable-length server random value used to derive session keys (see sections 5.3.4 and 5.3.5). The length in bytes is given by the **serverRandomLen** field.

serverCertificate (variable): The variable-length certificate containing the server's public key information. The length in bytes is given by the **serverCertLen** field.

2.2.1.4.3.1 Server Certificate (SERVER_CERTIFICATE)

The SERVER_CERTIFICATE structure describes the generic server certificate structure to which all server certificates present in the Server Security Data (section 2.2.1.4.3) conform.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">dwVersion</td></tr>
<tr><td colspan="32">certData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

dwVersion (4 bytes): 32-bit unsigned integer. The certificate version.

Value	Meaning
CERT_CHAIN_VERSION_1 0x00000001	The certificate contained in the **certData** field is a Server Proprietary Certificate (see section 2.2.1.4.3.1.1).
CERT_CHAIN_VERSION_2 0x00000002	The certificate contained in the **certData** field is an X.509 Certificate (see section 5.3.3.2).

certData (variable): Certificate data. The format of this certificate data is determined by the **dwVersion** field.

2.2.1.4.3.1.1 Server Proprietary Certificate (PROPRIETARYSERVERCERTIFICATE)

The PROPRIETARYSERVERCERTIFICATE structure describes a signed certificate containing the server's public key and conforming to the structure of a Server Certificate (section 2.2.1.4.3.1). For a detailed description of Proprietary Certificates, see section 5.3.3.1.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">dwVersion</td></tr>
<tr><td colspan="32">dwSigAlgId</td></tr>
<tr><td colspan="32">dwKeyAlgId</td></tr>
<tr><td colspan="16">wPublicKeyBlobType</td><td colspan="16">wPublicKeyBlobLen</td></tr>
<tr><td colspan="32">PublicKeyBlob (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">wSignatureBlobType</td><td colspan="16">wSignatureBlobLen</td></tr>
<tr><td colspan="32">SignatureBlob (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

dwVersion (4 bytes): A 32-bit unsigned integer. The certificate version number. This field MUST be set to CERT_CHAIN_VERSION_1 (0x00000001).

dwSigAlgId (4 bytes): A 32-bit unsigned integer. The signature algorithm identifier. This field MUST be set to SIGNATURE_ALG_RSA (0x00000001).

dwKeyAlgId (4 bytes): A 32-bit unsigned integer. The key algorithm identifier. This field MUST be set to KEY_EXCHANGE_ALG_RSA (0x00000001).

wPublicKeyBlobType (2 bytes): A 16-bit unsigned integer. The type of data in the **PublicKeyBlob** field. This field MUST be set to BB_RSA_KEY_BLOB (0x0006).

wPublicKeyBlobLen (2 bytes): A 16-bit unsigned integer. The size in bytes of the **PublicKeyBlob** field.

PublicKeyBlob (variable): Variable-length server public key bytes, formatted using the RSA Public Key (section 2.2.1.4.3.1.1.1) structure. The length in bytes is given by the **wPublicKeyBlobLen** field.

wSignatureBlobType (2 bytes): A 16-bit unsigned integer. The type of data in the **SignatureKeyBlob** field. This field is set to BB_RSA_SIGNATURE_BLOB (0x0008)

wSignatureBlobLen (2 bytes): A 16-bit unsigned integer. The size in bytes of the **SignatureKeyBlob** field.

SignatureBlob (variable): Variable-length signature of the certificate created with the Terminal Services Signing Key (see sections 5.3.3.1.1 and 5.3.3.1.2). The length in bytes is given by the **wSignatureBlobLen** field.

2.2.1.4.3.1.1.1 RSA Public Key (RSA_PUBLIC_KEY)

The structure used to describe a public key in a proprietary certificate.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">magic</td></tr>
<tr><td colspan="32">keylen</td></tr>
<tr><td colspan="32">bitlen</td></tr>
<tr><td colspan="32">datalen</td></tr>
<tr><td colspan="32">pubExp</td></tr>
<tr><td colspan="32">modulus (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

magic (4 bytes): A 32-bit unsigned integer. The sentinel value. This field MUST be set to 0x31415352 ("RSA1" in ANSI when the bytes are arranged in little-endian order).

keylen (4 bytes): A 32-bit unsigned integer. The size in bytes of the public key modulus.

bitlen (4 bytes): A 32-bit unsigned integer. The number of bits in the public key modulus.

datalen (4 bytes): A 32-bit unsigned integer. The maximum number of bytes that can be encoded using the public key.

pubExp (4 bytes): A 32-bit unsigned integer. The public exponent of the public key.

modulus (variable): Variable-length modulus of the public key. The length in bytes is given by the **keylen** field.

2.2.1.4.4 Server Network Data (TS_UD_SC_NET)

The TS_UD_SC_NET data block is a reply to the static virtual channel list presented in the Client Network Data (section 2.2.1.3.4) structure.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">header</td></tr>
<tr><td colspan="16">MCSChannelId</td><td colspan="16">channelCount</td></tr>
<tr><td colspan="32">channelIdArray (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">Pad (optional)</td></tr>
</table>

header (4 bytes): A GCC user data block header, as specified in section User Data Header (section 2.2.1.3.1). The User Data Header **type** field MUST be set to SC_NET (0x0C03).

MCSChannelId (2 bytes): 16-bit unsigned integer. The MCS channel identifier which the client should join to receive display data and send client input (I/O channel).

channelCount (2 bytes): 16-bit unsigned integer. The number of 16-bit unsigned integer MCS channel IDs in the **channelIdArray** field.

channelIdArray (variable): Variable-length array of unsigned short MCS channel IDs which have been allocated (the number is given by the **channelCount** field). Each MCS channel ID corresponds in position to the channels requested in the Client Network Data structure. A channel value of 0 indicates that the channel was not allocated.

Pad (2 bytes): 16-bit unsigned integer. Optional padding. Values in this field are ignored. The size in bytes of the Server Network Data structure MUST be a multiple of 4. If the **channelCount** field contains an odd value, then the size of the **channelIdArray** (and by implication the entire Server Network Data structure) will not be a multiple of 4. In this scenario, the **Pad** field MUST be present and it is used to add an additional 2 bytes to the size of the Server Network Data structure. If the **channelCount** field contains an even value, then the **Pad** field is not required and MUST not be present.

2.2.1.5 Client MCS Erect Domain Request PDU

The MCS Erect Domain Request PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Channel Connection phase (see section 1.3.1.1). It is sent after receiving the MCS Connect Response PDU (section 2.2.1.4).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsEDrq</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsEDrq (5 bytes): PER-encoded MCS Erect Domain Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 3).

2.2.1.6 Client MCS Attach User Request PDU

The MCS Attach User Request PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Channel Connection phase (see section 1.3.1.1) to request a user channel ID. It is sent after transmitting the MCS Erect Domain Request PDU (section 2.2.1.5).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsAUrq</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsAUrq (1 byte): PER-encoded MCS Attach User Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 5).

2.2.1.7 Server MCS Attach User Confirm PDU

The MCS Attach User Confirm PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Channel Connection phase (see section 1.3.1.1). It is sent as a response to the MCS Attach User Request PDU (section 2.2.1.6).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsAUcf</td></tr>
<tr><td colspan="24">...</td><td colspan="8"></td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in section [X224] 13.7.

mcsAUcf (4 bytes): PER-encoded MCS Attach User Confirm PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 5).

2.2.1.8 Client MCS Channel Join Request PDU

The MCS Channel Join Request PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Channel Connection phase (see section 1.3.1.1). It is sent after receiving the MCS Attach User Confirm PDU (section 2.2.1.7). The client uses the MCS Channel Join Request PDU to join the user channel obtained from the Attach User Confirm PDU, the I/O channel and all of the static virtual channels obtained from the Server Network Data (section 2.2.1.4.4) structure.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsCJrq</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsCJrq (5 bytes): PER-encoded MCS Channel Join Request PDU as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 6).

2.2.1.9 Client MCS Channel Join Confirm PDU

The MCS Channel Join Confirm PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Channel Connection phase (see section 1.3.1.1). It is sent as a response to the MCS Channel Join Request PDU (section 2.2.1.8).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsCJcf</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="24">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsCJcf (8 bytes): PER-encoded MCS Channel Join Confirm PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 6).

2.2.1.10 Client Security Exchange PDU

The Security Exchange PDU is a Standard RDP Connection Sequence PDU sent from client to server during the RDP Security Commencement phase (see section 1.3.1.1). It MAY be sent after receiving all requested MCS Channel Join Confirm PDUs (section 2.2.1.9).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityExchangePduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable-length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains the Security Exchange PDU data.

securityExchangePduData (variable): The actual contents of the Security Exchange PDU, as specified in section 2.2.1.10.1.

2.2.1.10.1 Security Exchange PDU Data (TS_SECURITY_PACKET)

The TS_SECURITY_PACKET structure contains the encrypted client random value which is used together with the server random (see section 2.2.1.4.3) to derive session keys to secure the connection (see sections 5.3.4 and 5.3.5).

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
basicSecurityHeader																															
length																															
encryptedClientRandom (variable)																															
...																															

basicSecurityHeader (4 bytes): TS_SECURITY_HEADER (4 bytes). The basic security header, as specified in section 2.2.8.1.1.2.1. The **flags** field of the security header MUST contain the SEC_EXCHANGE_PKT flag (0x0001).

length (4 bytes): 32-bit unsigned integer. The size in bytes of the buffer containing the encrypted client random value, not including the header length.

encryptedClientRandom (variable): The client random value encrypted with the public key of the server (see section 5.3.4).

2.2.1.11 Client Info PDU

The Client Info PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Secure Settings Exchange phase (see section 1.3.1.1). It is sent after transmitting a Security Exchange PDU (section 2.2.1.10) or, if the Security Exchange PDU was not sent, it is transmitted after receiving all requested MCS Channel Join Confirm PDUs (section 2.2.1.9).

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
tpktHeader																															
x224Data																								mcsSDrq (variable)							
...																															
clientInfoPduData (variable)																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains the Client Info PDU data.

clientInfoPduData (variable): The actual contents of the Client Info PDU, as specified in section 2.2.1.11.1.

2.2.1.11.1 Client Info PDU Data (CLIENT_INFO_PDU)

The CLIENT_INFO_PDU structure serves as a wrapper for a Security Header (section 2.2.8.1.1.2) and the actual client information contained in a TS_INFO_PACKET (section 2.2.1.11.1.1) structure.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
securityHeader (variable)																															
...																															
infoPacket (variable)																															
...																															

securityHeader (variable): The security header. This field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_NONE (0).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

The **flags** field of the security header MUST contain the SEC_INFO_PKT flag (see section 2.2.8.1.1.2.1).

infoPacket (variable): Client information, as specified in TS_INFO_PACKET.

2.2.1.11.1.1 Info Packet (TS_INFO_PACKET)

The TS_INFO_PACKET structure contains extra information not passed to the server during the Basic Settings Exchange phase (see section 1.3.1.1) of the Standard RDP Connection Sequence, primarily to ensure that it gets encrypted (as auto-logon password data and other sensitive information is passed here).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">CodePage</td></tr>
<tr><td colspan="32">flags</td></tr>
<tr><td colspan="16">cbDomain</td><td colspan="16">cbUserName</td></tr>
<tr><td colspan="16">cbPassword</td><td colspan="16">cbAlternateShell</td></tr>
<tr><td colspan="16">cbWorkingDir</td><td colspan="16">Domain (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">UserName (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Password (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">AlternateShell (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">WorkingDir (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">extraInfo (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

CodePage (4 bytes): A 32-bit unsigned integer. This field contains the ANSI codepage descriptor being used by the client (for a list of code pages, see [MSDN-CP]). However, if the Info Packet is in Unicode (the INFO_UNICODE flag is set), then the **CodePage** field is overridden to contain the active input locale identifier in the low word (for a list of possible input locales, see [MSDN-MUI]).

flags (4 bytes): A 32-bit unsigned integer. Option flags.

Flag	Meaning
INFO_MOUSE 0x00000001	Indicates that the client machine has a mouse attached.
INFO_DISABLECTRLALTDEL 0x00000002	Indicates that the CTRL+ALT+DEL (or the equivalent) secure access keyboard sequence is not required at the logon prompt.
INFO_AUTOLOGON 0x00000008	The client requests auto logon using the included user name, password and domain.
INFO_UNICODE 0x00000010	Indicates that the character set for strings in the Info Packet is Unicode. If this flag is absent, the character set is ANSI. The presence of this flag affects the meaning of the **CodePage** field in the Info Packet structure.
INFO_MAXIMIZESHELL 0x00000020	Specifies whether the alternate shell (specified in the **AlternateShell** field of the Info Packet structure) should be started in a maximized state.
INFO_LOGONNOTIFY 0x00000040	Indicates that the client wants to be informed of the user name and domain used to log on to the server, as well as the ID of the session to which the user connected. The Save Session Info PDU (section 2.2.10.1) is sent from the server to notify the client of this information using a Logon Info Version 1 (section 2.2.10.1.1.1) or Logon Info Version 2 (section 2.2.10.1.1.2) structure.
INFO_COMPRESSION 0x00000080	Indicates that the CompressionTypeMask is valid and contains the highest compression package type supported by the client.
CompressionTypeMask 0x00001E00	Indicates the highest compression package type supported. See the discussion which follows this table for more information.
INFO_ENABLEWINDOWSKEY 0x00000100	Indicates that the client uses the Windows key on Windows-compatible keyboards.
INFO_REMOTECONSOLEAUDIO 0x00002000	Requests that any audio played in a remote session be played on the remote computer (see [MS-RDPEA]).
INFO_FORCE_ENCRYPTED_CS_PDU 0x00004000	Indicates that the client will only send encrypted packets to the server if encryption is in force. Setting this flag prevents the server from processing unencrypted packets in man-in-the-middle attack scenarios. This flag is only understood by RDP 5.2 and later servers.
INFO_RAIL 0x00008000	Indicates that the remote connection being established is for the purpose of launching remote programs (see Section [MS-RDPERP]). This flag is only understood by RDP 6.0 and later servers.
INFO_LOGONERRORS 0x00010000	Indicates a request for logon error notifications using the Save Session Info PDU (section 2.2.10.1). This flag is only understood by RDP 6.0 and later servers.
INFO_MOUSE_HAS_WHEEL 0x00020000	Indicates that the mouse which is connected to the client machine has a scroll wheel. This flag is only understood by

Flag	Meaning
	RDP 6.0 and later servers.
INFO_PASSWORD_IS_SC_PIN 0x00040000	Indicates that the **Password** field in the Info Packet contains a smart card personal identification number (PIN). This flag is only understood by RDP 6.0 and later servers.
INFO_NOAUDIOPLAYBACK 0x00080000	Indicates that no audio redirection or playback (see [MS-RDPEA]) should take place. This flag is only understood by RDP 6.0 and later servers.
INFO_USING_SAVED_CREDS 0x00100000	Any user credentials sent on the wire during the RDP Connection Sequence (see Sections 1.3.1.1 and 1.3.1.2) have been retrieved from a credential store and were not obtained directly from the user.

The CompressionTypeMask is a four-bit enumerated value containing the highest compression package support available on the client. The packages codes are:

Value	Meaning
PACKET_COMPR_TYPE_8K 0	MPPC 8K compression (see MPPC-8K (section 3.1.8.4.1))
PACKET_COMPR_TYPE_64K 1	MPPC 64K compression (see MPPC-64K (section 3.1.8.4.2))
PACKET_COMPR_TYPE_RDP6 2	RDP 6.0 bulk compression (see [MS-RDPEGDI] section 3.1.8).

Support for package *n* implies support for all lesser compression packages *0...(n - 1)*.

cbDomain (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **Domain** field. This size excludes the length of the mandatory null terminator.

cbUserName (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **UserName** field. This size excludes the length of the mandatory null terminator.

cbPassword (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **Password** field. This size excludes the length of the mandatory null terminator.

cbAlternateShell (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **AlternateShell** field. This size excludes the length of the mandatory null terminator.

cbWorkingDir (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **WorkingDir** field. This size excludes the length of the mandatory null terminator.

Domain (variable): Variable length logon domain of the user (the length in bytes is given by the **cbDomain** field). The maximum length allowed by RDP 4.0 and RDP 5.0 servers is 52 bytes (including the mandatory null terminator). RDP 5.1 and later allow a maximum length of 512 bytes (including the mandatory null terminator). The field must contain at least a null terminator character in ANSI or Unicode format (depending on the presence of the INFO_UNICODE flag).

UserName (variable): Variable length logon user name of the user (the length in bytes is given by the **cbUserName** field). The maximum length allowed by RDP 4.0 and RDP 5.0

servers is 44 bytes (including the mandatory null terminator). RDP 5.1 and later allow a maximum length of 512 bytes (including the mandatory null terminator). The field must contain at least a null terminator character in ANSI or Unicode format (depending on the presence of the INFO_UNICODE flag).

Password (variable): Variable length logon password of the user (the length in bytes is given by the **cbPassword** field). The maximum length allowed by RDP 4.0 and RDP 5.0 servers is 32 bytes (including the mandatory null terminator). RDP 5.1 and later allow a maximum length of 512 bytes (including the mandatory null terminator). The field must contain at least a null terminator character in ANSI or Unicode format (depending on the presence of the INFO_UNICODE flag).

AlternateShell (variable): Variable length path to the executable file of an alternate shell, e.g. "c:\dir\prog.exe" (the length in bytes is given by the **cbAlternateShell** field). The maximum allowed length is 512 bytes (including the mandatory null terminator). This field MUST only be initialized if the client is requesting a shell other than the default. The field must contain at least a null terminator character in ANSI or Unicode format (depending on the presence of the INFO_UNICODE flag).

WorkingDir (variable): Variable length directory that contains the executable file specified in the **AlternateShell** field or any related files (the length in bytes is given by the **cbWorkingDir** field). The maximum allowed length is 512 bytes (including the mandatory null terminator). This field MAY be initialized if the client is requesting a shell other than the default. The field must contain at least a null terminator character in ANSI or Unicode format (depending on the presence of the INFO_UNICODE flag).

extraInfo (variable): Optional and variable length extended information added in RDP 5.0, as specified in section 2.2.1.11.1.1.1.

2.2.1.11.1.1.1 Extended Info Packet (TS_EXTENDED_INFO_PACKET)

The TS_EXTENDED_INFO_PACKET structure contains user information specific to RDP 5.0 and later.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">clientAddressFamily</td><td colspan="16">cbClientAddress</td></tr>
<tr><td colspan="32">clientAddress (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">cbClientDir</td><td colspan="16">clientDir (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">clientTimeZone</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

<table>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">(clientTimeZone cont'd for 35 rows)</td></tr>
<tr><td colspan="2">clientSessionId</td></tr>
<tr><td colspan="2">performanceFlags</td></tr>
<tr><td>cbAutoReconnectLen</td><td>autoReconnectCookie (optional)</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td colspan="2">...</td></tr>
<tr><td>...</td><td>reserved1 (optional)</td></tr>
<tr><td>reserved2 (optional)</td><td></td></tr>
</table>

clientAddressFamily (2 bytes): A 16-bit unsigned integer. The numeric socket descriptor for the client address type. RDP only uses TCP/IP, so this field MUST be set to AF_INET (0x0002).

cbClientAddress (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **clientAddress** field. This size includes the length of the mandatory null terminator.

clientAddress (variable): Variable length textual representation of the client IP address. The maximum allowed length (including the mandatory null terminator) is 64 bytes for versions of RDP prior to 6.0, and 80 bytes for RDP 6.0 and later.

cbClientDir (2 bytes): A 16-bit unsigned integer. The size in bytes of the character data in the **clientDir** field. This size includes the length of the mandatory null terminator.

clientDir (variable): Variable length directory that contains the folder path on the client machine from which the client software is being run. The maximum allowed length is 512 bytes (including the mandatory null terminator).

clientTimeZone (172 bytes): A TS_TIME_ZONE_INFORMATION (section 2.2.1.11.1.1.1.1) structure that contains time zone information for a client. This packet is ignored by RDP 5.1 servers and earlier, but is used by RDP 5.2 and later servers.

clientSessionId (4 bytes): A 32-bit unsigned integer. This field was added in RDP 5.1 and is currently ignored by the server. It SHOULD be set to 0.

performanceFlags (4 bytes): A 32-bit unsigned integer. It specifies a list of server desktop shell features to disable in the remote session for improving bandwidth performance. Used by RDP 5.1 and later servers.

Flag	Meaning
PERF_DISABLE_WALLPAPER 0x00000001	Disable desktop wallpaper.
PERF_DISABLE_FULLWINDOWDRAG 0x00000002	Disable full-window drag (only the window outline is displayed when the window is moved).
PERF_DISABLE_MENUANIMATIONS 0x00000004	Disable menu animations.
PERF_DISABLE_THEMING 0x00000008	Disable user interface themes.
PERF_RESERVED1 0x00000010	Reserved for future use.
PERF_DISABLE_CURSOR_SHADOW 0x00000020	Disable mouse cursor shadows.
PERF_DISABLE_CURSORSETTINGS 0x00000040	Disable cursor blinking.
PERF_ENABLE_FONT_SMOOTHING 0x00000080	Enable font smoothing.
PERF_ENABLE_DESKTOP_COMPOSITION 0x00000100	Enable Desktop Composition.
PERF_RESERVED2 0x80000000	Reserved for future use.

cbAutoReconnectLen (2 bytes): A 16-bit unsigned integer. The size in bytes of the cookie specified by the **autoReconnectCookie** field. This field is only read by RDP 5.2 and later servers.

autoReconnectCookie (28 bytes): Buffer containing an ARC_CS_PRIVATE_PACKET (section 2.2.4.3) structure. This buffer is a unique cookie that allows a disconnected client to seamlessly reconnect to a previously established session (see section 5.5 for more details). The **autoReconnectCookie** field is only read by RDP 5.2 and later servers.

reserved1 (2 bytes): This field is reserved for future use and has no affect on RDP wire traffic. If this field is present, the reserved2 field MUST be present.

reserved2 (2 bytes): This field is reserved for future use and has no affect on RDP wire traffic. This field MUST be present if the reserved1 field is present.

2.2.1.11.1.1.1.1 Time Zone Information (TS_TIME_ZONE_INFORMATION)

The TS_TIME_ZONE_INFORMATION structure contains client time zone information.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Bias																															
StandardName																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(StandardName cont'd for 8 rows)																															
StandardDate																															
...																															
...																															
...																															

StandardBias
DaylightName
...
...
...
...
...
...
...
(DaylightName cont'd for 8 rows)
DaylightDate
...
...
...
DaylightBias

Bias (4 bytes): A 32-bit unsigned integer. Current bias for local time translation on the client, in minutes. The bias is the difference, in minutes, between Coordinated Universal Time (UTC) and local time. All translations between UTC and local time are based on the following formula:

UTC = local time + bias

StandardName (64 bytes): A description for standard time on the client. For example, this field could contain the string "Pacific Standard Time" to indicate Pacific Standard Time. An array of 32 Unicode characters.

StandardDate (16 bytes): A TS_SYSTEMTIME (section 2.2.1.11.1.1.1.1.1) structure that contains the date and local time when the transition from daylight saving time to standard time occurs on the client. If this field is specified, the **DaylightDate** field is also specified.

StandardBias (4 bytes): A 32-bit unsigned integer. Bias value to be used during local time translations that occur during standard time. This field should be ignored if a value is not supplied in the **StandardDate** field. This value is added to the value of the **Bias** field to form the bias used during standard time. In most time zones, the value of this field is 0.

DaylightName (64 bytes): A description for daylight time on the client. For example, this field could contain "Pacific Daylight Time" to indicate Pacific Daylight Time. An array of 32 Unicode characters.

DaylightDate (16 bytes): A TS_SYSTEMTIME that contains a date and local time when the transition from standard time to daylight saving time occurs on the client. If this field is specified, the **StandardDate** field is also specified.

DaylightBias (4 bytes): A 32-bit unsigned integer. Bias value to be used during local time translations that occur during daylight saving time. This field should be ignored if a value for the **DaylightDate** field is not supplied. This value is added to the value of the **Bias** field to form the bias used during daylight saving time. In most time zones, the value of this field is 60.

2.2.1.11.1.1.1.1.1 System Time (TS_SYSTEMTIME)

The TS_SYSTEMTIME structure contains a date and local time when the transition occurs between daylight saving time to standard time occurs or standard time to daylight saving time.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">wYear</td><td colspan="16">wMonth</td></tr>
<tr><td colspan="16">wDayOfWeek</td><td colspan="16">wDay</td></tr>
<tr><td colspan="16">wHour</td><td colspan="16">wMinute</td></tr>
<tr><td colspan="16">wSecond</td><td colspan="16">wMilliseconds</td></tr>
</table>

wYear (2 bytes): A 16-bit unsigned integer. The year when transition occurs (1601 to 30827).

wMonth (2 bytes): A 16-bit unsigned integer. The month when transition occurs.

Value	Meaning
1	January
2	February
3	March
4	April
5	May
6	June

Value	Meaning
7	July
8	August
9	September
10	October
11	November
12	December

wDayOfWeek (2 bytes): A 16-bit unsigned integer. The day of the week when transition occurs.

Value	Meaning
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

wDay (2 bytes): A 16-bit unsigned integer. The occurrence of **wDayOfWeek** within the month when the transition takes place.

Value	Meaning
1	First occurrence of **wDayOfWeek**
2	Second occurrence of **wDayOfWeek**
3	Third occurrence of **wDayOfWeek**
4	Fourth occurrence of **wDayOfWeek**
5	Last occurrence of **wDayOfWeek**

wHour (2 bytes): A 16-bit unsigned integer. The hour when transition occurs (0 to 23).

wMinute (2 bytes): A 16-bit unsigned integer. The minute when transition occurs (0 to 59).

wSecond (2 bytes): A 16-bit unsigned integer. The second when transition occurs (0 to 59).

wMilliseconds (2 bytes): A 16-bit unsigned integer. The millisecond when transition occurs (0 to 999).

2.2.1.12 Server License Error PDU - Valid Client

The License Error (Valid Client) PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Licensing phase (see section 1.3.1.1). This licensing PDU indicates that the server will not issue the client a license to store and that the Licensing Phase has ended successfully. This is one possible message that may be sent during the Licensing Phase (see [MS-RDPELE] for a detailed discussion of the Remote Desktop Protocol: Licensing Extension).

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
tpktHeader																															
x224Data																								mcsSDin (variable)							
...																															
securityHeader (variable)																															
...																															
preamble																															
validClientMessage (variable)																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Demand Active PDU (section 2.2.1.13.1) data.

securityHeader (variable): Security header. This field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_NONE (0) or ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3) and the embedded **flags** field contains the SEC_ENCRYPT (0x0008) flag.

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4) and the embedded **flags** field contains the SEC_ENCRYPT (0x0008) flag.

If the Encryption Level is set to ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), ENCRYPTION_LEVEL_HIGH (3) or ENCRYPTION_LEVEL_FIPS (4) and the **flags** field of the security header does not contain the SEC_ENCRYPT (0x0008) flag (the licensing PDU is not encrypted), then the field MUST contain a Basic Security Header. This MUST be the case if SEC_LICENSE_ENCRYPT_SC (0x0200) flag was not set on the Security Exchange PDU (section 2.2.1.10).

The **flags** field of the security header MUST contain the SEC_LICENSE_PKT (0x0080) flag (see Basic (TS_SECURITY_HEADER)).

preamble (4 bytes): Licensing Preamble (section 2.2.1.12.1) structure containing header information. The **bMsgType** field of the preamble structure should be set to ERROR_ALERT (0xFF).

validClientMessage (variable): A Licensing Error Message (section 2.2.1.12.3) structure. The **dwErrorCode** field of the error message structure MUST be set to STATUS_VALID_CLIENT (0x00000007) and the **dwStateTransition** field MUST be set to ST_NO_TRANSITION (0x00000002). The **bbErrorInfo** field MUST contain an empty binary BLOB of type BB_ERROR_BLOB (0x0004).

2.2.1.12.1 Licensing Preamble (LICENSE_PREAMBLE)

The LICENSE_PREAMBLE structure precedes every licensing packet sent on the wire.

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
bMsgType								bVersion								wMsgSize															

bMsgType (1 byte): An 8-bit unsigned integer. A type of the licensing packet. For more details about the different licensing packets, see [MS-RDPELE].

Sent by server:

Value	Meaning
LICENSE_REQUEST 0x01	Indicates a License Request PDU (see [MS-RDPELE]).
PLATFORM_CHALLENGE 0x02	Indicates a Platform Challenge PDU (see [MS-RDPELE]).
NEW_LICENSE 0x03	Indicates a New License PDU (see [MS-RDPELE]).
UPGRADE_LICENSE 0x04	Indicates an Upgrade License PDU (see [MS-RDPELE]).

Sent by client:

Value	Meaning
LICENSE_INFO 0x12	Indicates a License Info PDU (see [MS-RDPELE]).
NEW_LICENSE_REQUEST 0x13	Indicates a New License Request PDU (see [MS-RDPELE]).
PLATFORM_CHALLENGE_RESPONSE 0x15	Indicates a Platform Challenge Response PDU (see [MS-RDPELE]).

Sent by either client or server:

Value	Meaning
ERROR_ALERT 0xFF	Indicates a Licensing Error Message PDU (section 2.2.1.12.3).

bVersion (1 byte): An 8-bit unsigned integer. The license protocol version.

Value	Meaning
PREAMBLE_VERSION_2_0 0x02	RDP 4.0
PREAMBLE_VERSION_3_0 0x03	RDP 5.0 and later

wMsgSize (1 byte): An 8-bit unsigned integer. The size in bytes of the licensing packet (including the size of the preamble).

2.2.1.12.2 Licensing Binary Blob (LICENSE_BINARY_BLOB)

The LICENSE_BINARY_BLOB structure is used to encapsulate arbitrary length binary licensing data.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">wBlobType</td><td colspan="16">wBlobLen</td></tr>
<tr><td colspan="32">blobData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

wBlobType (2 bytes): A 16-bit unsigned integer. The data type of the binary information. If **wBlobLen** is set to 0, then the contents of this field SHOULD be ignored.

Value	Meaning
BB_DATA_BLOB 0x0001	Used by License Info PDU and Platform Challenge Response PDU (see [MS-RDPELE]).

Value	Meaning
BB_RANDOM_BLOB 0x0002	Used by License Info PDU and New License Request PDU (see [MS-RDPELE]).
BB_CERTIFICATE_BLOB 0x0003	Used by License Request PDU (see [MS-RDPELE]).
BB_ERROR_BLOB 0x0004	Used by License Error PDU (section 2.2.1.12).
BB_ENCRYPTED_DATA_BLOB 0x0009	Used by Platform Challenge Response PDU and Server Upgrade License PDU (see [MS-RDPELE]).
BB_KEY_EXCHG_ALG_BLOB 0x000D	Used by License Request PDU (see [MS-RDPELE]).
BB_SCOPE_BLOB 0x000E	Used by License Request PDU ([MS-RDPELE]).
BB_CLIENT_USER_NAME_BLOB 0x000F	Used by New License Request PDU (see [MS-RDPELE]).
BB_CLIENT_MACHINE_NAME_BLOB 0x0010	Used by New License Request PDU (see [MS-RDPELE]).

wBlobLen (2 bytes): A 16-bit unsigned integer. The size in bytes of the binary information in the **blobData** field. If **wBlobLen** is set to 0, then the **blobData** field is not included in the Licensing Binary BLOB structure and the contents of the **wBlobType** field SHOULD be ignored.

blobData (variable): Variable-length binary data. The size of this data in bytes is given by the **wBlobLen** field. If **wBlobLen** is set to 0, then this field is not included in the LICENSE_BINARY_BLOB structure.

2.2.1.12.3 Licensing Error Message (LICENSE_ERROR_MESSAGE)

The LICENSE_ERROR_MESSAGE structure is used to indicate that an error occurred during the licensing protocol. Alternatively, it is also used to notify the peer of important status information.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">dwErrorCode</td></tr>
<tr><td colspan="32">dwStateTransition</td></tr>
<tr><td colspan="32">bbErrorInfo (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

dwErrorCode (4 bytes): A 32-bit unsigned integer. The error or status code.

Sent by client:

Name	Value
ERR_INVALID_SERVER_CERTIFICATE	0x00000001
ERR_NO_LICENSE	0x00000002

Sent by server:

Name	Value
ERR_INVALID_SCOPE	0x00000004
ERR_NO_LICENSE_SERVER	0x00000006
STATUS_VALID_CLIENT	0x00000007
ERR_INVALID_CLIENT	0x00000008
ERR_INVALID_PRODUCTID	0x0000000B
ERR_INVALID_MESSAGE_LEN	0x0000000C

Sent by client and server:

Name	Value
ERR_INVALID_MAC	0x00000003

dwStateTransition (4 bytes): A 32-bit unsigned integer. The licensing state to transition into upon receipt of this message. For more details about how this field is used, see [MS-RDPELE].

Name	Value
ST_TOTAL_ABORT	0x00000001
ST_NO_TRANSITION	0x00000002
ST_RESET_PHASE_TO_START	0x00000003
ST_RESEND_LAST_MESSAGE	0x00000004

bbErrorInfo (variable): A LICENSE_BINARY_BLOB (section 2.2.1.12.2) structure which MUST contain a BLOB of type BB_ERROR_BLOB (0x0004) that includes information relevant to the error code specified in **dwErrorCode**.

2.2.1.13 Mandatory Capability Negotiation

2.2.1.13.1 Server Demand Active PDU

The Demand Active PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Capabilities Negotiation phase (see section 1.3.1.1). It is sent upon successful completion of the Licensing phase (see section 1.3.1.1) of the connection sequence.

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
tpktHeader																															
x224Data																								mcsSDin (variable)							
...																															
securityHeader (variable)																															
...																															
demandActivePduData (variable)																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Demand Active PDU (section 2.2.1.13.1) data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0) then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

demandActivePduData (variable): The actual contents of the Demand Active PDU, as specified in section 2.2.1.13.1.1.

2.2.1.13.1.1 Demand Active PDU Data (TS_DEMAND_ACTIVE_PDU)

The TS_DEMAND_ACTIVE_PDU structure is a standard T.128 Demand Active PDU (see [T128] section 8.4.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareControlHeader</td></tr>
<tr><td colspan="16">...</td><td colspan="16">shareId</td></tr>
<tr><td colspan="16">...</td><td colspan="16">lengthSourceDescriptor</td></tr>
<tr><td colspan="16">lengthCombinedCapabilities</td><td colspan="16">sourceDescriptor (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">numberCapabilities</td><td colspan="16">pad2Octets</td></tr>
<tr><td colspan="32">capabilitySets (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">sessionId</td></tr>
</table>

shareControlHeader (6 bytes): Share Control Header (section 2.2.8.1.1.1.1) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header MUST be set to PDUTYPE_DEMANDACTIVEPDU (1).

shareId (4 bytes): A 32-bit unsigned integer. The share identifier for the packet (see [T128] section 8.4.2 for more information regarding share IDs).

lengthSourceDescriptor (2 bytes): A 16-bit unsigned integer. The size in bytes of the **sourceDescriptor** field.

lengthCombinedCapabilities (2 bytes): A 16-bit unsigned integer. The combined size in bytes of the **numberCapabilities**, **pad2Octets** and **capabilitySets** fields.

sourceDescriptor (variable): The source descriptor. The contents of this field SHOULD be set to { 0x52, 0x44, 0x50, 0x00 }, which is the ANSI representation of the null-terminated string "RDP" in hexadecimal.

numberCapabilities (2 bytes): A 16-bit unsigned integer. The number of capability sets included in the Demand Active_PDU.

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

capabilitySets (variable): An array of TS_CAPS_SET (section 2.2.1.13.1.1.1) structures. Collection of capability sets, each conforming to the TS_CAPS_SET structure. The number of capability sets is specified by the **numberCapabilities** field.

sessionId (4 bytes): A 32-bit unsigned integer. The session identifier. This field is ignored by the client and SHOULD be set to 0x00000000.

2.2.1.13.1.1.1 Capability Set (TS_CAPS_SET)

The TS_CAPS_SET structure is used to describe the type and size of a capability set exchanged between clients and servers. All capability sets conform to this basic structure (see section 2.2.7).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="32">capabilityData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type identifier of the capability set.

Value	Meaning
CAPSTYPE_GENERAL 1	General Capability Set (section 2.2.7.1.1).
CAPSTYPE_BITMAP 2	Bitmap Capability Set (section 2.2.7.1.2).
CAPSTYPE_ORDER 3	Order Capability Set (section 2.2.7.1.3).
CAPSTYPE_BITMAPCACHE 4	Revision 1 Bitmap Cache Capability Set (section 2.2.7.1.5.1).
CAPSTYPE_CONTROL 5	Control Capability Set (section 2.2.7.2.1).
CAPSTYPE_ACTIVATION 7	Window Activation Capability Set (section 2.2.7.2.2).
CAPSTYPE_POINTER 8	Pointer Capability Set (section 2.2.7.1.6).
CAPSTYPE_SHARE 9	Share Capability Set (section 2.2.7.2.3).
CAPSTYPE_COLORCACHE 10	Color Table Cache Capability Set (see [MS-RDPEGDI] section 2.2.1.1).
CAPSTYPE_SOUND	Sound Capability Set (section 2.2.7.1.12).

Value	Meaning
12	
CAPSTYPE_INPUT 13	Input Capability Set (section 2.2.7.1.7).
CAPSTYPE_FONT 14	Font Capability Set (section 2.2.7.2.4).
CAPSTYPE_BRUSH 15	Brush Capability Set (section 2.2.7.1.8).
CAPSTYPE_GLYPHCACHE 16	Glyph Cache Capability Set (section 2.2.7.1.9).
CAPSTYPE_OFFSCREENCACHE 17	Offscreen Bitmap Cache Capability Set (section 2.2.7.1.10).
CAPSTYPE_BITMAPCACHE_HOSTSUPPORT 18	Bitmap Cache Host Support Capability Set (section 2.2.7.1.4).
CAPSTYPE_BITMAPCACHE_REV2 19	Revision 2 Bitmap Cache Capability Set (section 2.2.7.1.5.2).
CAPSTYPE_VIRTUALCHANNEL 20	Virtual Channel Capability Set (section 2.2.7.1.11).
CAPSTYPE_DRAWNINEGRIDCACHE 21	DrawNineGrid Cache Capability Set (see [MS-RDPEGDI] section 2.2.1.2).
CAPSTYPE_DRAWGDIPLUS 22	Draw GDI+ Cache Capability Set (see [MS-RDPEGDI] section 2.2.1.3).
CAPSTYPE_RAIL 23	Remote Programs Capability Set (see [MS-RDPERP]).
CAPSTYPE_WINDOW 24	Window List Capability Set (see [MS-RDPERP]).
CAPSETTYPE_COMPDESK 25	Desktop Composition Extension Capability Set (see 2.2.7.2.7).
CAPSETTYPE_MULTIFRAGMENTUPDATE 26	Multifragment Update Capability Set (see section 2.2.7.2.5).
CAPSETTYPE_LARGE_POINTER 27	Large Pointer Capability Set (see section 2.2.7.2.6).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

capabilityData (variable): Capability set data which conforms to the structure of the type given by the **capabilitySetType** field.

2.2.1.13.2 Client Confirm Active PDU

The Confirm Active PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Capabilities Negotiation phase (see section 1.3.1.1). It is sent as a response to the Demand Active PDU.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
tpktHeader																															
x224Data																								mcsSDrq (variable)							
...																															
securityHeader (variable)																															
...																															
confirmActivePduData (variable)																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Confirm Active PDU (section 2.2.1.13.2) data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

The flags field of the security header SHOULD contain the SEC_RESET_SEQNO and SEC_IGNORE_SEQNO flags (see section 2.2.8.1.1.2.1).

confirmActivePduData (variable): The actual contents of the Confirm Active PDU, as specified in section 2.2.1.13.2.1.

2.2.1.13.2.1 Confirm Active PDU Data (TS_CONFIRM_ACTIVE_PDU)

The TS_CONFIRM_ACTIVE_PDU structure is a standard T.128 Confirm Active PDU (see [T128] section 8.4.1).

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
shareControlHeader																															
...																shareId															
...																originatorId															
lengthSourceDescriptor																lengthCombinedCapabilities															
sourceDescriptor (variable)																															
...																															
numberCapabilities																pad2Octets															
capabilitySets (variable)																															
...																															

shareControlHeader (6 bytes): Share Control Header (section 2.2.8.1.1.1.1) containing information about the packet.

The **type** subfield of the **pduType** field of the Share Control Header MUST be set to PDUTYPE_CONFIRMACTIVEPDU (3).

shareId (4 bytes): A 32-bit unsigned integer. The share identifier for the packet (see [T128] section 8.4.2 for more information regarding share IDs).

originatorId (2 bytes): A 16-bit unsigned integer. The identifier of the packet originator. This field MUST be set to the server channel ID (in Microsoft RDP server implementations, this value is always 0x03EA).

lengthSourceDescriptor (2 bytes): A 16-bit unsigned integer. The size in bytes of the **sourceDescriptor** field.

lengthCombinedCapabilities (2 bytes): A 16-bit unsigned integer. The combined size in bytes of the **numberCapabilities**, **pad2Octets** and **capabilitySets** fields.

sourceDescriptor (variable): Source descriptor. The Microsoft RDP client sets the contents of this field to { 0x4D, 0x53, 0x54, 0x53, 0x43, 0x00 }, which is the ANSI representation of the null-terminated string "MSTSC" in hexadecimal.

numberCapabilities (2 bytes): A 16-bit unsigned integer. Number of capability sets included in the Confirm Active PDU.

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

capabilitySets (variable): An array of TS_CAPS_SET (section 2.2.1.13.1.1.1) structures. Collection of capability sets, each conforming to the TS_CAPS_SET structure. The number of capability sets is specified by the **numberCapabilities** field.

2.2.1.14 Client Synchronize PDU

The Client Synchronize PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Confirm Active PDU.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">synchronizePduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Client Synchronize PDU (section 2.2.1.14.1) data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

The **flags** field of the security header SHOULD contain the SEC_IGNORE_SEQNO flag (see section 2.2.8.1.1.2.1).

synchronizePduData (22 bytes): The actual contents of the Synchronize PDU, as specified in section 2.2.1.14.1.

2.2.1.14.1 Synchronize PDU Data (TS_SYNCHRONIZE_PDU)

The TS_SYNCHRONIZE_PDU structure is a standard T.128 Synchronize PDU (see [T128] section 8.6.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">messageType</td></tr>
<tr><td colspan="16">targetUser</td><td colspan="16"></td></tr>
</table>

shareDataHeader (18 bytes): Share Control Header (section 2.2.8.1.1.1.1) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SYNCHRONIZE (31).

messageType (2 bytes): A 16-bit unsigned integer. The message type. This field MUST be set to SYNCMSGTYPE_SYNC (1).

targetUser (2 bytes): A 16-bit unsigned integer. The MCS channel ID of the target user.

2.2.1.15 Client Control PDU - Cooperate

The Client Control PDU (Cooperate) is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Client Synchronize PDU (section 2.2.1.14).

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
tpktHeader																															
x224Data																								mcsSDrq (variable)							
...																															
securityHeader (variable)																															
...																															
controlPduData																															
...																															
...																															
...																															
...																															
...																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU as described in section 13.7 of [X224].

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Control PDU (section 2.2.1.15.1) data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0) then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

controlPduData (26 bytes): The actual contents of the Control PDU, as specified in section 2.2.1.15.1. The **grantId** and **controlId** fields of the Control PDU Data MUST both be set to zero, while the **action** field MUST be set to CTRLACTION_COOPERATE (0x0004).

2.2.1.15.1 Control PDU Data (TS_CONTROL_PDU)

The TS_CONTROL_PDU structure is a standard T.128 Synchronize PDU (see [T128] section 8.12).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">action</td></tr>
<tr><td colspan="16">grantId</td><td colspan="16">controlId</td></tr>
<tr><td colspan="16">...</td><td colspan="16"></td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_CONTROL (20).

action (2 bytes): A 16-bit unsigned integer. The action code.

Value	Meaning
CTRLACTION_REQUEST_CONTROL 0x0001	Request control
CTRLACTION_GRANTED_CONTROL 0x0002	Granted control
CTRLACTION_DETACH 0x0003	Detach
CTRLACTION_COOPERATE 0x0004	Cooperate

grantId (2 bytes): A 16-bit unsigned integer. The grant identifier.

controlId (4 bytes): A 32-bit unsigned integer. The control identifier.

2.2.1.16 Client Control PDU - Request Control

The Client Control PDU (Request Control) is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Client Control PDU (Cooperate) (section 2.2.1.15).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">controlPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU as described in section 13.7 of [X224].

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Control PDU Data (section 2.2.1.15.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

controlPduData (26 bytes): The actual contents of the Control PDU, as specified in section 2.2.1.15.1. The **grantId** and **controlId** fields of the Control PDU Data MUST both be set to zero, while the **action** field MUST be set to CTRLACTION_REQUEST_CONTROL (0x0001).

2.2.1.17 Client Persistent Key List PDU

The Persistent Key List PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Finalization phase (see section 1.3.1.1). This PDU MAY be sent after transmitting the Client Control PDU (Request Control) (section 2.2.1.16). It MUST NOT be sent to a server which does not advertise support for the Bitmap Host Cache Support Capability Set (section 2.2.7.1.4).

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">persistentKeyListPduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Control PDU Data (section 2.2.1.15.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

  If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

persistentKeyListPduData (variable): The actual contents of the Persistent Key List PDU, as specified in section 2.2.1.17.1.

2.2.1.17.1 Persistent Key List PDU Data (TS_BITMAPCACHE_PERSISTENT_LIST_PDU)

The TS_BITMAPCACHE_PERSISTENT_LIST_PDU structure contains a list of cached bitmap keys saved from Cache Bitmap (Revision 2) Orders (see [MS-RDPEGDI] section 2.2.2.3.1.2.3) which were sent in previous sessions. By including a key in the Persistent Key List PDU Data the client indicates to the server that it has a local copy of the bitmap associated with the key, which implies that the server does not need to retransmit the bitmap to the client (for more details about the Persistent Bitmap Cache, see [MS-RDPEGDI] section 3.1.1.1.1). The bitmap keys can be sent in more than one Persistent Key List PDU, with each PDU being marked using flags in the **bBitMask** field.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">numEntriesCache0</td></tr>
<tr><td colspan="16">numEntriesCache1</td><td colspan="16">numEntriesCache2</td></tr>
<tr><td colspan="16">numEntriesCache3</td><td colspan="16">numEntriesCache4</td></tr>
<tr><td colspan="16">totalEntriesCache0</td><td colspan="16">totalEntriesCache1</td></tr>
<tr><td colspan="16">totalEntriesCache2</td><td colspan="16">totalEntriesCache3</td></tr>
<tr><td colspan="16">totalEntriesCache4</td><td colspan="8">bBitMask</td><td colspan="8">Pad2</td></tr>
<tr><td colspan="16">Pad3</td><td colspan="16">entries (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_BITMAPCACHE_PERSISTENT_LIST (43).

numEntriesCache0 (2 bytes): A 16-bit unsigned integer. The number of entries for bitmap cache 0 in the current Persistent Key List PDU.

numEntriesCache1 (2 bytes): A 16-bit unsigned integer. The number of entries for bitmap cache 1 in the current Persistent Key List PDU.

numEntriesCache2 (2 bytes): A 16-bit unsigned integer. The number of entries for bitmap cache 2 in the current Persistent Key List PDU.

numEntriesCache3 (2 bytes): A 16-bit unsigned integer. The number of entries for bitmap cache 3 in the current Persistent Key List PDU.

numEntriesCache4 (2 bytes): A 16-bit unsigned integer. The number of entries for bitmap cache 4 in the current Persistent Key List PDU.

totalEntriesCache0 (2 bytes): A 16-bit unsigned integer. The total number of entries for bitmap cache 0 expected across the entire sequence of Persistent Key List PDUs. This value will remain unchanged across the sequence.

totalEntriesCache1 (2 bytes): A 16-bit unsigned integer. The total number of entries for bitmap cache 1 expected across the entire sequence of Persistent Key List PDUs. This value will remain unchanged across the sequence.

totalEntriesCache2 (2 bytes): A 16-bit unsigned integer. The total number of entries for bitmap cache 2 expected across the entire sequence of Persistent Key List PDUs. This value will remain unchanged across the sequence.

totalEntriesCache3 (2 bytes): A 16-bit unsigned integer. The total number of entries for bitmap cache 3 expected across the entire sequence of Persistent Key List PDUs. This value will remain unchanged across the sequence.

totalEntriesCache4 (2 bytes): The total number of entries for bitmap cache 4 expected across the entire sequence of Persistent Key List PDUs. This value will remain unchanged across the sequence.

bBitMask (1 byte): An 8-bit unsigned integer. The sequencing flag.

Value	Meaning
PERSIST_FIRST_PDU 0x01	Indicates that the PDU is the first in a sequence of Persistent Key List PDUs.
PERSIST_LAST_PDU 0x02	Indicates that the PDU is the last in a sequence of Persistent Key List PDUs.

If neither PERSIST_FIRST_PDU (0x01) nor PERSIST_LAST_PDU (0x02) are set, then the current PDU is an intermediate packet in a sequence of Persistent Key List PDUs.

Pad2 (1 byte): An 8-bit unsigned integer. Padding. Values in this field are ignored.

Pad3 (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

entries (variable): An array of TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY structures which describe 64-bit bitmap keys. The keys MUST be arranged in order from low cache number to high cache number. For instance, if a PDU contains one key for cache 0 and two keys for cache 1, then **numEntriesCache0** will be set to 1, **numEntriesCache1** will be set to 2, and **numEntriesCache2**, **numEntriesCache3** and **numEntriesCache4** will all be set to zero. The keys will be arranged in the following order: (Cache 0, Key 1), (Cache 1, Key 1), (Cache 1, Key 2).

2.2.1.17.1.1 Persistent List Entry (TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY)

The TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY structure contains a 64-bit bitmap key to be sent back to the server.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Key1																															
Key2																															

Key1 (4 bytes): Low 32 bits of the 64-bit persistent bitmap cache key.

Key2 (4 bytes): A 32-bit unsigned integer. High 32 bits of the 64-bit persistent bitmap cache key.

2.2.1.18 Client Font List PDU

The Font List PDU is a Standard RDP Connection Sequence PDU sent from client to server during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting a Persistent Key List PDU (section 2.2.1.17) or, if the Persistent Key List PDU was not sent, it is sent after transmitting the Client Control PDU (Request Control) (section 2.2.1.16).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">fontListPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Font List PDU Data (section 2.2.1.18.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

fontListPduData (26 bytes): The actual contents of the Font List PDU, as specified in section 2.2.1.18.1.

2.2.1.18.1 Font List PDU Data (TS_FONT_LIST_PDU)

The TS_FONT_LIST_PDU structure contains information that is sent to the server for legacy reasons.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">numberFonts</td></tr>
<tr><td colspan="16">totalNumFonts</td><td colspan="16">listFlags</td></tr>
<tr><td colspan="16">entrySize</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_FONTLIST (39).

numberFonts (2 bytes): A 16-bit unsigned integer. The number of fonts. This field SHOULD be set to 0.

totalNumFonts (2 bytes): A 16-bit unsigned integer. The total number of fonts. This field SHOULD be set to 0.

listFlags (2 bytes): A 16-bit unsigned integer. The sequence flags. This field SHOULD be set to 0x0003, which is the logical OR'ed value of FONTLIST_FIRST (0x0001) and FONTLIST_LAST (0x0002).

entrySize (2 bytes): A 16-bit unsigned integer. The entry size. This field SHOULD be set to 0x0032 (50 bytes).

2.2.1.19 Server Synchronize PDU

The Server Synchronize PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Connection Finalization phase (see section 1.3.1.1). It is sent after receiving the Confirm Active PDU (section 2.2.1.13.2).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
tpktHeader																															
x224Data																								mcsSDin (variable)							
...																															
securityHeader (variable)																															
...																															
synchronizePduData																															
...																															
...																															
...																															
...																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in section 7, part 7 of [T125]). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Synchronize PDU Data (section 2.2.1.14.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

synchronizePduData (22 bytes): The actual contents of the Synchronize PDU as described in section 2.2.1.14.1.

2.2.1.20 Server Control PDU - Cooperate

The Server Control PDU (Cooperate) is a Standard RDP Connection Sequence PDU sent from server to client during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Server Synchronize PDU (section 2.2.1.19).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">controlPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Control PDU Data (section 2.2.1.15.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

controlPduData (26 bytes): The actual contents of the Control PDU as described in section 2.2.1.15.1. The **grantId** and **controlId** fields of the Control PDU Data MUST both be set to zero, while the **action** field MUST be set to CTRLACTION_COOPERATE (0x0004).

2.2.1.21 Server Control PDU - Granted Control

The Server Control PDU (Granted Control) is a Standard RDP Connection Sequence PDU sent from server to client during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Server Control PDU (Cooperate) (section 2.2.1.20).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">controlPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Control PDU Data (section 2.2.1.15.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

controlPduData (26 bytes): The actual contents of the Control PDU as described in section 2.2.1.15.1. The **action** field MUST be set to CTRLACTION_GRANTED_CONTROL (0x0002). The **grantId** field MUST be set to the user channel ID (see sections 2.2.1.6 and 2.2.1.7), while the **controlId** field MUST be set to the server channel ID (in Microsoft RDP server implementations this value is always 0x03EA).

2.2.1.22 Server Font Map PDU

The Font Map PDU is a Standard RDP Connection Sequence PDU sent from server to client during the Connection Finalization phase (see section 1.3.1.1). It is sent after transmitting the Server Control PDU (Granted Control) (section 2.2.1.21). This PDU is the last in the connection sequence and signals to the client that it can start sending input PDUs (see section 1.3.5) to the server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">fontMapPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in[X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Font Map PDU Data (section 2.2.1.22.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

fontMapPduData (26 bytes): The actual contents of the Font Map PDU, as specified in section 2.2.1.22.1.

2.2.1.22.1 Font Map PDU Data (TS_FONT_MAP_PDU)

The TS_FONT_MAP_PDU structure contains information that is sent to the server for legacy reasons.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">numberEntries</td></tr>
<tr><td colspan="16">totalNumEntries</td><td colspan="16">mapFlags</td></tr>
<tr><td colspan="16">entrySize</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2). The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_FONTMAP (40).

numberEntries (2 bytes): A 16-bit unsigned integer. The number of fonts. This field SHOULD be set to 0.

totalNumEntries (2 bytes): A 16-bit unsigned integer. The total number of fonts. This field SHOULD be set to 0.

mapFlags (2 bytes): A 16-bit unsigned integer. The sequence flags. This field SHOULD be set to 0x0003, which is the logical OR'ed value of FONTMAP_FIRST (0x0001) and FONTMAP_LAST (0x0002).

entrySize (2 bytes): A 16-bit unsigned integer. The entry size. This field SHOULD be set to 0x0004 (4 bytes).

2.2.2 Disconnection Sequences

2.2.2.1 MCS Disconnect Provider Ultimatum PDU

The MCS Disconnect Provider Ultimatum PDU is an MCS PDU sent as part of the disconnection sequences, as specified in section 1.3.1.4.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsDPum</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="24">...</td><td colspan="8"></td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsDPum (8 bytes): PER-encoded MCS Disconnect Provider Ultimatum PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 4).

2.2.2.2 Client Shutdown Request PDU

The Shutdown Request PDU is sent by the client as part of the disconnection sequences specified in section 1.3.1.4; specifically as part of the User-Initiated on Client disconnect sequence (see section 1.3.1.4.1).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">shutdownRequestPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Shutdown Request PDU Data (section 2.2.2.2.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

shutdownRequestPduData (18 bytes): The actual contents of the Shutdown Request PDU, as specified in section 2.2.2.2.1.

2.2.2.2.1 Shutdown Request PDU Data (TS_SHUTDOWN_REQ_PDU)

The TS_SHUTDOWN_REQ_PDU structure contains the contents of the Shutdown Request PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) with no PDU body.

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
shareDataHeader																															
...																															
...																															
...																															
...																															

shareDataHeader (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SHUTDOWN_REQUEST (36).

2.2.2.3 Server Shutdown Request Denied PDU

The Shutdown Request Denied PDU is sent by the server as part of the disconnection sequences specified in section 1.3.1.4; specifically as part of the "User-Initiated on Client" disconnect sequence (see section 1.3.1.4.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">shutdownRequestDeniedPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Shutdown Request Denied PDU (section 2.2.2.3.1) data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

shutdownRequestDeniedPduData (18 bytes): The actual contents of the Shutdown Request Denied PDU, as specified in section 2.2.2.3.1.

2.2.2.3.1 Shutdown Request Denied PDU Data (TS_SHUTDOWN_DENIED_PDU)

The TS_SHUTDOWN_DENIED_PDU structure contains the contents of the Shutdown Request Denied PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) with no PDU body.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SHUTDOWN_DENIED (37).

2.2.3 Deactivation-Reactivation Sequence

2.2.3.1 Server Deactivate All PDU

The Deactivate All PDU is sent from server to client to indicate that the connection will be dropped or that a capability renegotiation using a Deactivation-Reactivation Sequence (see section 1.3.1.3) will occur.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">deactivateAllPduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Deactivate All PDU data (section 2.2.3.1.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4)) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

deactivateAllPduData (variable): The actual contents of the Deactivate All PDU, as specified in section 2.2.3.1.1.

2.2.3.1.1 Deactivate All PDU Data (TS_DEACTIVATE_ALL_PDU)

The TS_DEACTIVATE_ALL_PDU structure is a standard T.128 Deactivate All PDU (see [T128] section 8.4.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareControlHeader</td></tr>
<tr><td colspan="16">...</td><td colspan="16">shareId</td></tr>
<tr><td colspan="16">...</td><td colspan="16">lengthSourceDescriptor</td></tr>
<tr><td colspan="32">sourceDescriptor (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

shareControlHeader (6 bytes): Share Control Header (section 2.2.8.1.1.1.1) containing information about the packet.

The **type** subfield of the **pduType** field of the Share Control Header MUST be set to TS_PDUTYPE_DEACTIVATEALLPDU (6).

shareId (4 bytes): A 32-bit unsigned integer. The share identifier for the packet (see [T128] section 8.4.2 for more information regarding share IDs).

lengthSourceDescriptor (2 bytes): A 16-bit unsigned integer. The size in bytes of the **sourceDescriptor** field.

sourceDescriptor (variable): Variable number of bytes. The source descriptor. This field SHOULD be set to 0x00.

2.2.4 Auto-Reconnect Sequence

2.2.4.1 Server Auto-Reconnect Status PDU

The Auto-Reconnect Status PDU contains error information after a failed auto-reconnection attempt has taken place.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">arcStatusPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Auto-Reconnect Status PDU data (section 2.2.4.1.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

arcStatusPduData (22 bytes): The actual contents of the Auto-Reconnect Status PDU, as specified in section 2.2.4.1.1.

2.2.4.1.1 Auto-Reconnect Status PDU Data (TS_AUTORECONNECT_STATUS_PDU)

The TS_AUTORECONNECT_STATUS_PDU structure contains the contents of the Auto-Reconnect Status PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) with a status field.

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">arcStatus</td></tr>
<tr><td colspan="16">...</td><td colspan="16"></td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_ARC_STATUS_PDU (50).

arcStatus (4 bytes): A 32-bit unsigned integer. Error code describing the reason for the auto-reconnect failure. Microsoft RDP servers populate this field with an NTSTATUS error code (see [ERRTRANS] for information on translating NTSTATUS error codes to usable text strings) which describes the issue which triggered the error.

2.2.4.2 Server Auto-Reconnect Packet (ARC_SC_PRIVATE_PACKET)

The ARC_SC_PRIVATE_PACKET structure contains server-supplied information used to seamlessly re-establish a client session connection after network interruption. It is sent as part of the Save Session Info PDU logon information (see section 2.2.10.1.1.4).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">cbLen</td></tr>
<tr><td colspan="32">Version</td></tr>
<tr><td colspan="32">LogonId</td></tr>
<tr><td colspan="32">ArcRandomBits</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

cbLen (4 bytes): A 32-bit unsigned integer. The length in bytes of the Server Auto-Reconnect packet. This field MUST be set to 0x0000001C (28 bytes).

Version (4 bytes): A 32-bit unsigned integer. The value representing the auto-reconnect version.

Value	Meaning
AUTO_RECONNECT_VERSION_1 0x00000001	Version 1 of auto-reconnect.

LogonId (4 bytes): A 32-bit unsigned integer. The session identifier for reconnection.

ArcRandomBits (16 bytes): Byte buffer containing a 16-byte random number generated as a key for secure session reconnection (see section 5.5).

2.2.4.3 Client Auto-Reconnect Packet (ARC_CS_PRIVATE_PACKET)

The ARC_CS_PRIVATE_PACKET structure contains the client response cookie used to seamlessly re-establish a client session connection after network interruption. It is sent as part of the extended information of the Client Info PDU (section 2.2.1.11.1.1).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">cbLen</td></tr>
<tr><td colspan="32">Version</td></tr>
<tr><td colspan="32">LogonId</td></tr>
<tr><td colspan="32">SecurityVerifier</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

cbLen (4 bytes): A 32-bit unsigned integer. The length in bytes of the Client Auto-Reconnect Packet. This field MUST be set to 0x0000001C (28 bytes).

Version (4 bytes): A 32-bit unsigned integer. The value representing the auto-reconnect version.

Value	Meaning
AUTO_RECONNECT_VERSION_1 0x00000001	Version 1 of auto-reconnect.

LogonId (4 bytes): A 32-bit unsigned integer. The session identifier for reconnection.

SecurityVerifier (16 bytes): Byte buffer containing a 16-byte verifier value derived using cryptographic methods (as specified in section 5.5) from the **ArcRandomBits** field of the Server Auto-Reconnect packet.

2.2.5 Server Error Reporting

2.2.5.1 Server Set Error Info PDU

The Set Error Info PDU is sent by the server when there is a connection or disconnection failure. This PDU is only sent to clients which have indicated that they are capable of handling error reporting using the RNS_UD_CS_SUPPORT_ERRINFO_PDU flag in the Client Core Data (section 2.2.1.3.2).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">errorInfoPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Set Error Info PDU Data (section 2.2.5.1.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

errorInfoPduData (22 bytes): The actual contents of the Set Error Info PDU, as specified in section 2.2.5.1.1.

2.2.5.1.1 Set Error Info PDU Data (TS_SET_ERROR_INFO_PDU)

The TS_SET_ERROR_INFO_PDU structure contains the contents of the Set Error Info PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) with an error value field.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">errorInfo</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SET_ERROR_INFO_PDU (47).

errorInfo (4 bytes): A 32-bit unsigned integer. Error code.

Protocol-independent codes:

Name	Value
ERRINFO_NOERROR	0x00000000
ERRINFO_RPC_INITIATED_DISCONNECT	0x00000001
ERRINFO_RPC_INITIATED_LOGOFF	0x00000002
ERRINFO_IDLE_TIMEOUT	0x00000003
ERRINFO_LOGON_TIMEOUT	0x00000004
ERRINFO_DISCONNECTED_BY_OTHERCONNECTION	0x00000005
ERRINFO_OUT_OF_MEMORY	0x00000006

Name	Value
ERRINFO_SERVER_DENIED_CONNECTION	0x00000007
ERRINFO_SERVER_DENIED_CONNECTION_FIPS	0x00000008
ERRINFO_SERVER_INSUFFICIENT_PRIVILEGES	0x00000009
ERRINFO_SERVER_FRESH_CREDENTIALS_REQUIRED	0x0000000A

Protocol-independent licensing codes:

Name	Value
ERRINFO_LICENSE_INTERNAL	0x00000100
ERRINFO_LICENSE_NO_LICENSE_SERVER	0x00000101
ERRINFO_LICENSE_NO_LICENSE	0x00000102
ERRINFO_LICENSE_BAD_CLIENT_MSG	0x00000103
ERRINFO_LICENSE_HWID_DOESNT_MATCH_LICENSE	0x00000104
ERRINFO_LICENSE_BAD_CLIENT_LICENSE	0x00000105
ERRINFO_LICENSE_CANT_FINISH_PROTOCOL	0x00000106
ERRINFO_LICENSE_CLIENT_ENDED_PROTOCOL	0x00000107
ERRINFO_LICENSE_BAD_CLIENT_ENCRYPTION	0x00000108
ERRINFO_LICENSE_CANT_UPGRADE_LICENSE	0x00000109
ERRINFO_LICENSE_NO_REMOTE_CONNECTIONS	0x0000010A

Reserved codes:

Name	Value
ERRINFO_SALEM_INVALIDHELPSESSION	0x0000200
ERRINFO_RDPENC_INVALID_CREDENTIALS	0x0000300
TS_ERRINFO_RDPENC_INTERNALERROR	0x0000301

RDP specific codes:

Name	Value
ERRINFO_UNKNOWNPDUTYPE2	0x000010C9
ERRINFO_UNKNOWNPDUTYPE	0x000010CA
ERRINFO_DATAPDUSEQUENCE	0x000010CB
ERRINFO_UNKNOWNFLOWPDU	0x000010CC

Name	Value
ERRINFO_CONTROLPDUSEQUENCE	0x000010CD
ERRINFO_INVALIDCONTROLPDUACTION	0x000010CE
ERRINFO_INVALIDINPUTPDUTYPE	0x000010CF
ERRINFO_INVALIDINPUTPDUMOUSE	0x000010D0
ERRINFO_INVALIDREFRESHRECTPDU	0x000010D1
ERRINFO_CREATEUSERDATAFAILED	0x000010D2
ERRINFO_CONNECTFAILED	0x000010D3
ERRINFO_CONFIRMACTIVEWRONGSHAREID	0x000010D4
ERRINFO_CONFIRMACTIVEWRONGORIGINATOR	0x000010D5
ERRINFO_PERSISTENTKEYPDUBADLENGTH	0x000010DA
ERRINFO_PERSISTENTKEYPDUILLEGALFIRST	0x000010DB
ERRINFO_PERSISTENTKEYPDUTOOMANYTOTALKEYS	0x000010DC
ERRINFO_PERSISTENTKEYPDUTOOMANYCACHEKEYS	0x000010DD
ERRINFO_INPUTPDUBADLENGTH	0x000010DE
ERRINFO_BITMAPCACHEERRORPDUBADLENGTH	0x000010DF
ERRINFO_SECURITYDATATOOSHORT	0x000010E0
ERRINFO_VCHANNELDATATOOSHORT	0x000010E1
ERRINFO_SHAREDATATOOSHORT	0x000010E2
ERRINFO_BADSUPRESSOUTPUTPDU	0x000010E3
ERRINFO_CONFIRMACTIVEPDUTOOSHORT	0x000010E5
ERRINFO_FLOWPDUTOOSHORT	0x000010E6
ERRINFO_CAPABILITYSETTOOSMALL	0x000010E7
ERRINFO_CAPABILITYSETTOOLARGE	0x000010E8
ERRINFO_NOCURSORCACHE	0x000010E9
ERRINFO_BADCAPABILITIES	0x000010EA
ERRINFO_BADUSERDATA	0x000010EB
ERRINFO_VIRTUALCHANNELDECOMPRESSIONERR	0x000010EC
ERRINFO_INVALIDVCCOMPRESSIONTYPE	0x000010ED
ERRINFO_INVALIDCHANNELID	0x000010EF

Name	Value
ERRINFO_VCHANNELSTOOMANY	0x000010F0
ERRINFO_BADSERVERCERTIFICATEDATA	0x000010F2
ERRINFO_REMOTEAPPSNOTENABLED	0x000010F3
ERRINFO_CACHECAPNOTSET	0x000010F4
ERRINFO_BITMAPCACHEERRORPDUBADLENGTH2	0x000010F5
ERRINFO_BITMAPCACHEERRORPDUBADLENGTH3	0x000010F6
ERRINFO_BITMAPCACHEERRORPDUBADLENGTH4	0x000010F7
ERRINFO_BITMAPCACHEERRORPDUBADLENGTH5	0x000010F8
ERRINFO_BADUSERDATA2	0x000010FE
ERRINFO_BADUSERDATA3	0x000010FF
ERRINFO_BADUSERDATA4	0x00001100
ERRINFO_BADUSERDATA5	0x00001101
ERRINFO_BADUSERDATA6	0x00001102
ERRINFO_BADUSERDATA7	0x00001103
ERRINFO_BADUSERDATA8	0x00001104
ERRINFO_BADUSERDATA9	0x00001105
ERRINFO_BADUSERDATA10	0x00001106
ERRINFO_BADUSERDATA11	0x00001107
ERRINFO_BADUSERDATA12	0x00001108
ERRINFO_BADUSERDATA13	0x00001109
ERRINFO_BADUSERDATA14	0x0000110A
ERRINFO_BADUSERDATA15	0x0000110B
ERRINFO_BADUSERDATA16	0x0000110C
ERRINFO_BADUSERDATA17	0x0000110D
ERRINFO_BADUSERDATA18	0x0000110E
ERRINFO_BADUSERDATA19	0x0000110F
ERRINFO_BADUSERDATA20	0x00001110
ERRINFO_SECURITYDATATOOSHORT2	0x00001111
ERRINFO_SECURITYDATATOOSHORT3	0x00001112

Name	Value
ERRINFO_SECURITYDATATOOSHORT4	0x00001113
ERRINFO_SECURITYDATATOOSHORT5	0x00001114
ERRINFO_SECURITYDATATOOSHORT6	0x00001115
ERRINFO_SECURITYDATATOOSHORT7	0x00001116
ERRINFO_SECURITYDATATOOSHORT8	0x00001117
ERRINFO_SECURITYDATATOOSHORT9	0x00001118
ERRINFO_SECURITYDATATOOSHORT10	0x00001119
ERRINFO_SECURITYDATATOOSHORT11	0x0000111A
ERRINFO_SECURITYDATATOOSHORT12	0x0000111B
ERRINFO_SECURITYDATATOOSHORT13	0x0000111C
ERRINFO_SECURITYDATATOOSHORT14	0x0000111D
ERRINFO_SECURITYDATATOOSHORT15	0x0000111E
ERRINFO_SECURITYDATATOOSHORT16	0x0000111F
ERRINFO_SECURITYDATATOOSHORT17	0x00001120
ERRINFO_SECURITYDATATOOSHORT18	0x00001121
ERRINFO_SECURITYDATATOOSHORT19	0x00001122
ERRINFO_SECURITYDATATOOSHORT20	0x00001123
ERRINFO_SECURITYDATATOOSHORT21	0x00001124
ERRINFO_SECURITYDATATOOSHORT22	0x00001125
ERRINFO_SECURITYDATATOOSHORT23	0x00001126
ERRINFO_UPDATESESSIONKEYFAILED	0x00001191
ERRINFO_DECRYPTFAILED	0x00001192
ERRINFO_ENCRYPTFAILED	0x00001193
ERRINFO_ENCPKGMISMATCH	0x00001194
ERRINFO_DECRYPTFAILED2	0x00001195

2.2.6 Static Virtual Channels

2.2.6.1 Virtual Channel PDU

The Virtual Channel PDU is sent from client to server or from server to client and is used to transport data between static virtual channel end-points.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsPdu (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">channelPduHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">virtualChannelData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsPdu (variable): If the PDU is being sent from client to server, this field MUST contain a variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the **static virtual channel data**.

If the PDU is being sent from server to client, this field MUST contain a variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the **static virtual channel data**.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the security headers described in Non-FIPS (TS_SECURITY_HEADER1) (section 2.2.8.1.1.2.2).

If the PDU is being sent from client to server:

- The **securityHeader** field will contain a Non-FIPS Security Header if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

If the PDU is being sent from server to client:

- The **securityHeader** field will contain a Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- The **securityHeader** field will contain a Non-FIPS Security Header if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

If the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4) the securityHeader field will contain a FIPS Security Header (section 2.2.8.1.1.2.3).

If Enhanced RDP Security (section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

channelPduHeader (8 bytes): Virtual Channel PDU Header (section 2.2.6.1.1) structure which contains control flags and describes the size of the opaque channel data.

virtualChannelData (variable): Variable length data to be processed by the static virtual channel protocol handler. This field MUST NOT be larger than CHANNEL_CHUNK_LENGTH (1600) bytes in size.

2.2.6.1.1 Channel PDU Header (CHANNEL_PDU_HEADER)

The CHANNEL_PDU_HEADER MUST precede all opaque static virtual channel traffic chunks transmitted via RDP between client and server.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">length</td></tr>
<tr><td colspan="32">flags</td></tr>
</table>

length (4 bytes): A 32-bit unsigned integer. The total length in bytes of the uncompressed channel data, excluding this header. The data can span over multiple frames and the individual chunks will need to be reassembled in that case (see section 1.3.3).

flags (4 bytes): A 32-bit unsigned integer. The channel control flags.

Value	Meaning
CHANNEL_FLAG_FIRST 0x00000001	Indicates that the chunk is the first in a sequence.
CHANNEL_FLAG_LAST 0x00000002	Indicates that the chunk is the last in a sequence.
CHANNEL_FLAG_SHOW_PROTOCOL 0x00000010	The Channel PDU Header MUST be visible to the application endpoint (see section 2.2.1.3.4.1).
CHANNEL_FLAG_SUSPEND 0x00000020	All virtual channel traffic MUST be suspended.

Value	Meaning
CHANNEL_FLAG_RESUME 0x00000040	All virtual channel traffic MUST be resumed.
CHANNEL_PACKET_COMPRESSED 0x00200000	The virtual channel data is compressed. This value corresponds to MPPC bit A (see [RFC2118] Common Details section).
CHANNEL_PACKET_AT_FRONT 0x00400000	The decompressed packet MUST be placed at the beginning of the history buffer. This value corresponds to MPPC bit B (see [RFC2118] Common Details section).
CHANNEL_PACKET_FLUSHED 0x00800000	The history buffer MUST be reinitialized. This value corresponds to MPPC bit C (see [RFC2118] Common Details section).
CompressionTypeMask 0x000F0000	Indicates the compression package which was used to compress the data. See the discussion which follows this table for a list of compression packages.

If neither the CHANNEL_FLAG_FIRST (0x00000001) or the CHANNEL_FLAG_LAST (0x00000002) flag is present, the chunk is from the middle of a sequence.

Possible compression packages codes are:

Value	Meaning
PACKET_COMPR_TYPE_8K 0	MPPC-8K compression (see section 3.1.8.4.1).
PACKET_COMPR_TYPE_64K 1	MPPC 64K compression (see section 3.1.8.4.2).
PACKET_COMPR_TYPE_RDP6 2	RDP 6.0 bulk compression (see [MS-RDPEGDI] section 3.1.8).

Instructions detailing how to compress a data stream are listed in section 3.1.8.2, while decompression of a data stream is described in section 3.1.8.3.

2.2.7 Capability Sets

2.2.7.1 Mandatory Capability Sets

2.2.7.1.1 General Capability Set (TS_GENERAL_CAPABILITYSET)

The TS_GENERAL_CAPABILITYSET structure is used to advertise general characteristics and is based on the capability set specified in [T128] section 8.2.3. This capability is sent by both client and server.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
capabilitySetType																lengthCapability															
osMajorType																osMinorType															
protocolVersion																pad2octetsA															
generalCompressionTypes																extraFlags															
updateCapabilityFlag																remoteUnshareFlag															
generalCompressionLevel																refreshRectSupport								suppressOutputSupport							

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_GENERAL (1).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

osMajorType (2 bytes): A 16-bit unsigned integer. The type of platform.

Value	Meaning
OSMAJORTYPE_UNSPECIFIED 0x0000	Unspecified platform
OSMAJORTYPE_WINDOWS 0x0001	Windows platform
OSMAJORTYPE_OS2 0x0002	OS/2 platform
OSMAJORTYPE_MACINTOSH 0x0003	Macintosh platform
OSMAJORTYPE_UNIX 0x0004	UNIX platform

osMinorType (2 bytes): A 16-bit unsigned integer. The version of the platform specified in the **osMajorType** field.

Value	Meaning
OSMINORTYPE_UNSPECIFIED 0x0000	Unspecified version
OSMINORTYPE_WINDOWS_31X 0x0001	Windows 3.1x

Value	Meaning
TS_OSMINORTYPE_WINDOWS_95 0x0002	Windows 95
TS_OSMINORTYPE_WINDOWS_NT 0x0003	Windows NT
TS_OSMINORTYPE_OS2_V21 0x0004	OS/2 2.1
TS_OSMINORTYPE_POWER_PC 0x0005	PowerPC
TS_OSMINORTYPE_MACINTOSH 0x0006	Macintosh
TS_OSMINORTYPE_NATIVE_XSERVER 0x0007	Native X Server
TS_OSMINORTYPE_PSEUDO_XSERVER 0x0008	Pseudo X Server

protocolVersion (2 bytes): A 16-bit unsigned integer. The protocol version. This field MUST be set to TS_CAPS_PROTOCOLVERSION (0x0200).

pad2octetsA (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

generalCompressionTypes (2 bytes): A 16-bit unsigned integer. General compression types. This field MUST be set to 0.

extraFlags (2 bytes): A 16-bit unsigned integer. General capability information.

Remote Desktop Protocol (RDP) 5.0 and later supports the following flags:

Flag	Meaning
FASTPATH_OUTPUT_SUPPORTED 0x0001	Advertiser supports fast-path output.
NO_BITMAP_COMPRESSION_HDR 0x0400	The 8-byte Compressed Data Header (section 2.2.9.1.1.3.1.2.2) MUST NOT be used in conjunction with compressed bitmap data.

RDP 5.1 and later supports the following additional flags:

Flag	Meaning
SHADOW_COMPRESSION_LEVEL 0x0002	Advertiser supports shadow compression. When this flag is set, the participating shadow client can support data compression during shadowing, provided that the compression level matches among the shadow clients. RDP 5.0 has no data compression for shadowing.
LONG_CREDENTIALS_SUPPORTED 0x0004	Advertiser (client or server) supports long-length credentials for the user name, password, or domain name.

RDP 5.2 and later supports the following additional flags:

Flag	Meaning
AUTORECONNECT_SUPPORTED 0x0008	Advertiser supports session auto-reconnection. This flag allows a disconnected client to seamlessly reconnect to its original session without the user resupplying logon credentials.
ENC_SALTED_CHECKSUM 0x0010	Advertiser supports salted message authentication code (MAC) generation (see section 5.3.6.1.1).

updateCapabilityFlag (2 bytes): A 16-bit unsigned integer. Support for update capability. This field MUST be set to 0.

remoteUnshareFlag (2 bytes): A 16-bit unsigned integer. Support for remote unsharing. This field MUST be set to 0.

generalCompressionLevel (2 bytes): A 16-bit unsigned integer. General compression level. This field MUST be set to 0.

refreshRectSupport (1 byte): An 8-bit unsigned integer. Server-only flag that indicates whether the Refresh Rect PDU (section 2.2.11.2) is supported.

Value	Meaning
FALSE 0x00	Server does not support Refresh Rect PDU.
TRUE 0x01	Server supports Refresh Rect PDU.

suppressOutputSupport (1 byte): An 8-bit unsigned integer. Server-only flag that indicates whether the Suppress Output PDU (section 2.2.11.3) is supported.

Value	Meaning
FALSE 0x00	Server does not support Suppress Output PDU.
TRUE 0x01	Server supports Suppress Output PDU.

2.2.7.1.2 Bitmap Capability Set (TS_BITMAP_CAPABILITYSET)

The TS_BITMAP_CAPABILITYSET structure is used to advertise bitmap-orientated characteristics and is based on the capability set specified in [T128] section 8.2.4. This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">preferredBitsPerPixel</td><td colspan="16">receive1BitPerPixel</td></tr>
<tr><td colspan="16">receive4BitsPerPixel</td><td colspan="16">receive8BitsPerPixel</td></tr>
<tr><td colspan="16">desktopWidth</td><td colspan="16">desktopHeight</td></tr>
<tr><td colspan="16">pad2octets</td><td colspan="16">desktopResizeFlag</td></tr>
<tr><td colspan="16">bitmapCompressionFlag</td><td colspan="8">highColorFlags</td><td colspan="8">drawingFlags</td></tr>
<tr><td colspan="16">multipleRectangleSupport</td><td colspan="16">pad2octetsB</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_BITMAP (2).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

preferredBitsPerPixel (2 bytes): A 16-bit unsigned integer. Preferred bits-per-pixel (bpp) for the session. In RDP 4.0 and 5.0, this field MUST be set to 8 (even for a 16-color session). In RDP 5.1 and later, this field MUST be set to the desktop color depth that the client requested in the Client Core Data (section 2.2.1.3.2).

receive1BitPerPixel (2 bytes): A 16-bit unsigned integer. Indicates whether the client can receive 1 bit-per-pixel. This field is ignored during capability negotiation and SHOULD be set to TRUE (0x0001).

receive4BitsPerPixel (2 bytes): A 16-bit unsigned integer. Indicates whether the client can receive 4 bit-per-pixel. This field is ignored during capability negotiation and SHOULD be set to TRUE (0x0001).

receive8BitsPerPixel (2 bytes): A 16-bit unsigned integer. Indicates whether the client can receive 8 bit-per-pixel. This field is ignored during capability negotiation and SHOULD be set to TRUE (0x0001).

desktopWidth (2 bytes): A 16-bit unsigned integer. The width of the client desktop. This field MAY be set to the desktop width that the client requested in the Client Core Data (see section 2.2.1.3.2).

desktopHeight (2 bytes): A 16-bit unsigned integer. The height of the client desktop. This field MAY be set to the desktop height that the client requested in the Client Core Data.

pad2octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

desktopResizeFlag (2 bytes): A 16-bit unsigned integer. Indicates whether desktop resizing is supported.

Value	Meaning
FALSE 0x0000	Desktop resizing is not supported.
TRUE 0x0001	Desktop resizing is supported.

If a desktop resize occurs, the server will deactivate the session (see section 1.3.1.3), and on session reactivation will specify the new desktop size in the **desktopWidth** and **desktopHeight** fields in the Bitmap Capability Set, along with a value of TRUE for the **desktopResizeFlag** field. The client should check these sizes and, if different from the previous desktop size, resize any windows to support this size.

bitmapCompressionFlag (2 bytes): A 16-bit unsigned integer. Indicates whether the client supports bitmap compression. RDP requires bitmap compression and hence this field MUST be set to TRUE (0x0001). If it is not set to TRUE, the server MUST NOT continue with the connection.

highColorFlags (1 byte): An 8-bit unsigned integer. Client support for 16 bits-per-pixel color modes. This field is ignored during capability negotiation and SHOULD be set to 0.

drawingFlags (1 byte): An 8-bit unsigned integer. Padding. Values in this field are ignored.

Flag	Meaning
DRAW_ALLOW_DYNAMIC_COLOR_FIDELITY 0x02	Indicates support for lossy compression of 32 bpp bitmaps by reducing color-fidelity on a per-pixel basis.
DRAW_ALLOW_SKIP_ALPHA 0x08	Indicates that the client supports the removal of the alpha-channel when compressing 32 bpp bitmaps. In this case the alpha is assumed to be 0xFF, that is, the bitmap is opaque.
DRAW_ALLOW_COLOR_SUBSAMPLING 0x04	Indicates support for chroma subsampling when compressing 32 bpp bitmaps.

Compression of 32 bpp bitmaps is specified in [MS-RDPEGDI] section 3.1.9.

multipleRectangleSupport (2 bytes): A 16-bit unsigned integer. Indicates whether the client supports the use of multiple bitmap rectangles. RDP requires the use of multiple bitmap rectangles and hence this field MUST be set to TRUE (0x0001). If it is not set to TRUE, the server MUST NOT continue with the connection.

pad2octetsB (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.1.3 Order Capability Set (TS_ORDER_CAPABILITYSET)

The TS_ORDER_CAPABILITYSET structure advertises support for primary drawing order-related capabilities and is based on the capability set specified in [T128] section 8.2.5 (for more information about primary drawing orders, see [MS-RDPEGDI] section 2.2.2.3.1.1). This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="32">terminalDescriptor</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">pad4octetsA</td></tr>
<tr><td colspan="16">desktopSaveXGranularity</td><td colspan="16">desktopSaveYGranularity</td></tr>
<tr><td colspan="16">pad2octetsA</td><td colspan="16">maximumOrderLevel</td></tr>
<tr><td colspan="16">numberFonts</td><td colspan="16">orderFlags</td></tr>
<tr><td colspan="32">orderSupport</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">textFlags</td><td colspan="16">pad2octetsB</td></tr>
<tr><td colspan="32">pad4octetsB</td></tr>
</table>

desktopSaveSize	
pad2octetsC	pad2octetsD
textANSICodePage	pad2octetsE

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_ORDER (3).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

terminalDescriptor (16 bytes): A 16 element array of 8-bit unsigned integers. Terminal descriptor. This field is ignored during capability negotiation and SHOULD be set to all zeros.

pad4octetsA (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

desktopSaveXGranularity (2 bytes): A 16-bit unsigned integer. X granularity used in conjunction with the SaveBitmap Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.12). This value is ignored during capability negotiation and assumed to be 1.

desktopSaveYGranularity (2 bytes): A 16-bit unsigned integer. Y granularity used in conjuction with the SaveBitmap Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.12). This value is ignored during capability negotiation and assumed to be 20.

pad2octetsA (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

maximumOrderLevel (2 bytes): A 16-bit unsigned integer. Maximum order level. This value is ignored during capability negotiation and SHOULD be set to ORD_LEVEL_1_ORDERS (1).

numberFonts (2 bytes): A 16-bit unsigned integer. Number of fonts. This value is ignored during capability negotiation and SHOULD be set to 0.

orderFlags (2 bytes): A 16-bit unsigned integer. A 16-bit unsigned integer. Support for drawing order options.

Value	Meaning
NEGOTIATEORDERSUPPORT 0x0002	Indicates support for negotiating drawing orders in the **orderSupport** field. This flag MUST be set in the **orderFlags** field.
ZEROBOUNDSDELTASSUPPORT 0x0008	Indicates support for the order encoding flag for zero bounds delta coordinates (see [MS-RDPEGDI] section 2.2.2.3.1.1.2). This flag MUST be set in the **orderFlags** field.
COLORINDEXSUPPORT 0x0020	Indicates support for sending color indices (not RGB values) in orders.
SOLIDPATTERNBRUSHONLY 0x0040	Indicates that this party can receive only solid and pattern brushes.

orderSupport (32 bytes): An array of 32 bytes indicating server or client support for various primary drawing orders. The indices of this array are the negotiation indices for the primary orders specified in [MS-RDPEGDI] section 2.2.2.3.1.1.

Negotiation Index	Primary Drawing Order
TS_NEG_DSTBLT_INDEX 0x00	DstBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.1).
TS_NEG_PATBLT_INDEX 0x01	PatBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.3).
TS_NEG_SCRBLT_INDEX 0x02	ScrBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.7).
TS_NEG_MEMBLT_INDEX 0x03	MemBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.9).
TS_NEG_MEM3BLT_INDEX 0x04	Mem3Blt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.10).
Unused Index 0x05	Not applicable
Unused Index 0x06	Not applicable
TS_NEG_DRAWNINEGRID_INDEX 0x07	DrawNineGrid Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.21).
TS_NEG_LINETO_INDEX 0x08	LineTo Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.11).
TS_NEG_MULTI_DRAWNINEGRID_INDEX 0x09	MultiDrawNineGrid Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.22).
TS_NEG_OPAQUERECT_INDEX 0x0A	OpaqueRect Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.5).
TS_NEG_SAVEBITMAP_INDEX 0x0B	SaveBitmap Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.12).
Unused Index 0x0C	Not applicable
Unused Index 0x0D	Not applicable
Unused Index 0x0E	Not applicable
TS_NEG_MULTIDSTBLT_INDEX 0x0F	MultiDstBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.2).

Negotiation Index	Primary Drawing Order
TS_NEG_MULTIPATBLT_INDEX 0x10	MultiPatBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.11.2.2).
TS_NEG_MULTISCRBLT_INDEX 0x11	MultiScrBlt Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.8).
TS_NEG_MULTIOPAQUERECT_INDEX 0x12	MultiOpaqueRect Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.6).
TS_NEG_FAST_INDEX_INDEX 0x13	FastIndex Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.14).
TS_NEG_POLYGON_SC_INDEX 0x14	PolygonSC Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.16).
TS_NEG_POLYGON_CB_INDEX 0x15	PolygonCB Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.17).
TS_NEG_POLYLINE_INDEX 0x16	Polyline Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.18).
Unused Index 0x17	Not applicable
TS_NEG_FAST_GLYPH_INDEX 0x18	FastGlyph Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.15).
TS_NEG_ELLIPSE_SC_INDEX 0x19	EllipseSC Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.19).
TS_NEG_ELLIPSE_CB_INDEX 0x1A	EllipseDB Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.20).
TS_NEG_INDEX_INDEX 0x1B	GlyphIndex Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.13).
Unused Index 0x1C	Not applicable
Unused Index 0x1D	Not applicable
Unused Index 0x1E	Not applicable
Unused Index 0x1F	Not applicable

If an order is supported, the byte at the given index contains the value 0x01. Any order not supported by the client causes the server to spend more time and bandwidth using workarounds, such as other primary orders or simply sending screen bitmap data in a Bitmap

Update (see sections 2.2.9.1.1.3.1.2 and 2.2.9.1.2.1.2). If no primary drawing orders are supported, this array can be initialized to all zeros.

textFlags (2 bytes): A 16-bit unsigned integer. Support for text options. This value is ignored during capability negotiation and SHOULD be set to 0.

pad2octetsB (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

pad4octetsB (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

desktopSaveSize (4 bytes): A 32-bit unsigned integer. The maximum usable size of bitmap space for bitmap packing in the SaveBitmap Primary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.1.2.12). This field is ignored by the client and assumed to be 230400 bytes (480 * 480).

pad2octetsC (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

pad2octetsD (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

textANSICodePage (2 bytes): A 16-bit unsigned integer. ANSI codepage descriptor being used by the client (for a list of code pages, see [MSDN-CP]). This field is ignored by the client and SHOULD be set to 0 by the server.

pad2octetsE (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.1.4 Bitmap Cache Host Support Capability Set (TS_BITMAPCACHE_HOSTSUPPORT_CAPABILITYSET)

The TS_BITMAPCACHE_HOSTSUPPORT_CAPABILITYSET structure is used to advertise support for persistent bitmap caching (see [MS-RDPEGDI] section 3.1.1.1.1). This capability set is only sent from server to client.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="8">cacheVersion</td><td colspan="8">pad1</td><td colspan="16">pad2</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_BITMAPCACHE_HOSTSUPPORT (18).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

cacheVersion (1 byte): An 8-bit unsigned integer. Cache version. This field MUST be set to TS_BITMAPCACHE_REV2 (0x01), which implies at a Revision 2 Bitmap Cache (see [MS-RDPEGDI] section 3.1.1.1.1).

pad1 (1 byte): An 8-bit unsigned integer. Padding. Values in this field are ignored.

pad2 (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.1.5 Bitmap Cache Capability Set

2.2.7.1.5.1 Revision 1 (TS_BITMAPCACHE_CAPABILITYSET)

The TS_BITMAPCACHE_CAPABILITYSET structure is used to advertise support for Revision 1 Bitmap Caches (see [MS-RDPEGDI] section 3.1.1.1.1). This capability is only sent from client to server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
capabilitySetType																lengthCapability															
pad1																															
pad2																															
pad3																															
pad4																															
pad5																															
pad6																															
Cache1Entries																Cache1MaximumCellSize															
Cache2Entries																Cache2MaximumCellSize															
Cache3Entries																Cache3MaximumCellSize															

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_BITMAPCACHE (4).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

pad1 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

pad2 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

pad3 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

pad4 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

pad5 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

pad6 (4 bytes): A 32-bit unsigned integer. Padding. Values in this field are ignored.

Cache1Entries (2 bytes): A 16-bit unsigned integer. The number of entries in Bitmap Cache 1 (maximum allowed value is 600 entries).

Cache1MaximumCellSize (2 bytes): A 16-bit unsigned integer. The maximum cell size in Bitmap Cache 1. This field SHOULD be set to 256, corresponding to the number of pixels in a 16 x 16 bitmap.

Cache2Entries (2 bytes): A 16-bit unsigned integer. The number of entries in Bitmap Cache 2 (maximum allowed value is 600 entries).

Cache2MaximumCellSize (2 bytes): A 16-bit unsigned integer. The maximum cell size in Bitmap Cache 2. This field SHOULD be set to 1024, corresponding to the number of pixels in a 32 x 32 bitmap.

Cache3Entries (2 bytes): A 16-bit unsigned integer. The number of entries in Bitmap Cache 3 (maximum allowed value is 65535 entries).

Cache3MaximumCellSize (2 bytes): A 16-bit unsigned integer. The maximum cell size in Bitmap Cache 3. This field SHOULD be set to 4096, corresponding to the number of pixels in a 64 x 64 bitmap.

2.2.7.1.5.2 Revision 2 (TS_BITMAPCACHE_CAPABILITYSET_REV2)

The TS_BITMAPCACHE_CAPABILITYSET_REV2 structure is used to advertise support for Revision 2 Bitmap Caches (see [MS-RDPEGDI] section 3.1.1.1.1). This capability is only sent from client to server.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">CacheFlags</td><td colspan="8">pad2</td><td colspan="8">NumCellCaches</td></tr>
<tr><td colspan="32">CellCacheInfo</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Pad3</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_BITMAPCACHE_REV2 (19).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

CacheFlags (2 bytes): A 16-bit unsigned integer. Properties which apply to all the bitmap caches.

Value	Meaning
PERSISTENT_KEYS_EXPECTED_FLAG 0x0001	Indicates that the client will send a Persistent Key List PDU during the Connection Finalization phase of the Standard RDP Connection Sequence (see section 1.3.1.1).
ALLOW_CACHE_WAITING_LIST_FLAG 0x0002	Indicates that the client supports a cache waiting list. If a waiting list is supported, new bitmaps are cached on the second hit rather than the first (bitmaps must be sent twice before they are cached).

pad2 (1 byte): An 8-bit unsigned integer. Padding. Values in this field are ignored.

NumCellCaches (1 byte): An 8-bit unsigned integer. Number of bitmap caches (with a maximum allowed value of 5). This field SHOULD be set to 3. Note that the bitmap cache cell sizes are not specified; they are assumed to be 256, 1024, and 4096 pixels, in order.

CellCacheInfo (20 bytes): An array of 5 TS_BITMAPCACHE_CELL_CACHE_INFO structures. Contains information about each of the different caches. The number of valid elements in the array is given by the **NumCellCaches** field.

Pad3 (12 bytes): Padding. An array of 8-bit unsigned integers. Values in this field are ignored.

2.2.7.1.5.2.1 Bitmap Cache Cell Info (TS_BITMAPCACHE_CELL_CACHE_INFO)

The TS_BITMAPCACHE_CELL_CACHE_INFO structure contains information about a bitmap cache on the client.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="31">NumEntries</td><td>k</td></tr>
</table>

NumEntries (31 bits): A 31-bit unsigned integer. Indicates the number of entries in the cache.

k (1 bit): A 1-bit flag. Indicates whether the client expects to receive a 64-bit bitmap key in the Cache Bitmap (Revision 2) Secondary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.2.3) for this cache. If this bit is set, 64-bit keys MUST be sent.

2.2.7.1.6 Pointer Capability Set (TS_POINTER_CAPABILITY_SET)

The TS_POINTER_CAPABILITYSET structure advertises pointer cache sizes and flags, and is based on the capability set specified in [T128] section 8.2.11. This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">colorPointerFlag</td><td colspan="16">colorPointerCacheSize</td></tr>
<tr><td colspan="16">pointerCacheSize</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_POINTER (8).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

colorPointerFlag (2 bytes): A 16-bit unsigned integer. Indicates support for a color pointer.

Value	Meaning
FALSE 0x0000	Monochrome mouse cursors are supported.
TRUE	Color mouse cursors are supported.

Value	Meaning
0x0001	

colorPointerCacheSize (2 bytes): A 16-bit unsigned integer. The number of available slots in the 24 bits-per-pixel color pointer cache used to store data received in the Color Pointer Update (section 2.2.9.1.1.4.4).

pointerCacheSize (2 bytes): A 16-bit unsigned integer. The number of available slots in the pointer cache used to store pointer data of arbitrary bit depth received in the New Pointer Update (section 2.2.9.1.1.4.5).

If the Pointer Capability Set sent from the client does not include this field, the server will not use the New Pointer Update.

2.2.7.1.7 Input Capability Set (TS_INPUT_CAPABILITY_SET)

The TS_INPUT_CAPABILITYSET structure is used to advertise support for input formats and devices. This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">inputFlags</td><td colspan="16">pad2octetsA</td></tr>
<tr><td colspan="32">keyboardLayout</td></tr>
<tr><td colspan="32">keyboardType</td></tr>
<tr><td colspan="32">keyboardSubType</td></tr>
<tr><td colspan="32">keyboardFunctionKey</td></tr>
<tr><td colspan="32">imeFileName</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">(imeFileName cont'd for 8 rows)</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_INPUT (13).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

inputFlags (2 bytes): A 16-bit unsigned integer. Input support flags.

Flag	Meaning
INPUT_FLAG_SCANCODES	Indicates support for using scancodes in the Keyboard Event

Flag	Meaning
0x0001	notifications (see sections 2.2.8.1.1.3.1.1 and 2.2.8.1.2.2.2).
INPUT_FLAG_MOUSEX 0x0004	Indicates support for Extended Mouse Event notifications (see sections 2.2.8.1.2.2.4 and 2.2.8.1.1.3.1.4).
INPUT_FLAG_FASTPATH_INPUT 0x0008	Advertised by RDP 5.0 and 5.1 servers. RDP 5.2 and later servers advertise the INPUT_FLAG_FASTPATH_INPUT2 flag to indicate support for fast-path input.
INPUT_FLAG_UNICODE 0x0010	Indicates support for Unicode Keyboard Event notifications (see sections 2.2.8.1.1.3.1.2 and 2.2.8.1.2.2.2).
INPUT_FLAG_FASTPATH_INPUT2 0x0020	Advertised by RDP 5.2 and later servers. Clients that do not support this flag (such as RDP 5.0 and 5.1 clients) will not be able to use fast-path input when connecting to RDP 5.2 and later servers.

The INPUT_FLAG_SCANCODES flag MUST be set by the client as RDP keyboard input is restricted to keyboard scancodes (unlike the code-point or virtual codes supported in section [T128]). The server MUST drop a client which does not advertise this flag.

pad2octetsA (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

keyboardLayout (4 bytes): A 32-bit unsigned integer. Keyboard layout (active input locale identifier). For a list of possible input locales refer to [MSDN-MUI]. This value is only specified in the client Input Capability Set and should correspond with that sent in the Client Core Data (section 2.2.1.3.2).

keyboardType (4 bytes): A 32-bit unsigned integer. Keyboard type.

Value	Meaning
1	IBM PC/XT or compatible (83-key) keyboard
2	Olivetti "ICO" (102-key) keyboard
3	IBM PC/AT (84-key) or similar keyboard
4	IBM enhanced (101- or 102-key) keyboard
5	Nokia 1050 and similar keyboards
6	Nokia 9140 and similar keyboards
7	Japanese keyboard

This value is only specified in the client Input Capability Set and should correspond with that sent in the Client Core Data.

keyboardSubType (4 bytes): A 32-bit unsigned integer. Keyboard subtype (an original equipment manufacturer-dependent value). This value is only specified in the client Input Capability Set and should correspond with that sent in the Client Core Data.

keyboardFunctionKey (4 bytes): A 32-bit unsigned integer. Number of function keys on the keyboard. This value is only specified in the client Input Capability Set and should correspond with that sent in the Client Core Data.

imeFileName (64 bytes): A 64-byte field. Input Method Editor (IME) file name associated with the input locale. This field contains up to 31 Unicode characters plus a null terminator and is only specified in the client Input Capability Set (its contents should correspond with that sent in the Client Core Data).

2.2.7.1.8 Brush Capability Set (TS_BRUSH_CAPABILITYSET)

The TS_BRUSH_CAPABILITYSET advertises client brush support. This capability is only sent from client to server.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
capabilitySetType																lengthCapability															
brushSupportLevel																															

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_BRUSH (15).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

brushSupportLevel (4 bytes): A 32-bit unsigned integer. The maximum brush level supported by the client.

Value	Meaning
BRUSH_DEFAULT 0x00000000	Support for solid-color and monochrome pattern brushes with no caching. This is an RDP 4.0 implementation.
BRUSH_COLOR_8x8 0x00000001	Ability to handle color brushes (4 or 8 bit in RDP 5.0, RDP 5.1 adds 16 and 24 bit) and caching. Brushes are limited to 8-by-8 pixels.
BRUSH_COLOR_FULL 0x00000002	Ability to handle color brushes (4 or 8 bit in RDP 5.0, RDP 5.1 adds 16 and 24 bit) and caching. Brushes can have arbitrary dimensions.

2.2.7.1.9 Glyph Cache Capability Set (TS_GLYPHCACHE_CAPABILITYSET)

The TS_GLYPHCACHE_CAPABILITYSET structure advertises the glyph support level and associated cache sizes. This capability is only sent from client to server.

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="32">GlyphCache</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">(GlyphCache cont'd for 2 rows)</td></tr>
<tr><td colspan="32">FragCache</td></tr>
<tr><td colspan="16">GlyphSupportLevel</td><td colspan="16">pad2octets</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_GLYPHCACHE (16).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

GlyphCache (40 bytes): An array of 10 TS_CACHE_DEFINITION structures. Glyph cache data, up to 10 elements. The maximum number of entries allowed in a cache is 254, and the largest allowed maximum size of an element is 2048 bytes.

FragCache (4 bytes): Fragment cache data. The maximum number of entries allowed in the cache is 256, and the largest allowed maximum size of an element is 256 bytes.

GlyphSupportLevel (2 bytes): A 16-bit unsigned integer. The level of glyph support.

Value	Meaning
GLYPH_SUPPORT_NONE 0	The client does not support glyph caching. All text output will be sent to the client as expensive Bitmap Updates (see sections 2.2.9.1.1.3.1.2 and 2.2.9.1.2.1.2).

Value	Meaning
GLYPH_SUPPORT_PARTIAL 1	Indicates support for Revision 1 Cache Glyph Secondary Drawing Orders (see [MS-RDPEGDI] section 2.2.2.3.1.2.5).
GLYPH_SUPPORT_FULL 2	Indicates support for Revision 1 Cache Glyph Secondary Drawing Orders (see [MS-RDPEGDI] section 2.2.2.3.1.2.5).
GLYPH_SUPPORT_ENCODE 3	Indicates support for Revision 2 Cache Glyph Secondary Drawing Orders (see [MS-RDPEGDI] section 2.2.2.3.1.2.6).

pad2octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.1.9.1 Cache Definition (TS_CACHE_DEFINITION)

The TS_CACHE_DEFINITION structure specifies details about a particular cache in the Glyph Capability Set (section 2.2.7.1.9) structure.

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
CacheEntries																CacheMaximumCellSize															

CacheEntries (2 bytes): A 16-bit unsigned integer. The number of entries in the cache.

CacheMaximumCellSize (2 bytes): A 16-bit unsigned integer. The maximum size in bytes of an entry in the cache.

2.2.7.1.10 Offscreen Bitmap Cache Capability Set (TS_OFFSCREEN_CAPABILITYSET)

The TS_OFFSCREEN_CAPABILITYSET structure is used to advertise support for offscreen bitmap caching (see [MS-RDPEGDI] section 3.1.1.1.5). This capability is only sent from client to server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
capabilitySetType																lengthCapability															
offscreenSupportLevel																															
offscreenCacheSize																offscreenCacheEntries															

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_OFFSCREENCACHE (17).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

offscreenSupportLevel (4 bytes): A 32-bit unsigned integer.

Offscreen bitmap cache support level.

Value	Meaning
FALSE 0x00000000	Offscreen bitmap cache is not supported.
TRUE 0x00000001	Offscreen bitmap cache is supported.

offscreenCacheSize (2 bytes): A 16-bit unsigned integer. The maximum size in kilobytes (KB) of the offscreen bitmap cache (largest allowed value is 7680 KB).

offscreenCacheEntries (2 bytes): A 16-bit unsigned integer. The maximum number of cache entries (largest allowed value is 500 entries)

2.2.7.1.11 Virtual Channel Capability Set (TS_VIRTUALCHANNEL_CAPABILITYSET)

The TS_VIRTUALCHANNEL_CAPABILITYSET structure is used to advertise virtual channel support characteristics. This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="32">flags</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_VIRTUALCHANNEL (20).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

flags (4 bytes): A 32-bit unsigned integer.

Virtual channel compression flags.

Value	Meaning
VCCAPS_NO_COMPR 0x00000000	Virtual channel compression is not supported.
VCCAPS_COMPR_SC 0x00000001	Virtual channel compression is supported for server-to-client traffic. The highest compression level supported by the client is advertised in the Client Info PDU (section 2.2.1.11).
VCCAPS_COMPR_CS_8K 0x00000002	Virtual channel compression is supported for client-to-server traffic. The compression level is implicitly limited to MPPC-8K for scalability reasons.

If the client-to-server Virtual Channel Capability Set does not contain the VCCAPS_COMPR_SC flag, the server will not compress any server-to-client virtual channel traffic. Similarly, if the server-to-client Virtual Channel Capability Set does not contain the VCCAPS_COMPR_CS_8K flag, the client will not compress any client-to-server virtual channel traffic.

2.2.7.1.12 Sound Capability Set (TS_SOUND_CAPABILITYSET)

The TS_SOUND_CAPABILITYSET structure advertises the ability to play a "beep" sound. This capability is only sent from client to server.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">soundFlags</td><td colspan="16">pad2octetsA</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_SOUND (12).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

soundFlags (2 bytes): A 16-bit unsigned integer. Support for sound options.

Flag	Meaning
SOUND_BEEPS_FLAG 0x0001	Playing a beep sound is supported.

If the client advertises support for beeps, it must support the Server Play Sound PDU (section 2.2.9.1.1.5).

pad2octetsA (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.2 Optional Capability Sets

2.2.7.2.1 Control Capability Set (TS_CONTROL_CAPABILITYSET)

The TS_CONTROL_CAPABILITYSET structure is used by the client to advertise control capabilities and is fully described in [T128] section 8.2.10. This capability is only sent from client to server and the server ignores its contents.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">controlFlags</td><td colspan="16">remoteDetachFlag</td></tr>
<tr><td colspan="16">controlInterest</td><td colspan="16">detachInterest</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_CONTROL (5).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

controlFlags (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to 0.

remoteDetachFlag (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to FALSE (0x0000).

controlInterest (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to CONTROLPRIORITY_NEVER (0x0002).

detachInterest (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to CONTROLPRIORITY_NEVER (0x0002).

2.2.7.2.2 Window Activation Capability Set (TS_WINDOWACTIVATION_CAPABILITYSET)

The TS_WINDOWACTIVATION_CAPABILITYSET structure is used by the client to advertise window activation characteristics capabilities and is fully specified in [T128] section 8.2.9. This capability is only sent from client to server and the server ignores its contents.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">helpKeyFlag</td><td colspan="16">helpKeyIndexFlag</td></tr>
<tr><td colspan="16">helpExtendedKeyFlag</td><td colspan="16">windowManagerKeyFlag</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_ACTIVATION (7).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

helpKeyFlag (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to FALSE (0x0000).

helpKeyIndexFlag (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to FALSE (0x0000).

helpExtendedKeyFlag (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to FALSE (0x0000).

windowManagerKeyFlag (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to FALSE (0x0000).

2.2.7.2.3 Share Capability Set (TS_SHARE_CAPABILITYSET)

The TS_SHARE_CAPABILITYSET structure is used to advertise the channel ID of the sender and is fully specified in [T128] section 8.2.12. This capability is sent by both client and server.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">nodeId</td><td colspan="16">pad2octets</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_SHARE (9).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

nodeId (2 bytes): A 16-bit unsigned integer. This field SHOULD be set to 0 by the client and to the server channel ID by the server (in Microsoft RDP server implementations, this value is always 0x03EA).

pad2octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.2.4 Font Capability Set (TS_FONT_CAPABILITYSET)

The TS_FONT_CAPABILITYSET structure is used to advertise font support options. This capability is sent by both client and server.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">fontSupportFlags</td><td colspan="16">pad2octets</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_FONT (14).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

fontSupportFlags (2 bytes): A 16-bit unsigned integer. The font support options. This field SHOULD be set to FONTSUPPORT_FONTLIST (0x0001).

pad2octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.7.2.5 Multifragment Update Capability Set (TS_MULTIFRAGMENTUPDATE_CAPABILITYSET)

The TS_MULTIFRAGMENTUPDATE_CAPABILITYSET structure is used to specify capabilities related to the fragmentation and reassembly of Fast-Path Updates (see section 2.2.9.1.2.1). This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="32">MaxRequestSize</td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. Type of the capability set. This field MUST be set to CAPSETTYPE_MULTIFRAGMENTUPDATE (26).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

MaxRequestSize (4 bytes): A 32-bit unsigned integer. The size of the buffer that MUST be used to reassemble the fragments of a Fast-Path Update (see section 2.2.9.1.2.1). The size of this buffer places a cap on the size of the largest Fast-Path Update that can be fragmented (there MUST always be enough buffer space to hold all of the related Fast-Path Update fragments for reassembly).

2.2.7.2.6 Large Pointer Capability Set (TS_LARGE_POINTER_CAPABILITYSET)

The TS_LARGE_POINTER_CAPABILITYSET structure is used to specify capabilities related to large mouse pointer shape support. This capability is sent by both client and server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">capabilitySetType</td><td colspan="16">lengthCapability</td></tr>
<tr><td colspan="16">largePointerSupportFlags</td><td colspan="16"></td></tr>
</table>

capabilitySetType (2 bytes): A 16-bit unsigned integer. Type of the capability set. This field MUST be set to CAPSETTYPE_LARGE_POINTER (27).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data, including the size of the capabilitySetType and lengthCapability fields.

largePointerSupportFlags (2 bytes): Support for large pointer shapes.

Value	Meaning
LARGE_POINTER_FLAG_96x96 0x00000001	96 pixel by 96 pixel mouse pointer shapes are supported.

Mouse pointer shapes are used by the following pointer updates:

- Color Pointer Update (see section 2.2.9.1.1.4.4)
- New Pointer Update (see section 2.2.9.1.1.4.5)
- Fast-Path Color Pointer Update (see section 2.2.9.1.2.1.7)
- Fast-Path New Pointer Update (see section 2.2.9.1.2.1.8)

The pointer shape data is contained within the Color Pointer Update structure (see section 2.2.9.1.1.4.4) encapsulated by each of these updates.

2.2.7.2.7 Desktop Composition Capability Set (TS_COMPDESK_CAPABILITYSET)

The TS_COMPDESK_CAPABILITYSET structure is used to support desktop composition. This capability is sent by both client and server.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
capabilitySetType																lengthCapability															
CompDeskSupportLevel																															

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of capability set. This field MUST be set to 0x0019 (CAPSETTYPE_COMPDESK).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length in bytes of the capability data.

CompDeskSupportLevel (2 bytes): A 16-bit unsigned integer. The desktop composition support level.

Value	Meaning
COMPDESK_NOT_SUPPORTED 0x0000	The client is not capable of supporting desktop composition services.
COMPDESK_SUPPORTED 0x0001	The client is capable of supporting desktop composition services.

2.2.8 Keyboard and Mouse Input

2.2.8.1 Input PDU Packaging

2.2.8.1.1 Slow-Path (T.128) Formats

2.2.8.1.1.1 Share Headers

2.2.8.1.1.1.1 Share Control Header (TS_SHARECONTROLHEADER)

The TS_SHARECONTROLHEADER header is a T.128 legacy mode header (see [T128] section 8.3) present in slow-path I/O packets.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">totalLength</td><td colspan="16">pduType</td></tr>
<tr><td colspan="16">pduSource</td><td colspan="16"></td></tr>
</table>

totalLength (2 bytes): A 16-bit unsigned integer. The total length of the packet in bytes (the length includes the size of the Share Control Header).

pduType (2 bytes): A 16-bit unsigned integer. It contains the PDU type and protocol version information. The format of the pduType word is described by the following bitmask diagram:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="4">type</td><td colspan="4">versionLow</td><td colspan="8">versionHigh</td><td colspan="16"></td></tr>
</table>

type (4 bits): Least significant 4 bits of the least significant byte

Value	Meaning
PDUTYPE_DEMANDACTIVEPDU 1	Demand Active PDU (section 2.2.1.13.1)
PDUTYPE_CONFIRMACTIVEPDU 3	Confirm Active PDU (section 2.2.1.13.2)
PDUTYPE_DEACTIVATEALLPDU 6	Deactivate All PDU (section 2.2.3.1)
PDUTYPE_DATAPDU 7	Data PDU (actual type is revealed by the **pduType2** field in the Share Data Header (section 2.2.8.1.1.1.2) structure).
PDUTYPE_SERVER_REDIR_PKT 10	Enhanced Security Server Redirection PDU (see [MS-RDPEGDI] section 2.2.3.3.1).

versionLow (4 bits): Most significant 4 bits of the least significant byte.

This field MUST be set to TS_PROTOCOL_VERSION (0x1).

versionHigh (1 byte): Most significant byte. This field MUST be set to 0x00.

pduSource (2 bytes): A 16-bit unsigned integer. The channel ID which is the transmission source of the PDU.

2.2.8.1.1.1.2 Share Data Header (TS_SHAREDATAHEADER)

The TS_SHAREDATAHEADER header is a T.128 legacy mode header (see [T128] section 8.3) present in slow-path I/O packets.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">shareControlHeader</td></tr>
<tr><td colspan="16">...</td><td colspan="16">shareId</td></tr>
<tr><td colspan="16">...</td><td colspan="8">pad1</td><td colspan="8">streamId</td></tr>
<tr><td colspan="16">uncompressedLength</td><td colspan="8">pduType2</td><td colspan="8">compressedType</td></tr>
<tr><td colspan="16">compressedLength</td></tr>
</table>

shareControlHeader (6 bytes): Share Control Header (section 2.2.8.1.1.1.1) containing information about the packet.

shareId (4 bytes): A 32-bit unsigned integer. Share identifier for the packet (see [T128] section 8.4.2 for more information about share IDs).

pad1 (1 byte): An 8-bit unsigned integer. Padding. Values in this field are ignored.

streamId (1 byte): An 8-bit unsigned integer. The stream identifier for the packet.

Value	Meaning
STREAM_UNDEFINED 0x00	Undefined stream priority. This value might be used in the Server Synchronize PDU (see section 2.2.1.19) due to a server-side RDP bug. It MUST not be used in conjunction with any other PDUs.
STREAM_LOW 0x01	Low-priority stream.
STREAM_MED 0x02	Medium-priority stream.
STREAM_HI 0x04	High-priority stream.

uncompressedLength (2 bytes): A 16-bit unsigned integer. The uncompressed length of the packet in bytes.

pduType2 (1 byte): An 8-bit unsigned integer. The type of data PDU.

Value	Meaning
PDUTYPE2_UPDATE 2	Update PDU (section 2.2.9.1.1.3)
PDUTYPE2_CONTROL 20	Control PDU (section 2.2.1.15.1)
PDUTYPE2_POINTER 27	Pointer Update PDU (section 2.2.9.1.1.4)
PDUTYPE2_INPUT 28	Input PDU (section 2.2.8.1.1.3)
PDUTYPE2_SYNCHRONIZE 31	Synchronize PDU (section 2.2.1.14.1)
PDUTYPE2_REFRESH_RECT 33	Refresh Rect PDU (section 2.2.11.2.1)
PDUTYPE2_PLAY_SOUND 34	Play Sound PDU (section 2.2.9.1.1.5.1)
PDUTYPE2_SUPPRESS_OUTPUT 35	Suppress Output PDU (section 2.2.11.3.1)
PDUTYPE2_SHUTDOWN_REQUEST 36	Shutdown Request PDU (section 2.2.2.2.1)
PDUTYPE2_SHUTDOWN_DENIED 37	Shutdown Request Denied PDU (section 2.2.2.3.1)
PDUTYPE2_SAVE_SESSION_INFO 38	Save Session Info PDU (section 2.2.10.1.1)
PDUTYPE2_FONTLIST 39	Font List PDU (section 2.2.1.18.1)
PDUTYPE2_FONTMAP 40	Font Map PDU (section 2.2.1.22.1)
PDUTYPE2_SET_KEYBOARD_INDICATORS 41	Set Keyboard Indicators PDU (section 2.2.8.2.1.1)
PDUTYPE2_BITMAPCACHE_PERSISTENT_LIST 43	Persistent Key List PDU (section 2.2.1.17.1)
PDUTYPE2_BITMAPCACHE_ERROR_PDU 44	Bitmap Cache Error PDU (see [MS-RDPEGDI] section 2.2.2.4.1).
PDUTYPE2_SET_KEYBOARD_IME_STATUS 45	Set Keyboard IME Status PDU (section 2.2.8.2.2.1)
PDUTYPE2_OFFSCRCACHE_ERROR_PDU 46	Offscreen Bitmap Cache Error PDU (see [MS-RDPEGDI] section 2.2.2.4.2).

Value	Meaning
PDUTYPE2_SET_ERROR_INFO_PDU 47	Set Error Info PDU (section 2.2.5.1.1)
PDUTYPE2_DRAWNINEGRID_ERROR_PDU 48	DrawNineGrid Cache Error PDU (see [MS-RDPEGDI] section 2.2.2.4.3).
PDUTYPE2_DRAWGDIPLUS_ERROR_PDU 49	GDI+ Error PDU (see [MS-RDPEGDI] section 2.2.2.4.4).
PDUTYPE2_ARC_STATUS_PDU 50	Auto-Reconnect Status PDU (section 2.2.4.1.1)

compressedType (1 byte): An 8-bit unsigned integer. The compression type and flags specifying the data following the Share Data Header (section 2.2.8.1.1.1.2).

Flag	Meaning
CompressionTypeMask 0x0F	Indicates the package which was used for compression. See the table which follows for a list of compression packages.
PACKET_COMPRESSED 0x20	The payload data is compressed. This value corresponds to MPPC bit C (see [RFC2118] section 3.1).
PACKET_AT_FRONT 0x40	The decompressed packet MUST be placed at the beginning of the history buffer. This value corresponds to MPPC bit B (see [RFC2118] section 3.1).
PACKET_FLUSHED 0x80	The history buffer MUST be reinitialized. This value corresponds to MPPC bit A (see [RFC2118] section 3.1).

Possible compression package values:

Value	Meaning
PACKET_COMPR_TYPE_8K 0	MPPC-8K compression (see section 3.1.8.4.1)
PACKET_COMPR_TYPE_64K 1	MPPC-64K compression (see section 3.1.8.4.2)
PACKET_COMPR_TYPE_RDP6 2	RDP 6.0 bulk compression (see [MS-RDPEGDI] section 3.1.8).

Instructions specifying how to compress a data stream are listed in section 3.1.8.2, while decompression of a data stream is described in section 3.1.8.3.

compressedLength (2 bytes): A 16-bit unsigned integer. The compressed length of the packet in bytes.

2.2.8.1.1.2 Security Headers

2.2.8.1.1.2.1 Basic (TS_SECURITY_HEADER)

The TS_SECURITY_HEADER structure is attached to server-to-client traffic when the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (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
flags																flagsHi															

flags (2 bytes): A 16-bit unsigned integer. The information flags describing properties of the attached data.

Flag	Meaning
SEC_EXCHANGE_PKT 0x0001	Indicates that the packet is a Security Exchange PDU (section 2.2.1.10). This packet type is sent from client to server only. The client only sends this packet if it will be encrypting further communication and standard RDP security methods are in effect.
SEC_ENCRYPT 0x0008	Indicates that encryption is being used for the packet.
SEC_RESET_SEQNO 0x0010	This flag is set for legacy reasons when the packet is a Confirm Active PDU (section 2.2.1.13.2). Otherwise this flag is never used.
SEC_IGNORE_SEQNO 0x0020	This flag is set for legacy reasons when the packet is a Confirm Active PDU or a Client Synchronize PDU (section 2.2.1.14). Otherwise this flag is never used.
SEC_INFO_PKT 0x0040	Indicates that the packet is a Client Info PDU (section 2.2.1.11). This packet type is sent from client to server only. If standard RDP security methods and encryption are in effect, then this packet MUST also be encrypted.
SEC_LICENSE_PKT 0x0080	Indicates that the packet is a Licensing PDU (section 2.2.1.12).
SEC_LICENSE_ENCRYPT_CS 0x0200	Indicates to the client that the server is capable of processing encrypted licensing packets. It is sent by the server together with any licensing PDUs it may send (see section 2.2.1.12).
SEC_LICENSE_ENCRYPT_SC 0x0200	Indicates to the server that the client is capable of processing encrypted licensing packets. It is sent by the client together with the SEC_EXCHANGE_PKT flag when sending a Security Exchange PDU (section 2.2.1.10).
SEC_REDIRECTION_PKT 0x0400	Indicates that the packet is a Standard Security Server Redirection PDU (see [MS-RDPEGDI] section 2.2.3.2.1). The presence of this flag implies that the PDU is encrypted, that is, the SEC_ENCRYPT (0x0008) flag MUST be considered to be set.
SEC_SECURE_CHECKSUM 0x0800	Indicates that the message authentication code (MAC) for the PDU was generated using the "salted MAC generation" technique (see section 5.3.6.1.1). If this flag is not present, then the standard technique was used (see Non-FIPS (section 2.2.8.1.1.2.2) and FIPS (section 2.2.8.1.1.2.3)).
SEC_FLAGSHI_VALID 0x8000	Indicates that the **flagsHi** field contains valid data. If this flag is not set, then the contents of the **flagsHi** field should be ignored.

flagsHi (2 bytes): A 16-bit unsigned integer. This field is reserved for future RDP needs. It is currently unused and all values are ignored. This field will contain valid data only if the SEC_FLAGSHI_VALID bit (0x8000) is set in the **flags** field. If this bit is not set, the **flagsHi** field is uninitialized and can contain any 16-bit unsigned integer value.

2.2.8.1.1.2.2 Non-FIPS (TS_SECURITY_HEADER1)

The TS_SECURITY_HEADER1 structure is attached to all client-to-server traffic when the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3). It is attached to all server-to-client traffic when the Encryption Level is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (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
basicSecurityHeader																															
dataSignature																															
...																															

basicSecurityHeader (4 bytes): Basic Security Header, as specified in section 2.2.8.1.1.2.1.

dataSignature (8 bytes): The message authentication code (MAC) generated over the packet, using one of the techniques described in Non-FIPS.

2.2.8.1.1.2.3 FIPS (TS_SECURITY_HEADER2)

The TS_SECURITY_HEADER2 structure is attached to all traffic when the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

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
basicSecurityHeader																															
length																version								padlen							
dataSignature																															
...																															

basicSecurityHeader (4 bytes): Basic Security Header, as specified in section 2.2.8.1.1.2.1.

length (2 bytes): A 16-bit unsigned integer. The length of the FIPS security header. This field MUST be set to 0x0010 (16 bytes) for legacy reasons.

version (1 byte): An 8-bit unsigned integer. The version of the FIPS header. This field SHOULD be set to TSFIPS_VERSION1 (0x01).

padlen (1 byte): An 8-bit unsigned integer. The number of padding bytes of padding appended to the end of the packet prior to encryption to make sure that the data to be encrypted is a multiple of the 3DES block size (that is, a multiple of 8 as the block size is 64 bits).

dataSignature (8 bytes): The message authentication code (MAC) generated over the packet, using the techniques specified in section 2.2.8.1.1.2.3.

2.2.8.1.1.3 Client Input Event PDU (TS_INPUT_PDU)

The slow-path Input Event PDU is used to transmit input events from client to server.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">numberEvents</td></tr>
<tr><td colspan="16">pad2Octets</td><td colspan="16">slowPathInputEvents (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData**

field of the MCS Send Data Request PDU contains a Security Header and the Shutdown Request PDU Data (section 2.2.2.2.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

- If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_INPUT (28).

numberEvents (2 bytes): A 16-bit unsigned integer. The number of slow-path input events packed together in the **slowPathInputEvents** field.

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

slowPathInputEvents (variable): A collection of slow-path input events to be processed by the server. The number of events present in this array is given by the **numberEvents** field.

2.2.8.1.1.3.1 Slow-Path Input Event (TS_INPUT_EVENT)

The TS_INPUT_EVENT structure is used to wrap event-specific information for all slow-path input events.

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
eventTime																															
messageType																slowPathInputData (variable)															
...																															

eventTime (4 bytes): A 32-bit unsigned integer. The 32-bit timestamp for the input event. This value is ignored by the server.

messageType (2 bytes): A 16-bit unsigned integer. The input event type.

Value	Meaning
INPUT_EVENT_SYNC 0x0000	Indicates a Synchronize Event (section 2.2.8.1.1.3.1.5).
INPUT_EVENT_SCANCODE 0x0004	Indicates a Keyboard Event (section 2.2.8.1.1.3.1.1).
INPUT_EVENT_UNICODE 0x0005	Indicates a Unicode Keyboard Event (section 2.2.8.1.1.3.1.2).
INPUT_EVENT_MOUSE 0x8001	Indicates a Mouse Event (section 2.2.8.1.1.3.1.3).
INPUT_EVENT_MOUSEX 0x8002	Indicates an Extended Mouse Event (section 2.2.8.1.1.3.1.4).

slowPathInputData (variable): TS_KEYBOARD_EVENT, TS_UNICODE_KEYBOARD_EVENT, TS_POINTER_EVENT, TS_POINTERX_EVENT or TS_SYNC_EVENT. The actual contents of the slow-path input event (see sections 2.2.8.1.1.3.1.1 through 2.2.8.1.1.3.1.5).

2.2.8.1.1.3.1.1 Keyboard Event (TS_KEYBOARD_EVENT)

The TS_KEYBOARD_EVENT structure is a standard T.128 Keyboard Event (see [T128] section 8.18.2). RDP keyboard input is restricted to keyboard scancodes, unlike the code-point or virtual codes supported in T.128 (a scancode is an eight-bit value specifying a key location on the keyboard). The server accepts a scancode value and translates it into the correct character depending on the language locale and keyboard layout used in the session.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">keyboardFlags</td><td colspan="16">keyCode</td></tr>
<tr><td colspan="16">pad2Octets</td><td colspan="16"></td></tr>
</table>

keyboardFlags (2 bytes): A 16-bit unsigned integer. The flags describing the keyboard event.

Flag	Meaning
KBDFLAGS_EXTENDED 0x0100	The keystroke message contains an extended scancode. For enhanced 101 and 102-key keyboards, extended keys include the right ALT and right CTRL keys on the main section of the keyboard; the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and ARROW keys in the clusters to the left of the numeric keypad; and the Divide ("/") and ENTER keys in the numeric keypad.
KBDFLAGS_DOWN 0x4000	Indicates that the key was down prior to this event.
KBDFLAGS_RELEASE 0x8000	The absence of this flag indicates a key-down event, while its presence indicates a key-release event.

keyCode (2 bytes): A 16-bit unsigned integer. The scancode of the key which triggered the event.

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.8.1.1.3.1.2 Unicode Keyboard Event (TS_UNICODE_KEYBOARD_EVENT)

The TS_UNICODE_KEYBOARD_EVENT structure is used to transmit a Unicode input code, as opposed to a keyboard scancode. Support for the Unicode Keyboard Event is advertised in the Input Capability Set (section 2.2.7.1.7).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">pad2OctetsA</td><td colspan="16">unicodeCode</td></tr>
<tr><td colspan="16">pad2OctetsB</td></tr>
</table>

pad2OctetsA (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

unicodeCode (2 bytes): A 16-bit unsigned integer. The Unicode character input code.

pad2OctetsB (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.8.1.1.3.1.3 Mouse Event (TS_POINTER_EVENT)

The TS_POINTER_EVENT structure is a standard T.128 Keyboard Event (see [T128] section 8.18.1). RDP adds flags to deal with wheel mice and extended mouse buttons.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">pointerFlags</td><td colspan="16">xPos</td></tr>
<tr><td colspan="16">yPos</td></tr>
</table>

pointerFlags (2 bytes): A 16-bit unsigned integer. The flags describing the pointer event.

Mouse wheel event:

Flag	Meaning
PTRFLAGS_WHEEL 0x0200	The event is a mouse wheel rotation. The only valid flags in a wheel rotation event are PTRFLAGS_WHEEL_NEGATIVE and the WheelRotationMask; all other pointer flags are ignored.
PTRFLAGS_WHEEL_NEGATIVE 0x0100	The PTRFLAGS_ROTATION_MASK value is negative and must be sign-extended before injection at the server.
WheelRotationMask 0x01FF	The bit field describing the number of rotation units the mouse wheel was rotated. The value is negative if the

Flag	Meaning
	PTRFLAGS_WHEEL_NEGATIVE flag is set.

Mouse movement event:

Flag	Meaning
PTRFLAGS_MOVE 0x0800	Indicates that the mouse position should be updated to the location specified by the **xPos** and **yPos** fields.

Mouse button events:

Flag	Meaning
PTRFLAGS_DOWN 0x8000	Indicates that a click event has occurred at the position specified by the **xPos** and **yPos** fields. The button flags indicate which button has been clicked and at least one of these flags MUST be set.
PTRFLAGS_BUTTON1 0x1000	Mouse button 1 (left button) was clicked or released. If the PTRFLAGS_DOWN flag is set, then the button was clicked, otherwise it was released.
PTRFLAGS_BUTTON2 0x2000	Mouse button 2 (right button) was clicked or released. If the PTRFLAGS_DOWN flag is set, then the button was clicked, otherwise it was released.
PTRFLAGS_BUTTON3 0x4000	Mouse button 3 (middle button or wheel) was clicked or released. If the PTRFLAGS_DOWN flag is set, then the button was clicked, otherwise it was released.

xPos (2 bytes): A 16-bit unsigned integer. The x coordinate of the pointer relative to the top-left corner of the server's virtual desktop.

yPos (2 bytes): A 16-bit unsigned integer. The y coordinate of the pointer relative to the top-left corner of the server's virtual desktop.

2.2.8.1.1.3.1.4 Extended Mouse Event (TS_POINTERX_EVENT)

The TS_POINTERX_EVENT structure has the same format as the TS_POINTER_EVENT (section 2.2.8.1.1.3.1.3). The fields and possible field values are all the same, except for the **pointerFlags** field. Support for the Extended Mouse Event is advertised in the Input Capability Set (section 2.2.7.1.7).

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
pointerFlags																xPos															
yPos																															

pointerFlags (2 bytes): A 16-bit unsigned integer. The flags describing the extended mouse event.

Flag	Meaning
PTRXFLAGS_DOWN 0x8000	Indicates that a click event has occurred at the position specified by the **xPos** and **yPos** fields. The button flags indicate which button has been clicked and at least one of these flags MUST be set.
PTRXFLAGS_BUTTON1 0x0001	Extended mouse button 1 was clicked or released. If the PTRXFLAGS_DOWN flag is set, then the button was clicked, otherwise it was released.
PTRXFLAGS_BUTTON2 0x0002	Extended mouse button 2 was clicked or released. If the PTRXFLAGS_DOWN flag is set, then the button was clicked, otherwise it was released.

xPos (2 bytes): A 16-bit unsigned integer. X coordinate of the pointer.

yPos (2 bytes): A 16-bit unsigned integer. Y coordinate of the pointer.

2.2.8.1.1.3.1.5 Synchronize Event (TS_SYNC_EVENT)

The TS_SYNC_EVENT structure is a standard T.128 Input Synchronize Event (see [T128] section 8.18.6). In RDP this event is used to synchronize the values of the toggle keys (that is, Caps Lock) and to reset the server key state to all keys up. This event is typically sent when the client needs to update the server with new settings. In current Microsoft RDP clients this is done whenever the client window loses focus in the client operating system, and then gets focus back with possibly new toggle and shift key values. The sync is then followed immediately with key-down events for whatever keyboard and mouse keys may be down.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">pad2Octets</td><td colspan="16">toggleFlags</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

toggleFlags (4 bytes): A 32-bit unsigned integer.

Flags indicating the "on" status of the keyboard toggle keys.

Flag	Meaning
TS_SYNC_SCROLL_LOCK 0x00000001	Indicates that the Scroll Lock indicator light SHOULD be on.
TS_SYNC_NUM_LOCK 0x00000002	Indicates that the Num Lock indicator light SHOULD be on.
TS_SYNC_CAPS_LOCK 0x00000004	Indicates that the Caps Lock indicator light SHOULD be on.
TS_SYNC_KANA_LOCK 0x00000008	Indicates that the Kana Lock indicator light SHOULD be on.

2.2.8.1.2 Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU)

Fast-path revises client input packets from the first byte with the goal of improving bandwidth. The TPKT (see [T123]), X.224 (see [X224]) and MCS SDrq (see [T125]) headers are replaced, the Security Header (section 2.2.8.1.1.2) is collapsed into the fast-path input header, and the Share Data Header (section 2.2.8.1.1.1.2) is replaced by a new fast-path format. The contents of the input notification events (see section 2.2.8.1.1.3.1) are also changed to reduce their size, particularly by removing or reducing headers. Support for fast-path input is advertised in the Input Capability Set (section 2.2.7.1.7).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">fpInputHeader</td><td colspan="8">length1</td><td colspan="8">length2 (optional)</td><td colspan="8">fipsInformation (optional)</td></tr>
<tr><td colspan="24">...</td><td colspan="8">dataSignature (optional)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="24">...</td><td colspan="8">numberEvents (optional)</td></tr>
<tr><td colspan="32">fpInputEvents (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

fpInputHeader (1 byte): An 8-bit unsigned integer. One-byte bit-packed header. This byte coincides with the first byte of the TPKT Header (see [T123] section 8), which is always 0x03. Three pieces of information are collapsed into this byte:

1. Encryption data
2. Number of events in the fast-path input PDU
3. Action code

The format of the fpInputHeader byte is described by the following bitmask diagram:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="2">actionCode</td><td colspan="4">numberEvents</td><td colspan="2">encryptionFlags</td></tr>
</table>

actionCode (2 bits): A 2-bit code indicating whether the PDU is in fast-path or slow-path format.

Value	Meaning
FASTPATH_INPUT_ACTION_FASTPATH 00	Indicates the PDU is a fast-path input PDU.

Value	Meaning
FASTPATH_INPUT_ACTION_X224 11	Indicates the presence of a TPKT Header initial version byte, which implies that the PDU is a slow-path input PDU (in this case the full value of the initial byte MUST be 0x03).

numberEvents (4 bits): Collapses the number of fast-path input events packed together in the **fpInputEvents** field into 4 bits if the number of events is in the range 1 to 15. If the number of input events is greater than 15, then the **numberEvents** bit field in the fast-path header byte should be set to zero, and the **numberEvents** optional field inserted after the **dataSignature** field. This allows up to 255 input events in one PDU.

encryptionFlags (2 bits): A 2-bit field containing the flags that describe the cryptographic parameters of the PDU.

Flag	Meaning
FASTPATH_INPUT_SECURE_CHECKSUM 01	Indicates that the MAC signature for the PDU was generated using the "salted MAC generation" technique (see section 5.3.6.1.1). If this bit is not set, then the standard technique was used (see sections Non-FIPS (section 2.2.8.1.1.2.2) and FIPS (section 2.2.8.1.1.2.3)).
FASTPATH_INPUT_ENCRYPTED 10	Indicates that the PDU contains an 8-byte message authentication code (MAC) signature after the optional **length2** field (that is, the **dataSignature** field is present) and the contents of the PDU are encrypted using the negotiated encryption package (see sections 5.3.2 and 5.3.6).

length1 (1 byte): An 8-bit unsigned integer. If the most significant bit of the **length1** field is not set, then the size of the PDU is in the range 1 to 127 bytes and the **length1** field contains the overall PDU length (the **length2** field is not present in this case). However, if the most significant bit of the length1 field is set, then the overall PDU length is given by the low 7 bits of the **length1** field concatenated with the 8 bits of the **length2** field, in big-endian order (the **length2** field contains the low-order bits).

length2 (1 byte): An 8-bit unsigned integer. If the most significant bit of the **length1** field is not set, then the **length2** field is not present. If the most significant bit of the **length1** field is set, then the overall PDU length is given by the low 7 bits of the **length1** field concatenated with the 8 bits of the **length2** field, in big-endian order (the **length2** field contains the low-order bits).

fipsInformation (4 bytes): Optional FIPS header information, present when the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4). The fast-path FIPS information structure is specified in section 2.2.8.1.2.1.

dataSignature (8 bytes): The message authentication code (MAC) generated over the packet using one of the techniques described in Non-FIPS (the FASTPATH_INPUT_SECURE_CHECKSUM flag, which is set in the fpInputHeader field, describes the method used to generate the signature). This field is present if the FASTPATH_INPUT_ENCRYPTED flag is set in the fpInputHeader field.

numberEvents (1 byte): An 8-bit unsigned integer. The number of fast-path input events packed together in the **fpInputEvents** field (up to 255). This field is present if the **numberEvents** bit field in the fast-path header byte is zero.

fpInputEvents (variable): A collection of Fast-Path Input Event (section 2.2.8.1.2.2) structures to be processed by the server. The number of events present in this array is given by the **numberEvents** bit field in the fast-path header byte, or by the **numberEvents** field in the Fast-Path Input Event PDU (if it is present).

2.2.8.1.2.1 Fast-Path FIPS Information (TS_FP_FIPS_INFO)

The TS_FP_FIPS_INFO structure contains fast-path information for inclusion in a fast-path header.

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
length																version								padlen							

length (2 bytes): A 16-bit unsigned integer. The length of the FIPS Security Header (section 2.2.8.1.1.2.3). This field MUST be set to 0x0010 (16 bytes).

version (1 byte): An 8-bit unsigned integer. The version of the FIPS Header. This field SHOULD be set to TSFIPS_VERSION1 (0x01).

padlen (1 byte): An 8-bit unsigned integer. The number of padding bytes of padding appended to the end of the packet prior to encryption to make sure that the data to be encrypted is a multiple of the 3DES block size (that is, a multiple of 8 as the block size is 64 bits).

2.2.8.1.2.2 Fast-Path Input Event (TS_FP_INPUT_EVENT)

The TS_FP_INPUT_EVENT structure is used to describe the type and encapsulate the data for a fast-path input event sent from client to server. All fast-path input events conform to this basic structure (see sections 2.2.8.1.2.2.1 to 2.2.8.1.2.2.5).

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
eventHeader								eventData (variable)																							
...																															

eventHeader (1 byte): An 8-bit unsigned integer. One byte bit-packed event header. Two pieces of information are collapsed into this byte:

1. Fast-path input event type

2. Flags specific to the input event

The **eventHeader** field is structured as follows:

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
eventFlags					eventCode																										

eventFlags (5 bits): 5 bits. The flags specific to the input event.

eventCode (3 bits): 3 bits. The type code of the input event.

Value	Meaning
FASTPATH_INPUT_EVENT_SCANCODE 000	Indicates a Fast-Path Keyboard Event (section 2.2.8.1.2.2.1).
FASTPATH_INPUT_EVENT_MOUSE 001	Indicates a Fast-Path Mouse Event (section 2.2.8.1.2.2.3).
FASTPATH_INPUT_EVENT_MOUSEX 010	Indicates a Fast-Path Extended Mouse Event (section 2.2.8.1.2.2.4).
FASTPATH_INPUT_EVENT_SYNC 011	Indicates a Fast-Path Synchronize Event (section 2.2.8.1.2.2.5).
FASTPATH_INPUT_EVENT_UNICODE 100	Indicates a Fast-Path Unicode Keyboard Event (section 2.2.8.1.2.2.2).

eventData (variable): Optional and variable length data specific to the input event.

2.2.8.1.2.2.1 Fast-Path Keyboard Event (TS_FP_KEYBOARD_EVENT)

The TS_FP_KEYBOARD_EVENT structure is the fast-path variant of the TS_KEYBOARD_EVENT.

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
eventHeader								keyCode																							

eventHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **eventHeader** byte field described in section 2.2.8.1.2.2. The **eventCode** bitfield (3 bits in size) MUST be set to FASTPATH_INPUT_EVENT_SCANCODE (0). The flags which can populate the **keyboardFlags** field (specified in section 2.2.8.1.2.2) are pushed into the **eventFlags** bitfield (5 bits in size).

keyCode (1 byte): An 8-bit unsigned integer. The scancode of the key which triggered the event.

2.2.8.1.2.2.2 Fast-Path Unicode Keyboard Event (TS_FP_UNICODE_KEYBOARD_EVENT)

The TS_FP_UNICODE_KEYBOARD_EVENT structure is the fast-path variant of the TS_UNICODE_KEYBOARD_EVENT (section 2.2.8.1.1.3.1.2) structure. Support for the Unicode Keyboard Event is advertised in the Input Capability Set (section 2.2.7.1.7).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">eventHeader</td><td colspan="16">unicodeCode</td><td colspan="8"></td></tr>
</table>

eventHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **eventHeader** byte field, specified in section 2.2.8.1.2.2. The **eventCode** bitfield (3 bits in size) MUST be set to FASTPATH_INPUT_EVENT_UNICODE (4). The **eventFlags** bitfield (5 bits in size) MUST be zeroed out.

unicodeCode (2 bytes): A 16-bit unsigned integer. The Unicode character input code.

2.2.8.1.2.2.3 Fast-Path Mouse Event (TS_FP_POINTER_EVENT)

The TS_FP_POINTER_EVENT structure is the fast-path variant of the TS_POINTER_EVENT (section 2.2.8.1.1.3.1.3) structure.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">eventHeader</td><td colspan="16">pointerFlags</td><td colspan="8">xPos</td></tr>
<tr><td colspan="8">...</td><td colspan="16">yPos</td><td colspan="8"></td></tr>
</table>

eventHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **eventHeader** byte field, specified in section 2.2.8.1.2.2. The **eventCode** bitfield (3 bits in size) MUST be set to FASTPATH_INPUT_EVENT_MOUSE (1). The **eventFlags** bitfield (5 bits in size) MUST be zeroed out.

pointerFlags (2 bytes): A 16-bit unsigned integer. The flags describing the pointer event. The possible flags are identical to those found in the **pointerFlags** field of the TS_POINTER_EVENT structure.

xPos (2 bytes): A 16-bit unsigned integer. The x coordinate of the pointer.

yPos (2 bytes): A 16-bit unsigned integer. The y coordinate of the pointer.

2.2.8.1.2.2.4 Fast-Path Extended Mouse Event (TS_FP_POINTERX_EVENT)

The TS_FP_POINTERX_EVENT structure is the fast-path variant of the TS_POINTERX_EVENT (section 2.2.8.1.1.3.1.4) structure. Support for the Extended Mouse Event is advertised in the Input Capability Set (section 2.2.7.1.7).

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
eventHeader								pointerFlags																xPos							
...								yPos																							

eventHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **eventHeader** byte field, specified in section 2.2.8.1.2.2. The **eventCode** bitfield (3 bits in size) MUST be set to FASTPATH_INPUT_EVENT_MOUSEX (2). The **eventFlags** bitfield (5 bits in size) MUST be zeroed out.

pointerFlags (2 bytes): A 16-bit unsigned integer. The flags describing the pointer event. The possible flags are identical to those found in the **pointerFlags** field of the TS_POINTERX_EVENT structure.

xPos (2 bytes): A 16-bit unsigned integer. The x coordinate of the pointer.

yPos (2 bytes): A 16-bit unsigned integer. The y coordinate of the pointer.

2.2.8.1.2.2.5 Fast-Path Synchronize Event (TS_FP_SYNC_EVENT)

The TS_FP_SYNC_EVENT structure is the fast-path variant of the TS_SYNC_EVENT structure.

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
eventHeader																															

eventHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **eventHeader** byte field, specified in section 2.2.8.1.2.2. The **eventCode** bitfield (3 bits in size) MUST be set to FASTPATH_INPUT_EVENT_SYNC (3). The flags which can populate the **toggleFlags** field (specified in section 2.2.8.1.1.3.1.5) are pushed into the **eventFlags** bitfield (5 bits in size).

2.2.8.2 Keyboard Status PDUs

2.2.8.2.1 Server Set Keyboard Indicators PDU

The Set Keyboard Indicators PDU is sent by the server to synchronize the state of the keyboard toggle keys (Scroll Lock, Num Lock, and so on). It is similar in operation to the Client Synchronize Input Event Notification (see sections 2.2.8.1.1.3.1.5 and 2.2.8.1.2.2.5), but flows in the opposite direction.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
tpktHeader																															
x224Data																								mcsSDin (variable)							
...																															
securityHeader (variable)																															
...																															
setKeyBdIndicatorsPduData																															
...																															
...																															
...																															
...																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Set Keyboard Indicators PDU (section 2.2.8.2.1.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

setKeyBdIndicatorsPduData (22 bytes): The actual contents of the Set Keyboard Indicators PDU, as specified in section 2.2.8.2.1.1.

2.2.8.2.1.1 Set Keyboard Indicators PDU Data (TS_SET_KEYBOARD_INDICATORS_PDU)

The TS_SET_KEYBOARD_INDICATORS_PDU structure contains the actual contents of the Set Keyboard Indicators PDU (section 2.2.8.2.1). The contents of the **LedFlags** field is identical to the flags used in the Client Synchronize Input Event Notification (see section 2.2.8.1.1.3.1.5).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">UnitId</td></tr>
<tr><td colspan="16">LedFlags</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SET_KEYBOARD_INDICATORS (41).

UnitId (2 bytes): A 16-bit unsigned integer. Hardware related value. This field SHOULD be ignored by the client and as a consequence SHOULD be set to 0 by the server.

LedFlags (2 bytes): A 16-bit unsigned integer. The flags indicating the "on" status of the keyboard toggle keys.

Flag	Meaning
TS_SYNC_SCROLL_LOCK 0x0001	Indicates that the Scroll Lock indicator light SHOULD be on.
TS_SYNC_NUM_LOCK 0x0002	Indicates that the Num Lock indicator light SHOULD be on.
TS_SYNC_CAPS_LOCK 0x0004	Indicates that the Caps Lock indicator light SHOULD be on.

Flag	Meaning
TS_SYNC_KANA_LOCK 0x0008	Indicates that the Kana Lock indicator light SHOULD be on.

2.2.8.2.2 Server Set Keyboard IME Status PDU

The Set Keyboard IME Status PDU PDU is sent by the server when the user session employs input method editors (IMEs) and is used to set the IME state. This PDU is accepted and ignored by non-IME aware clients.

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
tpktHeader																															
x224Data																								mcsSDin (variable)							
...																															
securityHeader (variable)																															
...																															
setKeyBdImeStatusPduData																															
...																															
...																															
...																															
...																															
...																															
...																															

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Set Keyboard IME Status PDU data (see section 2.2.8.2.2.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (see section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (see section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

setKeyBdImeStatusPduData (28 bytes): The actual contents of the Set Keyboard IME Status PDU, as specified in section 2.2.8.2.2.1.

2.2.8.2.2.1 Set Keyboard IME Status PDU Data (TS_SET_KEYBOARD_IME_STATUS_PDU)

The TS_SET_KEYBOARD_IME_STATUS_PDU structure contains the actual contents of the Set Keyboard IME Status PDU (section 2.2.8.2.2). On RDP 5.0 and later clients the latter two fields are used as input parameters to a Fujitsu Oyayubi specific IME control function of East Asia IME clients.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">UnitId</td></tr>
<tr><td colspan="32">ImeOpen</td></tr>
<tr><td colspan="32">ImeConvMode</td></tr>
</table>

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SET_KEYBOARD_IME_STATUS (45).

UnitId (2 bytes): A 16-bit unsigned integer. The unit identifier for which the IME message is intended. This field SHOULD be ignored by the client and as a consequence SHOULD be set to 0 by the server.

ImeOpen (4 bytes): A 32-bit unsigned integer. Indicates the open or close state of the IME.

ImeConvMode (4 bytes): A 32-bit unsigned integer. Indicates the IME conversion status.

2.2.9 Basic Output

2.2.9.1 Output PDU Packaging

2.2.9.1.1 Slow-Path (T.128) Format

2.2.9.1.1.1 Share Headers

The Share Headers used in conjunction with slow-path output PDUs are the same as those used in conjunction with slow-path input PDUs. These headers are described in section 2.2.8.1.1.1.

2.2.9.1.1.2 Security Headers

The Security Headers used in conjunction with slow-path output PDUs are the same as those used in conjunction with slow-path input PDUs. These headers are described in section 2.2.8.1.1.2.

2.2.9.1.1.3 Server Graphics Update PDU (TS_GRAPHICS_PDU)

The slow-path Graphics Update PDU is used to transmit graphics updates from server to client.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">slowPathGraphicsUpdate (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Shutdown Request PDU Data (section 2.2.2.2.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_UPDATE (2).

slowPathGraphicsUpdate (variable): Slow-path graphics update to be processed by the client (see section 2.2.9.1.1.3.1).

2.2.9.1.1.3.1 Slow Path Graphics Update (TS_GRAPHICS_UPDATE)

The TS_GRAPHICS_UPDATE structure is used to describe the type and encapsulate the data for a slow-path graphics update sent from server to client. All slow-path graphic updates conform to this basic structure (see sections 0 to 2.2.9.1.1.3.1.3).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">updateType</td><td colspan="16">updateData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

updateType (2 bytes): A 16-bit unsigned integer. Type of the graphics update.

Value	Meaning
UPDATETYPE_ORDERS 0x0000	Indicates an Orders Update (see [MS-RDPEGDI] section 2.2.2.2).
UPDATETYPE_BITMAP 0x0001	Indicates a Bitmap Graphics Update (see section 2.2.9.1.1.3.1.2).
UPDATETYPE_PALETTE 0x0002	Indicates a Palette Update (see section 2.2.9.1.1.3.1.1).
UPDATETYPE_SYNCHRONIZE 0x0003	Indicates a Synchronize Update (see section 2.2.9.1.1.3.1.3).

updateData (variable): Variable length data specific to the graphics update.

2.2.9.1.1.3.1.1 Palette Update (TS_UPDATE_PALETTE_PDU_DATA)

The TS_UPDATE_PALETTE_PDU_DATA structure contains global palette information that covers the entire session's palette (see [T128] section 8.18.6).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">updateType</td><td colspan="16">pad2Octets</td></tr>
<tr><td colspan="32">numberColors</td></tr>
<tr><td colspan="32">paletteData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

updateType (2 bytes): A 16-bit unsigned integer. The graphics update type. This field MUST be set to UPDATETYPE_PALETTE (0x0002).

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

numberColors (4 bytes): A 32-bit unsigned integer. The number of RGB triplets in the **paletteData** field. This field MUST be set to NUM_8BPP_PAL_ENTRIES (256).

paletteData (variable): Array of TS_PALETTE_ENTRY structures. Array of palette entries in RGB triplet format (see section 2.2.9.1.1.3.1.1.1) packed on byte boundaries. The number of triplet entries is given by the **numberColors** field - there must be NUM_8BPP_PAL_ENTRIES (256) entries.

2.2.9.1.1.3.1.1.1 RGB Palette Entry (TS_PALETTE_ENTRY)

The TS_PALETTE_ENTRY structure is used to express the red, green and blue components necessary to reproduce a color in the additive RGB space.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">red</td><td colspan="8">green</td><td colspan="8">blue</td><td colspan="8"></td></tr>
</table>

red (1 byte): An 8-bit unsigned integer. The red RGB color component.

green (1 byte): An 8-bit unsigned integer. The green RGB color component.

blue (1 byte): An 8-bit unsigned integer. The blue RGB color component.

2.2.9.1.1.3.1.2 Bitmap Update (TS_UPDATE_BITMAP_PDU_DATA)

The TS_UPDATE_BITMAP_PDU_DATA structure contains one or more rectangular clippings taken from the server-side screen frame buffer (see [T128] section 8.17).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">updateType</td><td colspan="16">numberRectangles</td></tr>
<tr><td colspan="32">rectangles (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

updateType (2 bytes): A 16-bit unsigned integer. The graphics update type. This field MUST be set to UPDATETYPE_BITMAP (0x0001).

numberRectangles (2 bytes): A 16-bit unsigned integer. The number of screen rectangles present in the rectangles field.

rectangles (variable): Variable length array of TS_BITMAP_DATA (section 2.2.9.1.1.3.1.2.3) structures, each of which contains a rectangular clipping taken from the server-side screen frame buffer. The number of screen clippings in the array is specified by the **numberRectangles** field.

2.2.9.1.1.3.1.2.1 RLE Compressed Bitmap Stream (RLE_BITMAP_STREAM)

The RLE_BITMAP_STREAM structure contains a stream of bitmap data compressed using Interleaved Run-Length Encoding (RLE). Compressed bitmap data MUST follow a Compressed Data Header (section 2.2.9.1.1.3.1.2.2) structure unless exclusion of this header has been negotiated in the General Capability Set (section 2.2.7.1.1).

A compressed bitmap is sent as a series of compression codes and color codes that instruct the decoder how to assemble the bitmap. A particular bitmap may have many valid compressed representations. A compression code consists of a code identifier, followed by an optional **length** field, followed by optional associated data (that is dependent on the compression code). Some codes instruct the decoder to refer to the previous row of bitmap data and because of this fact, the first row sometimes requires special cases for decoding.

The compression codes fill a full single byte of address space. The high order bits are used to identify the code type. The low order bits encode the length of the associated run. There are two forms of order:

- Regular orders with a 3-bit **code** field and a 5-bit **length** field.
- Lite orders with a 4-bit **code** field and a 4-bit **length** field.

A value of 0 in the 4 or 5-bit **length** field indicates an extended length (a MEGA run), where the following byte contains the length of the data. For MEGA runs the encoded length is the length of the run minus the maximum length of the non-MEGA form (unless otherwise specified).

The codespace also contains special case variants of the two main forms:

- The MEGA_MEGA form indicates an extended length, where the following two bytes contain the length of the data. In the MEGA_MEGA form the length encoded is the plain 16-bit length.

- A set of codes at the high end of the codespace is used to encode commonly occurring short sequences. These sequences are single byte FGBG encodings and single bytes of BLACK or WHITE.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
rleCompressedBitmapStream (variable)																															
...																															

rleCompressedBitmapStream (variable): An array of compression codes describing compressed structures in the bitmap.

3-Bit Codes (Regular Orders)	Meaning
MEGA_BG_RUN 000	A run where each byte matches the uncompressed byte from the previous line. The length of the run in bytes is 32 plus the value contained in the next byte between 0 and 255.
BG_RUN 000	Represents a background run of the specified length. The background color is black by default.
MEGA_FG_RUN 001	A run where each byte is the XOR of the uncompressed byte from the previous line with the foreground color. The length of the run in bytes is 32 plus the value contained in the next byte between 0 and 255.
FG_RUN 001	Represents a continuous foreground run of the specified length. The foreground color is white by default.
MEGA_FG_BG_IMAGE 010	Represents a binary image containing only the current foreground and background colors. The **length** field is the actual number of pixels. XOR is performed.
FG_BG_IMAGE 010	Represents a binary image containing only the current foreground and background colors. The length of a short run is encoded as (**length** / 8). XOR is performed.
MEGA_COLOR_RUN 011	A single-color run. The length of the run in bytes is 32 plus the value contained in the next byte between 0 and 255. The color is specified in the following byte.
COLOR_RUN 011	A single-color run. The length of the run in bytes is the other 5 bits of the byte. The color is specified in the following byte.
MEGA_COLOR_IMAGE 100	An uncompressed run. The length of the run in bytes is 32 plus the value contained in the next byte between 0 and 255. The data is specified in the following bytes as one pixel per byte.
COLOR_IMAGE 100	Represents a color image of the specified length. No XOR is performed. This data is uncompressed.
NOT_USED_RESERVED 101	Not used.

4-Bit Codes (Lite Orders)	Meaning
SET_FG_MEGA_FG_RUN 1100	A run where each byte is the XOR of the uncompressed byte from the previous line with a new foreground color. The length of the run in bytes is 16 plus the value contained in the next byte between 0 and 255. The new foreground color is specified in the following byte.
SET_FG_FG_RUN 1100	Represents a continuous foreground run of the specified length. The foreground color is white by default, and is changed by this code.
SET_FG_MEGA_FG_BG 1101	A run where each byte is either the matching uncompressed byte from the previous line or the XOR of that byte with the foreground color. The length of the run in bytes is 8 multiplied by the value of the other 4 bits of the byte, between 0 and 15. The new foreground color is specified in the next byte. The data is specified in the following bytes.
SET_FG_FG_BG 1101	A run where each byte is either the matching uncompressed byte from the previous line or the XOR of that byte with the foreground color. The length of the run in bytes is 8 multiplied by the value of the other 4 bits of the byte, between 0 and 15. The new foreground color is specified in the next byte. The data is specified in the following bytes.
MEGA_DITHERED_RUN 1110	An alternating run of two colors. The length of the run in bytes is 16 plus the value contained in the next byte, between 0 and 255. The colors are specified in the following two bytes using one byte each.
DITHERED_RUN 1110	Represents a run of alternating colors of the specified colors and length, where the length is in pixel pairs. The colors are specified in the following two bytes using one byte each. No XOR is performed.

8-Bit Codes	Meaning
MEGA_MEGA_BG_RUN 11110000	A run where each byte matches the uncompressed byte from the previous line. The length of the run in bytes is specified in the next two bytes between 1 and 65,536.
MEGA_MEGA_FG_RUN 11110001	A run where each byte is the XOR of the uncompressed byte from the previous line with the foreground color. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. If this code occurs on the first line the foreground color alone should be used.
MEGA_MEGA_FGBG 11110010	A long run where each byte is either the uncompressed byte from the previous line or the XOR of that byte with the foreground color. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The data is specified in the following bytes.
MEGA_MEGA_COLOR_RUN 11110011	A long single-color run of pixels. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The color is specified in the following byte.
MEGA_MEGA_CLR_IMG 11110100	A long uncompressed run of pixels. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The data is specified in the following bytes as one pixel per byte.

8-Bit Codes	Meaning
NOT_USED_RESERVED 11110101	Not used.
MEGA_MEGA_SET_FG_RUN 11110110	A long run where each byte is the XOR of the uncompressed byte from the previous line with a new foreground color. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The new foreground color is specified in the following byte.
MEGA_MEGA_SET_FGBG 11110111	A long run where each byte is either the uncompressed byte from the previous line or the XOR of that byte with a new foreground color. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The new foreground color is specified in the byte after the length. The data is specified in the following bytes.
MEGA_MEGA_DITHER 11111000	A long alternating run of two colors. The length of the run in bytes is specified in the next two bytes between 1 and 65,536. The colors are specified in the following two bytes as one byte each.
SPECIAL_FGBG_CODE_1 11111001	This code should be treated as an FG_BG_IMAGE run with predefined parameters. This is always 8 pixels in length and has a data pattern of FGBG code 0x03 = binary: 11000000.
SPECIAL_FGBG_CODE_2 11111010	This code should be treated as an FG_BG_IMAGE run with predefined parameters. This is always 8 pixels in length and has a data pattern of FGBG code 0x05 = binary: 10100000.
SPECIAL_FGBG_CODE_3 11111011	This code should be treated as an FG_BG_IMAGE run with predefined parameters. This is always 8 pixels in length and has a data pattern of FGBG code 0x07 = binary: 11100000.
SPECIAL_FGBG_CODE_4 11111100	This code should be treated as an FG_BG_IMAGE run with predefined parameters. This is always 8 pixels in length and has a data pattern of FGBG code 0x0F = binary: 11110000.
WHITE 11111110	A single white pixel.
BLACK 11111101	A single black pixel.
START_LOSSY 11111111	Informs the decoder that lossy mode has been established and any of the following color runs will need pixel doubling performed. RLE decoding will remain in this mode until the end of this block.

2.2.9.1.1.3.1.2.2 Compressed Data Header (TS_CD_HEADER)

The TS_CD_HEADER structure is used to describe compressed bitmap data.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">cbCompFirstRowSize</td><td colspan="16">cbCompMainBodySize</td></tr>
<tr><td colspan="16">cbScanWidth</td><td colspan="16">cbUncompressedSize</td></tr>
</table>

cbCompFirstRowSize (2 bytes): A 16-bit unsigned integer. The field MUST be set to 0x0000.

cbCompMainBodySize (2 bytes): A 16-bit unsigned integer. The size in bytes of the compressed bitmap data (which follows this header).

cbScanWidth (2 bytes): A 16-bit unsigned integer. The width of the bitmap (which follows this header) in pixels (this value MUST be divisible by 4).

cbUncompressedSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the bitmap data (which follows this header) after it has been decompressed.

2.2.9.1.1.3.1.2.3 Bitmap Data (TS_BITMAP_DATA)

The TS_BITMAP_DATA structure wraps the bitmap data bytestream for a screen area rectangle containing a clipping taken from the server-side screen frame buffer.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">destLeft</td><td colspan="16">destTop</td></tr>
<tr><td colspan="16">destRight</td><td colspan="16">destBottom</td></tr>
<tr><td colspan="16">width</td><td colspan="16">height</td></tr>
<tr><td colspan="16">bitsPerPixel</td><td colspan="16">Flags</td></tr>
<tr><td colspan="16">bitmapLength</td><td colspan="16">bitmapComprHdr (optional)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">bitmapDataStream (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

destLeft (2 bytes): A 16-bit unsigned integer. Left bound of the rectangle.

destTop (2 bytes): A 16-bit unsigned integer. Top bound of the rectangle.

destRight (2 bytes): A 16-bit unsigned integer. Right bound of the rectangle.

destBottom (2 bytes): A 16-bit unsigned integer. Bottom bound of the rectangle.

width (2 bytes): A 16-bit unsigned integer. The width of the rectangle.

height (2 bytes): A 16-bit unsigned integer. The height of the rectangle.

bitsPerPixel (2 bytes): A 16-bit unsigned integer. The color depth of the rectangle data in bits-per-pixel.

Flags (2 bytes): A 16-bit unsigned integer. The flags describing the format of the bitmap data in the **bitmapDataStream** field.

Flags	Meaning
BITMAP_COMPRESSION 0x0001	Indicates that the bitmap data is compressed. This implies that the **bitmapComprHdr** field is present if the NO_BITMAP_COMPRESSION_HDR (0x0400) flag is not set.
NO_BITMAP_COMPRESSION_HDR 0x0400	Indicates that the **bitmapComprHdr** field is not present (removed for bandwidth efficiency to save 8 bytes).

bitmapLength (2 bytes): A 16-bit unsigned integer. The size in bytes of the data in the **bitmapComprHdr** and **bitmapDataStream** fields.

bitmapComprHdr (8 bytes): Optional Compressed Data Header structure (see Compressed Data Header (TS_CD_HEADER) (section 2.2.9.1.1.3.1.2.2)) specifying the bitmap data in the **bitmapDataStream**. This field MUST be present if the BITMAP_COMPRESSION (0x0001) flag is present in the **Flags** field, but the NO_BITMAP_COMPRESSION_HDR (0x0400) flag is not.

bitmapDataStream (variable): A variable-sized array of bytes. Uncompressed bitmap data represents a bitmap as a bottom-up, left-to-right series of pixels. Each pixel is a whole number of bytes. Each row contains a multiple of four bytes (including up to three bytes of padding, as necessary). Compressed bitmaps not in 32 bits-per-pixel format are compressed using Interleaved Run-Length Encoding (RLE) and encapsulated in an (see section 2.2.9.1.1.3.1.2.1) while compressed bitmap data at a color depth of 32 bits-per-pixel is compressed using RDP 6.0 Bitmap Compression and stored inside an RDP 6.0 Bitmap Compressed Stream structure (see section 2.2.9.1.1.3.1.2.1).

2.2.9.1.1.3.1.3 Synchronize Update (TS_UPDATE_SYNC_PDU_DATA)

The TS_UPDATE_SYNC_PDU_DATA structure is an artifact of the T.128 protocol (see [T128] section 8.6.2) and is ignored by current Microsoft RDP client implementations.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">updateType</td><td colspan="16">pad2Octets</td></tr>
</table>

updateType (2 bytes): A 16-bit unsigned integer. Graphics update type. This field MUST be set to UPDATETYPE_SYNCHRONIZE (0x0003).

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

2.2.9.1.1.4 Server Pointer Update PDU (TS_POINTER_PDU)

The Pointer Update PDU is sent from server to client and is used to convey pointer information, including pointers' bitmap images, use of system or hidden pointers, use of cached cursors and position updates.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">messageType</td></tr>
<tr><td colspan="16">pad2Octets</td><td colspan="16">pointerAttributeData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The userData field of the MCS Send Data Indication PDU contains a Security Header and the Pointer Update PDU data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

shareDataHeader (18 bytes): Share Data Header (section 2.2.8.1.1.1.2) containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_POINTER (27).

messageType (2 bytes): A 16-bit unsigned integer. Type of pointer update.

Value	Meaning
TS_PTRMSGTYPE_SYSTEM 0x0001	Indicates a System Pointer Update (section 2.2.9.1.1.4.3).
TS_PTRMSGTYPE_POSITION 0x0003	Indicates a Pointer Position Update (section 2.2.9.1.1.4.2).
TS_PTRMSGTYPE_COLOR 0x0006	Indicates a Color Pointer Update (section 2.2.9.1.1.4.4).
TS_PTRMSGTYPE_CACHED 0x0007	Indicates a Cached Pointer Update (section 2.2.9.1.1.4.6).
TS_PTRMSGTYPE_POINTER 0x0008	Indicates a New Pointer Update (section 2.2.9.1.1.4.5).

T.128 Monochrome Pointer updates (see [T128] section 8.14.2) are not used in RDP and are not planned for a future version. Monochrome pointers are translated into 24 bits-per-pixel cursors using the Color Pointer Update (section 2.2.9.1.1.4.4) when the New Pointer Update (section 2.2.9.1.1.4.5) is not supported, or sent as 1 bit-per-pixel using the New Pointer Update.

pad2Octets (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

pointerAttributeData (variable): TS_POINTERPOSATTRIBUTE (4 bytes), TS_SYSTEMPOINTERATTRIBUTE (4 bytes), TS_COLORPOINTERATTRIBUTE (variable number of bytes), TS_POINTERATTRIBUTE (variable number of bytes) or TS_CACHEDPOINTERATTRIBUTE (2 bytes):

The actual contents of the slow-path pointer update (see sections 2.2.9.1.1.4.2 to 2.2.9.1.1.4.6).

2.2.9.1.1.4.1 Point (TS_POINT16)

The TS_POINT16 structure specifies a point relative to the top-left corner of the server's virtual desktop.

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
xPos																yPos															

xPos (2 bytes): A 16-bit unsigned integer. The X coordinate relative to the top-left corner of the server's virtual desktop.

yPos (2 bytes): A 16-bit unsigned integer. The Y coordinate relative to the top-left corner of the server's virtual desktop.

2.2.9.1.1.4.2 Pointer Position Update (TS_POINTERPOSATTRIBUTE)

The TS_POINTERPOSATTRIBUTE structure is used to indicate that the client pointer should be moved to the specified position relative to the top-left corner of the server's virtual desktop (see [T128] section 8.14.4).

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
position																															

position (4 bytes): Point (section 2.2.9.1.1.4.1) structure containing the new X and Y coordinates of the pointer.

2.2.9.1.1.4.3 System Pointer Update (TS_SYSTEMPOINTERATTRIBUTE)

The TS_SYSTEMPOINTERATTRIBUTE structure is used to hide the pointer or to set its shape to that of the operating system default (see [T128] section 8.14.1).

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
systemPointerType																															

systemPointerType (4 bytes): A 32-bit unsigned integer. The type of system pointer.

Value	Meaning
SYSPTR_NULL 0x00000000	The hidden pointer.
SYSPTR_DEFAULT 0x00007F00	The default system pointer.

2.2.9.1.1.4.4 Color Pointer Update (TS_COLORPOINTERATTRIBUTE)

The TS_COLORPOINTERATTRIBUTE structure represents a regular T.128 24 bits-per-pixel color pointer, as specified in [T128] section 8.14.3. This pointer update is used for both monochrome and color pointers in RDP.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
cacheIndex																hotSpot															
...																width															
height																lengthAndMask															
lengthXorMask																xorMaskData (variable)															
...																															
andMaskData (variable)																															
...																															

cacheIndex (2 bytes): A 16-bit unsigned integer. The zero-based cache entry in the pointer cache in which to store the pointer image. The number of cache entries is negotiated using the Pointer Capability Set (section 2.2.7.1.6).

hotSpot (4 bytes): Point (section 2.2.9.1.1.4.1) structure containing the X and Y coordinates of the pointer hotspot.

width (2 bytes): A 16-bit unsigned integer. The width of the pointer in pixels (the maximum allowed pointer width is 32 pixels).

height (2 bytes): A 16-bit unsigned integer. The height of the pointer in pixels (the maximum allowed pointer height is 32 pixels).

lengthAndMask (2 bytes): A 16-bit unsigned integer. The size in bytes of the **andMaskData** field.

lengthXorMask (2 bytes): A 16-bit unsigned integer. The size in bytes of the **xorMaskData** field.

xorMaskData (variable): Variable number of bytes: Contains the 24 bits-per-pixel bottom-up XOR mask scan-line data. The XOR mask is padded to a 2-byte boundary for each encoded scan-line. For example, if a 3x3 pixel cursor is being sent, then each scan-line will consume 10 bytes (3 pixels per scan-line multiplied by 3 bytes per pixel, rounded up to the next even number of bytes).

andMaskData (variable): Variable number of bytes: Contains the 1 bit-per-pixel bottom-up AND mask scan-line data. The AND mask is padded to a 2-byte boundary for each encoded scan-line. For example, if a 7x7 pixel cursor is being sent, then each scan-line will consume 2 bytes (7 pixels per scan-line multiplied by 1 bit per pixel, rounded up to the next even number of bytes).

2.2.9.1.1.4.5 New Pointer Update (TS_POINTERATTRIBUTE)

The TS_POINTERATTRIBUTE structure is used to send pointer data at an arbitrary color depth. Support for the New Pointer Update is advertised in the Pointer Capability Set (section 2.2.7.1.6).

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">xorBpp</td><td colspan="16">colorPtrAttr (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

xorBpp (2 bytes): A 16-bit unsigned integer. The color depth in bits-per-pixel of the XOR mask contained in the **colorPtrAttr** field.

colorPtrAttr (variable): Encapsulated Color Pointer Update (section 2.2.9.1.1.4.4) structure which contains information about the pointer. The Color Pointer Update fields are all used, as specified in section 2.2.9.1.1.4.4; however, the XOR mask data alignment packing is slightly different. For monochrome (1 bit-per-pixel) pointers the XOR data is always padded to a 4-byte boundary per scan line, while color pointer XOR data is still packed on a 2-byte boundary. Color XOR data is presented in the color depth described in the xorBpp field (for 8 bits-per-pixel, each byte contains one palette index; for 4 bits-per-pixel there are two palette indices per byte).

2.2.9.1.1.4.6 Cached Pointer Update (TS_CACHEDPOINTERATTRIBUTE)

The TS_CACHEDPOINTERATTRIBUTE structure is used to instruct the client to change the current pointer shape to one already present in the pointer cache.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">cacheIndex</td><td colspan="16"></td></tr>
</table>

cacheIndex (2 bytes): A 16-bit unsigned integer. A zero-based cache entry containing the cache index of the cached pointer to which the client's pointer should be changed. The pointer data should have already been cached using either the Color Pointer Update (section 2.2.9.1.1.4.4) or New Pointer Update (section 2.2.9.1.1.4.5).

2.2.9.1.1.5 Server Play Sound PDU

The Play Sound PDU instructs the client to play a "beep" sound.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">playSoundPduData</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Play Sound PDU Data (section 2.2.9.1.1.5.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- [FIPS Security Header (section 2.2.8.1.1.2.3)](#) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

playSoundPduData (26 bytes): The actual contents of the Play Sound PDU, as specified in section 2.2.9.1.1.5.1.

2.2.9.1.1.5.1 Play Sound PDU Data (TS_PLAY_SOUND_PDU_DATA)

The TS_PLAY_SOUND_PDU_DATA structure contains the contents of the Play Sound PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) and two fields.

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
shareDataHeader																															
...																															
...																															
...																															
...																duration															
...																frequency															
...																															

shareDataHeader (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_PLAY_SOUND (34).

duration (4 bytes): A 32-bit unsigned integer. Duration of the beep the client should play.

frequency (4 bytes): A 32-bit unsigned integer. Frequency of the beep the client should play.

2.2.9.1.2 Server Fast-Path Update PDU (TS_FP_UPDATE_PDU)

Fast-path revises server output packets from the first byte with the goal of improving bandwidth. The TPKT (see [T123]), X.224 (see [X224]) and MCS SDin (see [T125]) headers are replaced, the Security Header (section 2.2.8.1.1.2) is collapsed into the fast-path output header, and the Share Data Header (section 2.2.8.1.1.1.2) is replaced by a new fast-path format. The contents of the graphics and pointer updates (see sections 2.2.9.1.1.3 and 2.2.9.1.1.4) are also changed to reduce their size, particularly by removing or reducing headers. Support for fast-path output is advertised in the General Capability Set (section 2.2.7.1.1).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8">fpOutputHeader</td><td colspan="8">length1</td><td colspan="8">length2 (optional)</td><td colspan="8">fipsInformation (optional)</td></tr>
<tr><td colspan="24">...</td><td colspan="8">dataSignature (optional)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="24">...</td><td colspan="8">fpOutputUpdates (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

fpOutputHeader (1 byte): An 8-bit unsigned integer. One-byte bit-packed header. This byte coincides with the first byte of the TPKT Header (see [T123] section 8), which is always 0x03. Two pieces of information are collapsed into this byte:

1. Encryption data

2. Action code

The format of the **fpOutputHeader** byte is described by the following bitmask diagram:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="2">actionCode</td><td colspan="4">reserved</td><td colspan="2">encryptionFlags</td><td colspan="24"></td></tr>
</table>

actionCode (2 bits): 2 bits: Code indicating whether the PDU is in fast-path or slow-path format.

Value	Meaning
FASTPATH_OUTPUT_ACTION_FASTPATH 0x0 (binary: 00)	Indicates that the PDU is a fast-path output PDU.
FASTPATH_OUTPUT_ACTION_X224 0x3 (binary: 11)	Indicates the presence of a TPKT Header (see [T123] section 8) initial version byte, which implies that the PDU is a slow-path output PDU (in this case the full value of the initial byte MUST be 0x03).

reserved (4 bits): 4 bits: unused bits reserved for future use. This bitfield MUST be set to 0.

encryptionFlags (2 bits): 2 bits: flags describing cryptographic parameters of the PDU.

Value	Meaning
FASTPATH_OUTPUT_SECURE_CHECKSUM 0x1 (binary: 01)	Indicates that the MAC signature for the PDU was generated using the "salted MAC generation" technique (see section 5.3.6.1.1). If this bit is not set, then the standard technique was used (see Non-FIPS (section 2.2.8.1.1.2.2) and FIPS (section 2.2.8.1.1.2.3)).
FASTPATH_OUTPUT_ENCRYPTED 0x2 (binary: 10)	Indicates that the PDU contains an 8-byte message authentication code (MAC) signature after the optional **length2** field (that is, the **dataSignature** field is present) and the contents of the PDU are encrypted using the negotiated encryption package (see sections 5.3.2 and 5.3.6).

length1 (1 byte): An 8-bit unsigned integer. If the most significant bit of the **length1** field is not set, then the size of the PDU is in the range 1 to 127 bytes and the **length1** field contains the overall PDU length (the **length2** field is not present in this case). However, if the most significant bit of the **length1** field is set, then the overall PDU length is given by the low 7 bits of the **length1** field concatenated with the 8 bits of the **length2** field, in big-endian order (the **length2** field contains the low-order bits).

length2 (1 byte): An 8-bit unsigned integer. If the most significant bit of the **length1** field is not set, then the **length2** field is not present. If the most significant bit of the **length1** field is set, then the overall PDU length is given by the low 7 bits of the **length1** field concatenated with the 8 bits of the **length2** field, in big-endian order (the **length2** field contains the low-order bits).

fipsInformation (4 bytes): Optional FIPS header information, present when the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4). The fast-path FIPS information structure is specified in section 2.2.8.1.2.1.

dataSignature (8 bytes): 8 bytes. Message authentication code (MAC) generated over the packet using one of the techniques specified in Non-FIPS (the FASTPATH_INPUT_SECURE_CHECKSUM flag, which is set in the fpInputHeader field, describes the method used to generate the signature). This field is present if the FASTPATH_INPUT_ENCRYPTED flag is set in the **fpInputHeader** field.

fpOutputUpdates (variable): An array of TS_FP_UPDATE structures (variable number of bytes) containing a collection of fast-path updates to be processed by the client.

2.2.9.1.2.1 Fast-Path Update (TS_FP_UPDATE)

The TS_FP_UPDATE structure is used to describe and encapsulate the data for a fast-path update sent from server to client. All fast-path updates conform to this basic structure (see sections 2.2.9.1.2.1.1 to 2.2.9.1.2.1.9).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">updateHeader</td><td colspan="8">compressionFlags (optional)</td><td colspan="16">size</td></tr>
<tr><td colspan="32">updateData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

updateHeader (1 byte): An 8-bit unsigned integer. The TS_FP_UPDATE structure begins with a one- byte bit-packed update **header** field. Two pieces of information are collapsed into this byte:

1. Fast-path update type

2. Compression usage indication

The format of the **updateHeader** byte is described by the following bitmask diagram:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="4">updateCode</td><td colspan="2">reserved</td><td colspan="2">compression</td></tr>
</table>

updateCode (4 bits): 4 bits: Type code of the update.

Value	Meaning
FASTPATH_UPDATETYPE_ORDERS 0x0 (binary: 0000)	Indicates a Fast-Path Orders Update (see [MS-RDPEGDI] section 2.2.2.3).
FASTPATH_UPDATETYPE_BITMAP 0x1 (binary: 0001)	Indicates a Fast-Path Bitmap Update (see section 2.2.9.1.2.1.2).
FASTPATH_UPDATETYPE_PALETTE 0x2 (binary: 0010)	Indicates a Fast-Path Palette Update (see section 2.2.9.1.2.1.1).
FASTPATH_UPDATETYPE_SYNCHRONIZE 0x3 (binary: 0011)	Indicates a Fast-Path Synchronize Update (see section 2.2.9.1.2.1.3).
FASTPATH_UPDATETYPE_PTR_NULL 0x5 (binary: 0101)	Indicates a Fast-Path System Pointer Hidden Update (see section 2.2.9.1.2.1.5).
FASTPATH_UPDATETYPE_PTR_DEFAULT 0x6 (binary: 0110)	Indicates a Fast-Path System Pointer Default Update (see section 2.2.9.1.2.1.6).
FASTPATH_UPDATETYPE_PTR_POSITION 0x8 (binary: 1000)	Indicates a Fast-Path Pointer Position Update (see section 2.2.9.1.2.1.4).
FASTPATH_UPDATETYPE_COLOR 0x9 (binary: 1001)	Indicates a Fast-Path Color Pointer Update (see section 2.2.9.1.2.1.7).

Value	Meaning
FASTPATH_UPDATETYPE_CACHED 0xA (binary: 1010)	Indicates a Fast-Path Cached Pointer Update (see section 2.2.9.1.2.1.9).
FASTPATH_UPDATETYPE_POINTER 0xB (binary: 1011)	Indicates a Fast-Path New Pointer Update (see section 2.2.9.1.2.1.8).

reserved (2 bits): 2 bits. Unused bits reserved for future use. This bitfield MUST be set to 0.

compression (2 bits): 2 bits. Compression usage indication flags.

Value	Meaning
FASTPATH_OUTPUT_COMPRESSION_USED 0x2 (binary: 10)	Indicates that the **compressionFlags** field is present.

compressionFlags (1 byte): An 8-bit unsigned integer. Optional compression flags. The flags used in this field are exactly the same as the MPPC flags used in the **compressedType** field in the Share Data Header (section 2.2.8.1.1.1.2) and have the same meaning.

size (2 bytes): A 16-bit unsigned integer. The size in bytes of the data in the **updateData** field.

updateData (variable): Optional and variable length data specific to the update.

2.2.9.1.2.1.1 Fast-Path Palette Update (TS_FP_UPDATE_PALETTE)

The TS_FP_UPDATE_PALETTE structure is the fast-path variant of the TS_UPDATE_PALETTE_PDU_DATA (section 2.2.9.1.1.3.1.1) structure.

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
updateHeader								compressionFlags (optional)								size															
paletteUpdateData (variable)																															
...																															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field, specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_PALETTE (2).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

paletteUpdateData (variable): Variable length palette data. Both slow and fast-path utilize the same data format, a Palette Update structure to represent this information.

2.2.9.1.2.1.2 Fast-Path Bitmap Update (TS_FP_UPDATE_BITMAP)

The TS_FP_UPDATE_BITMAP structure is the fast-path variant of the TS_UPDATE_BITMAP_PDU_DATA (section 2.2.9.1.1.3.1.2) structure.

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
updateHeader								compressionFlags (optional)								size															
bitmapUpdateData (variable)																															
...																															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_BITMAP (1).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

bitmapUpdateData (variable): Variable length bitmap data. Both slow and fast-path utilize the same data format, a Bitmap Update structure to represent this information.

2.2.9.1.2.1.3 Fast-Path Synchronize Update (TS_FP_UPDATE_SYNCHRONIZE)

The TS_FP_UPDATE_SYNCHRONIZE structure is the fast-path variant of the TS_UPDATE_SYNCHRONIZE_PDU_DATA (section 2.2.9.1.1.3.1.3) structure.

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
updateHeader								compressionFlags (optional)								size															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field described in the Fast-Path Update (section 2.2.9.1.2.1). The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_SYNCHRONIZE (3).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field described in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. This field MUST be set to 0.

2.2.9.1.2.1.4 Fast-Path Pointer Position Update (TS_FP_POINTERPOSATTRIBUTE)

The TS_FP_POINTERPOSATTRIBUTE structure is the fast-path variant of the TS_POINTERPOSATTRIBUTE structure (see section 2.2.9.1.1.4.2).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
updateHeader								compressionFlags (optional)								size															
pointerPositionUpdateData																															

updateHeader (1 byte): The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_PTR_POSITION (8).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

pointerPositionUpdateData (4 bytes): TS_POINTERPOSATTRIBUTE structure (4 bytes): Pointer coordinates. Both slow and fast-path utilize the same data format, a Pointer Position Update structure to represent this information.

2.2.9.1.2.1.5 Fast-Path System Pointer Hidden Update (TS_FP_SYSTEMPOINTERHIDDENATTRIBUTE)

The TS_FP_SYSTEMPOINTERHIDDENATTRIBUTE structure is the fast-path variant of the TS_SYSTEMPOINTERATTRIBUTE (section 2.2.9.1.1.4.3) structure which contains the SYSPTR_NULL (0x00000000) flag.

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
updateHeader								compressionFlags (optional)								size															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_PTR_NULL (5).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. This field MUST be set to 0.

2.2.9.1.2.1.6 Fast-Path System Pointer Default Update (TS_FP_SYSTEMPOINTERDEFAULTATTRIBUTE)

The TS_FP_SYSTEMPOINTERDEFAULTATTRIBUTE structure is the fast-path variant of the TS_SYSTEMPOINTERATTRIBUTE (section 2.2.9.1.1.4.3) structure which contains the SYSPTR_DEFAULT (0x00007F00) flag.

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
updateHeader								compressionFlags (optional)								size															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_PTR_DEFAULT (6).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. This field MUST be set to 0.

2.2.9.1.2.1.7 Fast-Path Color Pointer Update (TS_FP_COLORPOINTERATTRIBUTE)

The TS_FP_COLORPOINTERATTRIBUTE structure is the fast-path variant of the TS_COLORPOINTERATTRIBUTE (section 2.2.9.1.1.4.4) structure.

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
updateHeader								compressionFlags (optional)								size															
colorPointerUpdateData (variable)																															
...																															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_COLOR (9).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

colorPointerUpdateData (variable): Color pointer data. Both slow and fast-path utilize the same data format, a Color Pointer Update structure to represent this information.

2.2.9.1.2.1.8 Fast-Path New Pointer Update (TS_FP_POINTERATTRIBUTE)

The TS_FP_POINTERATTRIBUTE structure is the fast-path variant of the TS_POINTERATTRIBUTE (section 2.2.9.1.1.4.5) structure.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
updateHeader								compressionFlags (optional)								size															
newPointerUpdateData (variable)																															
...																															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure. The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_POINTER (11).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

newPointerUpdateData (variable): Color pointer data at arbitrary color depth. Both slow and fast-path utilize the same data format, a New Pointer Update structure to represent this information.

2.2.9.1.2.1.9 Fast-Path Cached Pointer Update (TS_FP_CACHEDPOINTERATTRIBUTE)

The TS_FP_CACHEDPOINTERATTRIBUTE structure is the fast-path variant of the TS_CACHEDPOINTERATTRIBUTE (section 2.2.9.1.1.4.6) structure.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
updateHeader								compressionFlags (optional)								size															
cachedPointerUpdateData																															

updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field specified in the Fast-Path Update (section 2.2.9.1.2.1) structure.

The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_CACHED (10).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field specified in the Fast-Path Update structure.

size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field specified in the Fast-Path Update structure.

cachedPointerUpdateData (2 bytes): Cached pointer data. Both slow and fast-path utilize the same data format (a Cached Pointer Update structure) to represent this information.

2.2.10 Logon Notifications

2.2.10.1 Server Save Session Info PDU

The Save Session Info PDU is used by the server to transmit session and user logon information back to the client after the user has logged on.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDin (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">saveSessionInfoPduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable length PER-encoded MCS Send Data Indication PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Indication PDU contains a Security Header and the Save Session Info PDU Data (section 2.2.10.1.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Basic Security Header (section 2.2.8.1.1.2.1) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1).
- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

saveSessionInfoPduData (variable): The actual contents of the Save Session Info PDU, as specified in section 2.2.10.1.1.

2.2.10.1.1 Save Session Info PDU Data (TS_SAVE_SESSION_INFO_PDU_DATA)

The TS_SAVE_SESSION_INFO_PDU_DATA structure is a wrapper around different classes of user logon information.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="16">infoType</td></tr>
<tr><td colspan="16">...</td><td colspan="16">infoData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

shareDataHeader (18 bytes): TS_SHAREDATAHEADER structure (18 bytes): Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SAVE_SESSION_INFO (38).

infoType (4 bytes): A 32-bit unsigned integer. The type of logon information.

Value	Meaning
INFOTYPE_LOGON	This is a notification that the user has logged on. The

Value	Meaning
0x00000000	**infoData** field which follows contains a Logon Info Version 1 (section 2.2.10.1.1.1) structure.
INFOTYPE_LOGON_LONG 0x00000001	This is a notification that the user has logged on. The **infoData** field which follows contains a Logon Info Version 2 (section 2.2.10.1.1.2) structure. This type was added in RDP 5.1 and SHOULD be used if the LONG_CREDENTIALS_SUPPORTED (0x00000004) flag is set in the General Capability Set (section 2.2.7.1.1).
INFOTYPE_LOGON_PLAINNOTIFY 0x00000002	This is a notification that the user has logged on. The **infoData** field which follows contains a Plain Notify structure which contains 576 bytes of padding (see Section 2.2.10.1.1.3). This type was added in RDP 5.1.
INFOTYPE_LOGON_EXTENDED_INF 0x00000003	The **infoData** field which follows contains a Logon Info Extended (section 2.2.10.1.1.4) structure. This type was added in RDP 5.2.

infoData (variable): TS_LOGON_INFO, TS_LOGON_INFO_VERSION_2 or TS_LOGON_INFO_EXTENDED: Variable length logon information structure. The type of data which follows depends on the value of the **infoType** field.

2.2.10.1.1.1 Logon Info Version 1 (TS_LOGON_INFO)

TS_LOGON_INFO is a fixed-length structure which contains logon information intended for the client.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">cbDomain</td></tr>
<tr><td colspan="32">Domain</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

(Domain cont'd for 5 rows)
cbUserName
UserName
...
...
...
...
...
...
...
(UserName cont'd for 120 rows)
SessionId

cbDomain (4 bytes): A 32-bit unsigned integer. The size of the Unicode character data (including the mandatory null terminator), in bytes, present in the fixed-length **Domain** field.

Domain (52 bytes): An array of 26 Unicode characters: Null-terminated Unicode string containing the name of the domain to which the user is logged on. The length of the character data in bytes is given by the **cbDomain** field.

cbUserName (4 bytes): A 32-bit unsigned integer. Size of the Unicode character data (including the mandatory null terminator), in bytes, present in the fixed-length **UserName** field.

UserName (512 bytes): An array of 256 Unicode characters: Null-terminated Unicode string containing the username which was used to log on. The length of the character data in bytes is given by the **cbUserName** field.

SessionId (4 bytes): A 32-bit unsigned integer. Optional session ID of the session according to the server. Sent by RDP 5.0 and later servers.

2.2.10.1.1.2 Logon Info Version 2 (TS_LOGON_INFO_VERSION_2)

TS_LOGON_INFO_VERSION_2 is a variable length structure which contains logon information intended for the client.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="16">Version</td><td colspan="16">Size</td></tr>
<tr><td colspan="16">...</td><td colspan="16">SessionId</td></tr>
<tr><td colspan="16">...</td><td colspan="16">cbDomain</td></tr>
<tr><td colspan="16">...</td><td colspan="16">cbUserName</td></tr>
<tr><td colspan="16">...</td><td colspan="16">Pad</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">(Pad cont'd for 132 rows)</td></tr>
<tr><td colspan="32">Domain (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">UserName (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

Version (2 bytes): A 16-bit unsigned integer. The logon version.

Value	Meaning
SAVE_SESSION_PDU_VERSION_ONE 0x0001	Version 1

Size (4 bytes): A 32-bit unsigned integer. The total size in bytes of this structure, excluding the **Domain** and **UserName** variable length fields.

SessionId (4 bytes): A 32-bit unsigned integer. The session ID of the session according to the server.

cbDomain (4 bytes): A 32-bit unsigned integer. Size, in bytes, of the **Domain** field (including the mandatory null terminator).

cbUserName (4 bytes): A 32-bit unsigned integer. The size, in bytes, of the **UserName** field (including the mandatory null terminator).

Pad (558 bytes): 558 bytes. Padding. Values in this field are ignored.

Domain (variable): Variable length null-terminated Unicode string containing the name of the domain to which the user is logged on. The size of this field in bytes is given by the **cbDomain** field.

UserName (variable): Variable length null-terminated Unicode string containing the user name which was used to log on. The size of this field in bytes is given by the **cbUserName** field.

2.2.10.1.1.3 Plain Notify (TS_PLAIN_NOTIFY)

The TS_PLAIN_NOTIFY is a fixed-length structure which contains 576 bytes of padding.

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
Pad																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(Pad cont'd for 136 rows)																															

Pad (576 bytes): 576 bytes. Padding. Values in this field are ignored.

2.2.10.1.1.4 Logon Info Extended (TS_LOGON_INFO_EXTENDED)

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">Length</td><td colspan="16">FieldsPresent</td></tr>
<tr><td colspan="16">...</td><td colspan="16">LogonFields (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Pad</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">(Pad cont'd for 134 rows)</td></tr>
<tr><td colspan="16">...</td></tr>
</table>

Length (2 bytes): A 16-bit unsigned integer. The total size in bytes of this structure, including the variable **LogonFields** field.

FieldsPresent (4 bytes): A 32-bit unsigned integer. The flags indicating which fields are present in the **LogonFields** field.

Value	Meaning
LOGON_EX_AUTORECONNECTCOOKIE 0x00000001	An auto-reconnect cookie field is present. The **LogonFields** field of the associated Logon Info (section 2.2.10.1.1.4.1) structure MUST contain a Server Auto-Reconnect (section 2.2.4.2) structure.
LOGON_EX_LOGONERRORS 0x00000002	A logon error field is present. The **LogonFields** field of the associated Logon Info MUST contain a Logon Errors Info

Value	Meaning
	(section 2.2.10.1.1.4.1.1) structure.

LogonFields (variable): Extended logon information fields encapsulated in Logon Info Field structures. The presence of an information field is indicated by the flags within the **FieldsPresent** field of the Logon Info Extended structure. The ordering of the fields is implicit and is as follows:

1. Auto-reconnect cookie data
2. Logon notification data

If a field is not present, the next field which is present is read.

Pad (570 bytes): 570 bytes. Padding. Values in this field are ignored.

2.2.10.1.1.4.1 Logon Info Field (TS_LOGON_INFO_FIELD)

The TS_LOGON_INFO_FIELD is used to encapsulate extended logon information field data of variable length.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
cbFieldData																															
FieldData (variable)																															
...																															

cbFieldData (4 bytes): A 32-bit unsigned integer. The size in bytes of the variable length data in the **FieldData** field.

FieldData (variable): Variable length data conforming to the structure for the type given in the **FieldsPresent** field of the Logon Info Extended (section 2.2.10.1.1.4) structure.

2.2.10.1.1.4.1.1 Logon Errors Info (TS_LOGON_ERRORS_INFO)

The TS_LOGON_ERRORS_INFO structure contains information which describes a logon error notification.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">ErrorNotificationType</td></tr>
<tr><td colspan="32">ErrorNotificationData</td></tr>
</table>

ErrorNotificationType (4 bytes): A 32-bit unsigned integer. The type code of the notification.

Value	Meaning
LOGON_FAILED_BAD_PASSWORD 0x00000000	The logon process failed. The logon credentials which were supplied are invalid.
LOGON_FAILED_UPDATE_PASSWORD 0x00000001	The logon process failed. The user cannot continue with the logon process until the password is changed.
LOGON_FAILED_OTHER 0x00000002	The logon process failed. The reason for the failure can be deduced from the **ErrorNotificationData** field.
LOGON_WARNING 0x00000003	The user received a warning during the logon process. The reason for the warning can be deduced from the **ErrorNotificationData** field.

ErrorNotificationData (4 bytes): A 32-bit unsigned integer. Error code describing the reason for the notification. Microsoft RDP servers populate this field with an NTSTATUS error code (see [ERRTRANS] for information on translating NTSTATUS error codes to usable text strings) which describes the issue which triggered the error.

2.2.11 Controlling Server Graphics Output

2.2.11.1 Inclusive Rectangle (TS_RECTANGLE16)

The TS_RECTANGLE16 structure describes a rectangle expressed in inclusive coordinates (the right and bottom coordinates are included in the rectangle bounds).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="16">left</td><td colspan="16">top</td></tr>
<tr><td colspan="16">right</td><td colspan="16">bottom</td></tr>
</table>

left (2 bytes): A 16-bit unsigned integer. The leftmost bound of the rectangle.

top (2 bytes): A 16-bit unsigned integer. The upper bound of the rectangle.

right (2 bytes): A 16-bit unsigned integer. The rightmost bound of the rectangle.

bottom (2 bytes): A 16-bit unsigned integer. The lower bound of the rectangle.

2.2.11.2 Client Refresh Rect PDU

The Refresh Rect PDU allows the client to request that the server redraw one or more rectangles of the session screen area. The client can use it to repaint sections of the client window that were obscured by other windowed applications. Server support for this PDU is indicated in the General Capability Set (section 2.2.7.1.1).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">refreshRectPduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given [T125] in section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Refresh Rect PDU data (section 2.2.11.2.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

refreshRectPduData (variable): TS_REFRESH_RECT_PDU (variable number of bytes): The actual contents of the Refresh Rect PDU, as specified in section 2.2.11.2.1.

2.2.11.2.1 Refresh Rect PDU Data (TS_REFRESH_RECT_PDU)

The TS_REFRESH_RECT_PDU structure contains the contents of the Refresh Rect PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) and two fields.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">shareDataHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="16">...</td><td colspan="8">numberOfAreas</td><td colspan="8">pad3Octects</td></tr>
<tr><td colspan="16">...</td><td colspan="16">areasToRefresh (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

shareDataHeader (18 bytes): A Share Data Header containing information about the packet. The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_REFRESH_RECT (33).

numberOfAreas (1 byte): An 8-bit unsigned integer. The number of Inclusive Rectangle (section 2.2.11.1) structures in the **areasToRefresh** field.

pad3Octects (3 bytes): A 3 element array of 8-bit unsigned integer values. Padding. Values in this field are ignored.

areasToRefresh (variable): An array of TS_RECTANGLE16 structures (variable number of bytes). Array of screen area Inclusive Rectangles to redraw. The number of rectangles is given by the **numberOfAreas** field.

2.2.11.3 Client Suppress Output PDU

The Suppress Output PDU is sent by the client to toggle all display updates from the server. This packet does not end the session or socket connection. Typically, a client sends this packet when its window is either minimized or restored.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>2
0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>3
0</th><th>1</th></tr>
<tr><td colspan="32">tpktHeader</td></tr>
<tr><td colspan="24">x224Data</td><td colspan="8">mcsSDrq (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">securityHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">suppressOutputPduData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable length PER-encoded MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is given in [T125] section 7, part 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Client Suppress Output PDU Data (section 2.2.11.3.1).

securityHeader (variable): Optional security header. If Standard RDP Security is in effect and the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), then this field will contain one of the following headers:

- Non-FIPS Security Header (section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2) or ENCRYPTION_LEVEL_HIGH (3).

- FIPS Security Header (section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see section 5.4) is in effect or the Encryption Method selected by the server (see sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), then this header is not included in the PDU.

suppressOutputPduData (variable): TS_SUPPRESS_OUTPUT_PDU (variable number of bytes):

The actual contents of the Suppress Output PDU, as specified in section 2.2.11.3.1.

2.2.11.3.1 Suppress Output PDU Data (TS_SUPPRESS_OUTPUT_PDU)

The TS_SUPPRESS_OUTPUT_PDU structure contains the contents of the Suppress Output PDU, which is a Share Data Header (section 2.2.8.1.1.1.2) and two fields.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
shareDataHeader																															
...																															
...																															
...																															
...																supressOutput								pad3Octects							
...																desktopRect															
...																															
...																															

shareDataHeader (18 bytes): AShare Data Header containing information about the packet (see section 2.2.8.1.1.1.2). The **type** subfield of the **pduType** field of the Share Control Header (section 2.2.8.1.1.1.1) MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_SUPPRESS_OUTPUT (35).

supressOutput (1 byte): An 8-bit unsigned integer. If set to zero, all screen updates from the server are turned off. Any value greater than zero will restore display updates from the server.

pad3Octects (3 bytes): A 3 element array of 8-bit unsigned integer values. Padding. Values in this field are ignored.

desktopRect (8 bytes): An Inclusive Rectangle (section 2.2.11.1) which contains the coordinates of the virtual desktop if the **suppressOutput** field is greater than zero. If the **suppressOutput** field is set to zero, this field is not included in the PDU.

3 Protocol Details

The following sections specify details of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification, including abstract data models and message processing rules.

3.1 Common Details

3.1.1 Abstract Data Model

No common abstract data model is specified.

3.1.2 Timers

No common timers are used.

3.1.3 Initialization

No common initialization steps are specified.

3.1.4 Higher-Layer Triggered Events

No common higher-layer triggered events are used.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Disconnection Sequences

3.1.5.1.1 Sending of MCS Disconnect Provider Ultimatum PDU

The structure and fields of the MCS Disconnect Provider Ultimatum PDU are specified in section 2.2.2.1.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The MCS Disconnect Provider Ultimatum PDU (embedded within the **mcsDPum** field) is specified in detail in [T125]. Only the rn- provider-initiated (1) or rn-user-requested (3) reason codes SHOULD be used in the reason field.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.1.5.1.2 Processing of MCS Disconnect Provider Ultimatum PDU

The structure and fields of the MCS Disconnect Provider Ultimatum PDU are specified in section 2.2.2.1.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Disconnect Provider Ultimatum PDU (embedded within the **mcsDPum** field) is specified in detail in [T125].

3.1.5.2 Static Virtual Channels

3.1.5.2.1 Sending of Virtual Channel PDU

The Virtual Channel PDU is transmitted by both the client and the server. Its structure and fields are specified in Virtual Channel PDU (section 2.2.6.1).

If the client is sending the virtual channel data, Construction of a Basic Client-to-Server Slow-Path PDU (section 3.2.5.1) describes how to construct the PDU. Construction of a Basic Server-to-Client Slow-Path PDU (section 3.3.5.1) describes how to construct the PDU if the server is sending the virtual channel data. In both of these referenced descriptions, the instructions regarding the Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2) MUST be ignored since it is not present in the PDU.

The **mcsPdu** field encapsulates either an MCS Send Data Request PDU or an MCS Send Data Indication PDU (see [T125]). In either case, the embedded **channelId** field MUST contain the server-assigned virtual channel ID – the static virtual channels are requested by name in the Client Network Data (see section 2.2.1.3.4), and the server-assigned IDs for each of those channels are enumerated in the Server Network Data (see section 2.2.1.4.4).

The usage of compression for virtual channel traffic is negotiated in the Virtual Channel Capability Set (see section 2.2.7.1.11), while the highest compression level supported by the client is advertised in the Client Info PDU (see section 3.2.5.3.11). If compression of the opaque virtual channel traffic has been negotiated, the sending entity SHOULD compress the data before it is encrypted.

If compression is to be applied to client-to-server traffic, MPCC-8K MUST be used (for scalability reasons), while the compression type to apply to server-to-client traffic MUST be the highest type advertised by the client and supported by the server. Furthermore, server-to-client virtual channel traffic MUST always be compressed with the same history buffer used to compress any other server-to-client RDP traffic (see Abstract Data Model (section 3.1.8.1)).

The resultant virtual channel data sent on the wire (contained in the **virtualChannelData** field) MUST be smaller than 1600 bytes in length. If the data is chunked, the Channel PDU Header (section 2.2.6.1.1) flags MUST be updated appropriately.

3.1.5.2.2 Processing of Virtual Channel PDU

The Virtual Channel PDU is received by both the client and the server. Its structure and fields are specified in 2.2.6.1.

If the client has received the virtual channel data, 3.3.5.2 describes how to process the PDU. 3.3.5.2 describes how to process the PDU if the server has received the virtual channel data. In both of these referenced descriptions, the instructions regarding the Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2) MUST be ignored since it is not present in the PDU.

The **mcsPdu** field encapsulates either an MCS Send Data Request PDU or an MCS Send Data Indication PDU (see [T125]). In either case, the embedded **channelId** field MUST contain the server-assigned virtual channel ID. This ID is used to route the **virtualChannel** payload to the appropriate virtual channel endpoint after decryption of the PDU and any necessary decompression of the payload has been conducted.

The received virtual channel data can span multiple Virtual Channel PDUs as the data MUST be chunked into 1600 byte blocks by the sender. Individual implementations can decide whether to pass on individual chunks as they are received or to assemble the separate chunks of data into a complete block before passing it on to the appropriate virtual channel endpoint.

3.1.6 Timer Events

No common timer events are used.

3.1.7 Other Local Events

No additional events are used.

3.1.8 MPPC-Based Bulk Data Compression

RDP uses a modified form of the Microsoft Point-to-Point Compression (MPPC) protocol to perform bulk compression of the PDU contents. This protocol is described in [RFC2118]. There are two forms of compression used at the server and client:

1. The original MPPC protocol, with an 8K history buffer (MPPC-8K).
2. A modified version of MPPC which uses a 64K history buffer and implements rearranged Huffman style encoding for the bitstream formats (MPPC-64K).

Both the server and client may operate as the sender of compressed data. Server-to-client compression can be used for Fast-Path output data (see Fast-Path Update (TS_FP_UPDATE) (section 2.2.9.1.2.1)), Slow-Path output data (see Slow-Path (T.128) Format (section 2.2.9.1.1)) or virtual channel data (see Virtual Channel PDU (section 2.2.6.1)). Client-to-server compression can currently only be used for virtual channel data (see Virtual Channel PDU (section 2.2.6.1)).

The client advertises the maximum compression type it supports in the Client Info PDU see Client Security Exchange PDU (section 2.2.1.10). In response the server selects a compression type within the range advertised by the client. This compression type is then used when performing all subsequent server-to-client and client-to-server bulk compression.

The compression type usage is indicated on a per-PDU basis by compression flags which are set in the header flags associated with each PDU. Besides being used to indicate the compression type, the compression flags are also used to communicate compression state changes which are required to maintain state synchronization. The header used to transmit the compression flags will depend on the type of data payload, such as Fast-Path output data see Fast-Path Update (TS_FP_UPDATE) (section 2.2.9.1.2.1), virtual channel data (see Virtual Channel PDU (section 2.2.6.1)) or Slow-Path data see Slow-Path (T.128) Format (section 2.2.9.1.1).

3.1.8.1 Abstract Data Model

The shared state necessary to support the transmission and reception of compressed data between a client and server requires a history buffer and a current offset into the history buffer (**HistoryOffset**). The size of the history buffer depends on the compression type being used (8 KB for MPPC-8K and 64 KB for MPPC-64K). The history buffer and **HistoryOffset** MUST both start initialized to zero.

While compressing data, the sender endpoint inserts the uncompressed data at the position in the history buffer given by the **HistoryOffset**. After insertion, the **HistoryOffset** is advanced by the amount of data added. If the data does not fit into the history buffer (the sum of the **HistoryOffset** and the size of the uncompressed data exceeds the size of the history buffer), the **HistoryOffset** MUST be reset to the start of the history buffer (offset 0).

As the receiver endpoint decompresses the data, it inserts the decompressed data at the position in the history buffer given by its local copy **HistoryOffset**. If a reset occurs, the sender endpoint MUST notify the target receiver so it can reset its local state. In this way, the sender and receiver endpoints maintain an exact replica of the history buffer and **HistoryOffset**.

3.1.8.2 Compressing Data

The uncompressed data is first inserted into the local history buffer at the position indicated by HistoryOffset by the sender. The compressor then runs through the length of newly added uncompressed data to be sent and produces as output a sequence of literals (bytes to be sent uncompressed) or copy-tuples which consists of a <copy-offset, length-of-match> pair.

The copy-offset component of the copy-tuple is an index into HistoryBuffer (counting backwards from the current byte being compressed in the history buffer towards the start of the buffer) where there is a match to the data to be sent. The length-of-match component is the length of that match in bytes. If the resulting data is not smaller than the original bytes (that is, expansion instead of compression results), then this results in a flush and the data is sent uncompressed so as never to send more data than the original uncompressed bytes.

In this way the compressor aims to reduce the size of data that needs to be transmitted. For example, consider the following string:

```
0         1         2         3         4
01234567890123456789012345678901234567890
for whom the bell tolls, the bell tolls for thee.
```

The compressor would produce:

```
for whom the bell tolls,<16,15> <40,4><19,3>e.
```

The <16,15> tuple is the compression of '.the.bell.tolls' and <40,4> is 'for.', <19,3> gives 'the'. (The '.' values indicate space characters.)

The literal and copy-tuples are then encoded using the MPPC encoding scheme for the 8K and 64K variants see MPPC-8K (section 3.1.8.4.1) and MPPC-64K (section 3.1.8.4.2) respectively.

3.1.8.2.1 Setting the Compression Flags

The sender MUST always specify the compression flags associated with a compressed payload. These flags MUST be set in the header field appropriate to the type of data payload, such as Fast-Path output data (see Fast-Path Update (TS_FP_UPDATE) (section 2.2.9.1.2.1), virtual channel data (see Virtual Channel PDU (section 2.2.6.1), or Slow-Path data (see Slow-Path (T.128) Format (section 2.2.9.1.1).

The compression flags are produced by performing a logical OR operation of the compression type with one or more of the following flags.

Compression flag	Meaning
PACKET_FLUSHED 0x80	Used to indicate that the history buffer MUST be reinitialized. This value corresponds to MPPC bit A (see [RFC2118] section 3.1). This flag MUST be set without setting any other flags except the compression type. This flag MUST be set if the compression would generate an expansion of the data and indicates to the decompressor that it should reset its history buffer, HistoryOffset value and restart on reception of the next batch of compressed bytes. If this condition occurs, the data MUST be sent in uncompressed form.
PACKET_AT_FRONT 0x40	Used to indicate that the decompressed data MUST be placed at the beginning of the local history buffer. This value corresponds to MPPC bit B (see section 3.1 in [RFC2118]. This flag MUST be set in conjunction with the PACKET_COMPRESSED (0x80) flag. There are two conditions on the "compressor-side" that generate this scenario: (1) this is the first packet to be compressed and (2) the data to be compressed will not fit at the end of the history buffer but instead needs to be placed at the start of the history buffer.
PACKET_COMPRESSED 0x20	Used to indicate that the data is compressed. This value corresponds to MPPC bit C (see [RFC2118] section 3.1). This flag MUST be set when compression of the data was successful.

The flowchart in Figure 4 illustrates the general operation of the compressor and the production of the various compression flags.

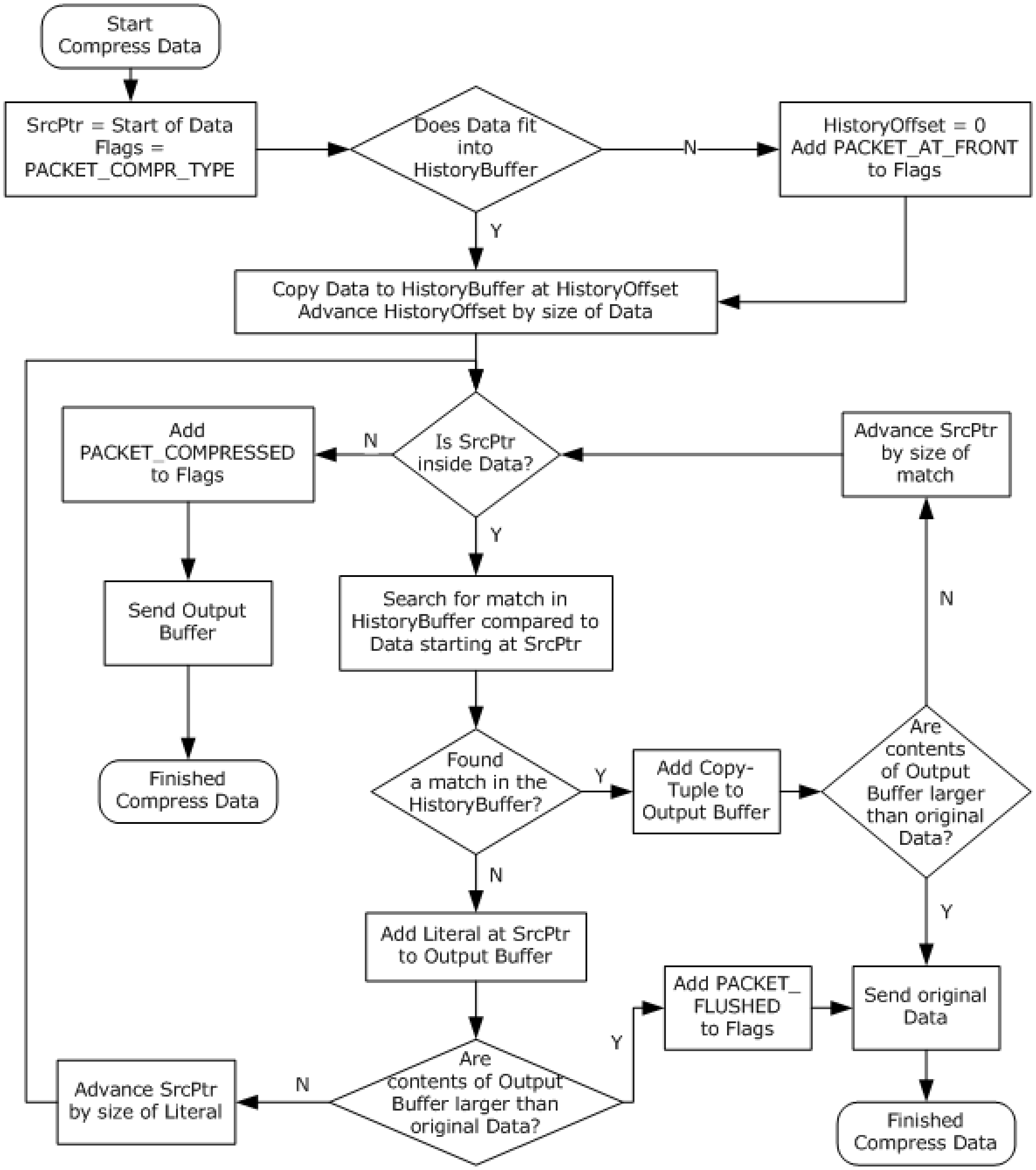


Figure 5: Operation of the bulk compressor

3.1.8.3 Decompressing Data

An endpoint which receives compressed data MUST decompress the data and store the resultant data at the end of the history buffer. The order of actions depends on the compression flags associated with the compressed data.

Compression flag	Meaning
PACKET_FLUSHED 0x80	If this flag is set, the decompressor MUST reset its state, by clearing the history buffer and resetting the HistoryOffset to 0.
PACKET_AT_FRONT 0x40	If this flag is set, the decompressor MUST start decompressing to the start of the history buffer, by resetting the HistoryOffset to 0. Otherwise, the decompressor MUST append the decompressed data to the end of the history buffer.
PACKET_COMPRESSED 0x20	If this flag is set, the decompressor MUST decompress the data, appending the decompressed data to the history buffer and advancing the HistoryOffset by the size of the resulting decompressed data.

3.1.8.4 Compression Types

3.1.8.4.1 MPPC-8K

3.1.8.4.1.1 Literal Encoding

Literals are bytes sent uncompressed. If the value of a literal is below 0x80, it is not encoded in any special manner. If the literal has a value greater than 0x7F it is sent as the bits 10 followed by the lower 7 bits of the literal. For example, 0x56 is transmitted as the binary value 01010110, while 0xE7 is transmitted as the binary value 101100111.

3.1.8.4.1.2 Copy-Tuple Encoding

Copy-tuples consist of a <copy-offset> and <length-of-match> pair; see Compressing Data (section 3.1.8.2) for more details.

3.1.8.4.1.2.1 Copy-Offset Encoding

Encoding of the copy-offset value is performed according to the following table:

Copy-Offset range	Encoding (binary header + copy-offset bits)
0...63	1111 + lower 6 bits of copy-offset
64...319	1110 + lower 8 bits of (copy-offset – 64)
320...8191	110 + lower 13 bits of (copy-offset – 320)

For example:

- A copy-offset value of 3 is encoded as the binary value 1111 000011.
- A copy-offset value of 128 is encoded as the binary value 1110 01000000
- A copy-offset value of 1024 is encoded as the binary value 110 0001011000000.

A copy-offset value MUST be followed by a length-of-match value.

3.1.8.4.1.2.2 Length-of-Match Encoding

Encoding of the length-of-match (L-o-M) value is performed according to the following table.

L-o-M range	Encoding (binary header + L-o-M bits)
3	0
4...7	10 + 2 lower bits of (L-o-M - 4)
8...15	110 + 3 lower bits of (L-o-M – 8)
16...31	1110 + 4 lower bits of (L-o-M – 16)
32...63	11110 + 5 lower bits of (L-o-M – 32)
64...127	111110 + 6 lower bits of (L-o-M – 64)
128...255	1111110 + 7 lower bits of (L-o-M – 128)
256...511	11111110 + 8 lower bits of (L-o-M – 256)
512...1023	111111110 + 9 lower bits of (L-o-M – 512)
1024...2047	1111111110 + 10 lower bits of (L-o-M – 1024)
2048...4095	11111111110 + 11 lower bits of (L-o-M – 2048)
4096...8191	111111111110 + 12 lower bits of (L-o-M – 4096)

3.1.8.4.2 MPPC-64K

The rules for MPPC-64K are very similar to those of MPPC-8K (see MPPC-8K (section 3.1.8.4.1). MPPC-64K has a history buffer size of 64 KB, thus both endpoints MUST maintain a 64 K window.

3.1.8.4.2.1 Literal Encoding

Literals are bytes sent uncompressed. If the value of a literal is below 0x80, it is not encoded in any special manner. If the literal has a value greater than 0x7F it is sent as the bits 10 followed by the lower 7 bits of the literal. For example, 0x56 is transmitted as the binary value 01010110, while 0xE7 is transmitted as the binary value 101100111.

3.1.8.4.2.2 Copy-Tuple Encoding

Copy-tuples consist of a <copy-offset> and <length-of-match> pair; see Compressing Data (section 3.1.8.2) for more details.

3.1.8.4.2.2.1 Copy-Offset Encoding

Encoding of the copy-offset value is performed according to the following table.

Copy-offset range	Encoding (binary header + copy-offset bits)
0...63	11111 + lower 6 bits of offset
64...319	11110 + lower 8 bits of (offset – 64)
320...2367	1110 + lower 11 bits of (offset – 320)
2368+	110 + lower 16 bits of (offset – 2368)

A copy-offset value MUST be followed by a length-of-match value.

3.1.8.4.2.2.2 Length-of-Match Encoding

Encoding of the length-of-match (L-o-M) value is performed according to the following table.

L-o-M range	Encoding (binary header + L-o-M bits)
3	0
4..7	10 + 2 lower bits of (L-o-M – 4)
8..15	110 + 3 lower bits of (L-o-M – 8)
16..31	1110 + 4 lower bits of (L-o-M – 16)
32..63	11110 + 5 lower bits of (L-o-M – 32)
64..127	111110 + 6 lower bits of (L-o-M – 64)
128..255	1111110 + 7 lower bits of (L-o-M – 128)
256..511	11111110 + 8 lower bits of (L-o-M – 256)
512..1023	111111110 + 9 lower bits of (L-o-M – 512)
1024..2047	1111111110 + 10 lower bits of (L-o-M – 1024)
2048..4095	11111111110 + 11 lower bits of (L-o-M – 2048)
4096..8191	111111111110 + 12 lower bits of (L-o-M – 4096)
8192..16383	1111111111110 + 13 lower bits of (L-o-M – 8192)
16384..32767	11111111111110 + 14 lower bits of (L-o-M – 16384)
32768..65535	111111111111110 + 15 lower bits of (L-o-M – 32768)

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

Note The conceptual data below can be implemented using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.2.1.1 Received Server Data

The Received Server Data store contains data received from the server during execution of the Remote Desktop Protocol. This store is initialized when processing the MCS Connect Response PDU with GCC Conference Create Response (see sections 2.2.1.4 and 3.2.5.3.4).

3.2.1.2 Static Virtual Channel IDs

The Static Virtual Channel IDs store contains the MCS channel identifiers of the static virtual channels. This data store is initialized when processing the Server Network Data (see sections 2.2.1.4.4 and 3.2.5.3.4).

3.2.1.3 I/O Channel ID

The I/O Channel ID store contains the MCS channel identifier of the I/O channel. This data store is initialized when processing the Server Network Data (see sections 2.2.1.4.4 and 3.2.5.3.4).

3.2.1.4 User Channel ID

The User Channel ID store contains the MCS channel identifier of the user channel. This data store is initialized when processing the MCS Attach User Confirm PDU (see sections 2.2.1.7 and 3.2.5.3.7).

3.2.1.5 Server Channel ID

The Server Channel ID store contains the MCS channel identifier of the server channel. This data store is initialized when processing the Demand Active PDU, see sections 2.2.1.13.1.1 and 3.2.5.3.13.1.

3.2.1.6 Server Capabilities

The Server Capabilities store contains capability sets (see Versioning and Capability Negotiation (section 1.7) and 0) received from the server in the Demand Active PDU (see Server Demand Active PDU (section 2.2.1.13.1) and Processing of Demand Active PDU (section 3.2.5.3.13.1)). The client MUST ensure that it does not violate any of the server capabilities when sending data to the server—for example, if the server does not support Fast-Path input (see section Input Capability Set (TS_INPUT_CAPABILITY_SET)), the client MUST only send Slow-Path input PDUs. In effect, the client MUST ensure that all of the RDP traffic which it sends on the wire is consistent with the expectations of the server as described by the data held in the Server Capabilities Store.

3.2.1.7 Share ID

The Share ID store holds the share identifier selected by the server (see [T128] section 8.4.2 for more information regarding share IDs). This data store is initialized when processing the Demand Active PDU (see Server Demand Active PDU (section 2.2.1.13.1) and Processing of Demand Active PDU (section 3.2.5.3.13.1)) and is used to initialize the **shareId** field of the Share Data Header when sending basic client-to-server Slow-Path PDUs (Construction of a Basic Client-to-Server Slow-Path PDU (section 3.2.5.1)).

3.2.1.8 Automatic Reconnection Cookie

The Automatic Reconnection Cookie store contains a cookie received from the server which may be used to seamlessly auto-reconnect if the connection is broken due to short-term network failure (see Automatic Reconnection (section 5.5)). The cookie is received in a Save Session Info PDU (see Server Save Session Info PDU (section 2.2.10.1) and Processing of Save Session Info PDU (section 3.2.5.9.5.1).

3.2.1.9 Server Licensing Encryption Ability

The Server Licensing Encryption Ability store determines whether the server has the ability to handle encrypted licensing packets when using Standard RDP Security mechanisms (see the discussion of the SEC_LICENSE_ENCRYPT_CS flag in Basic (TS_SECURITY_HEADER) (section 2.2.8.1.1.2.1)). This fact is communicated to the client by setting the SEC_LICENSE_ENCRYPT_CS (0x0200) flag in all licensing PDUs sent from the server.

3.2.1.10 Pointer Image Cache

The Pointer Image Cache contain a collection of pointer images saved from Color Pointer Updates (see sections Fast-Path Color Pointer Update (TS_FP_COLORPOINTERATTRIBUTE) (section 2.2.9.1.2.1.7), Processing of Slow-Path Pointer Update PDU (section 3.2.5.9.2) and Processing of Fast-Path Update PDU (section 3.2.5.9.3)) and New Pointer Updates (see sections Fast-Path New Pointer Update (TS_FP_POINTERATTRIBUTE), Processing of Slow-Path Pointer Update PDU (section 3.2.5.9.2) and Processing of Fast-Path Update PDU (section 3.2.5.9.3)). The images stored in the cache are used to set the shape of the pointer when processing a Cached Pointer Update (see sections Cached Pointer Update (TS_CACHEDPOINTERATTRIBUTE) (section 2.2.9.1.1.4.6), Processing of Slow-Path Pointer Update PDU (section 3.2.5.9.2) and Processing of Fast-Path Update PDU (section 3.2.5.9.3)). The size and color depth (either variable or fixed at 24 bits-per-pixel) of the cache is negotiated in the Pointer Capability Set (see Pointer Capability Set (TS_POINTER_CAPABILITY_SET) (section 2.2.7.1.6)).

3.2.2 Timers

No client timers are used.

3.2.3 Initialization

No client initialization steps are specified.

3.2.4 Higher-Layer Triggered Events

No client higher-layer triggered events are used.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Constructing a Basic Client-to-Server Slow-Path PDU

The majority of client-to-server Slow-Path PDUs have the same basic structure (see sections 5.3.8 and 5.4.4):

- **tpktHeader**: TPKT Header (see [T123] section 8)
- **x224Data**: X.224 Data TPDU (see [X224] section 13.7)
  - **mcsSDrq**: MCS Send Data Request PDU (see [T125] section 7, Part 7)
    - **securityHeader**: Optional Security Header (see section 2.2.8.1.1.2)
    - shareDataHeader: Share Data Header (see section 2.2.8.1.1.1.2)
    - Actual PDU Contents (see section 2.2 describing the PDU structure and fields)

The PDUs conforming to this basic structure MAY be constructed using the same techniques.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The **mcsSDrq** field is initialized as specified in [T125]. The embedded **initiator** field MUST be set to the MCS user channel ID (held in the User Channel ID store specified in section 3.2.1.4) and the embedded **channelId** field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID store described in section 3.2.1.3). The embedded **userData** field contains the remaining fields of the PDU.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario, the **securityHeader** field MUST NOT be present.

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the optional **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level and Encryption Method selected by the server as part of the negotiation specified in section 5.3.2), using the methods and techniques specified in 5.3.6. The format of the **securityHeader** field is selected (as specified in section 2.2), and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded flags field of the securityHeader field MUST contain the SEC_ENCRYPT (0x0008) flag.

The **shareDataHeader** field contains a Share Data Header structure as described in Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2). The pduSource of the embedded Share Control Header MUST be set to the MCS user channel ID (held in the User Channel ID (section 3.2.1.4) store). If the contents of the PDU are to be compressed (this MUST be done before any MAC signature is constructed and encryption methods applied), the embedded **compressedType** field of the shareDataHeader MUST be initialized as specified in Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2). The remaining **Share Data Header** and **Share Control Header** fields MUST be populated as specified in Share Control Header (TS_SHARECONTROLHEADER) (section 2.2.8.1.1.1.1), Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2) and, section 2.2.

The remaining fields are populated as specified in the section 2.2.

3.2.5.2 Processing a Basic Server-to-Client Slow-Path PDU

The majority of server-to-client Slow-Path PDUs have the same basic structure (see sections 5.3.8 and 5.4.4):

- **tpktHeader**: TPKT Header (see [T123] section 8)
- **x224Data**: X.224 Data TPDU (see [X224] section 13.7)
    - **mcsSDin**: MCS Send Data Indication PDU (see [T125] section 7, part 7)
        - **securityHeader**: Optional Security Header (see section 2.2.8.1.1.2)
        - **shareDataHeader**: Share Data Header (see Share Data Header (TS_SHAREDATAHEADER) (section 2.2.8.1.1.1.2))
        - Actual PDU Contents (see section 2.2 describing the PDU structure and fields)

The PDUs conforming to this basic structure MAY be processed using the same techniques.

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]), **x224Data** (see [X224]), and **mcsSDin** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDin** is used to route the PDU to the appropriate target channel.

The conditions mandating the presence of the **securityHeader** field, as well as the type of Security Header structure present in this field, are explained in the section specifying the PDU structure and fields (see section 2.2). If the **securityHeader** field is present, the embedded flags field MUST be examined for the presence of the **SEC_ENCRYPT (0x0008)** flag (see Basic (TS_SECURITY_HEADER) (section 2.2.8.1.1.2.1)), and, if it is present, the data following the **securityHeader** field MUST be verified and decrypted using the methods and techniques specified in section 5.3.6. If the MAC signature is incorrect, or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect, and the **SEC_ENCRYPT (0x0008)** flag is present, the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol (section 5.4.5) provider.

3.2.5.3 Normal Connection Sequence

3.2.5.3.1 Sending X.224 Connection Request PDU

The structure and fields of the X.224 Connection Request PDU are specified in section 2.2.1.1.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Crq** field is initialized as specified in [X224] (the Destination reference and Source reference fields are both set to zero, and the Class and options field is set to zero). Parameter fields MUST NOT be specified in the variable part of the Connection Request PDU. This implies that the default maximum size of an X.224 Data PDU payload (65528 bytes) is used, since the maximum TPDU size and preferred maximum TPDU size are not present.

The **routingToken** field is optional. The client may have obtained a routing token from an external source (via a scriptable API), or it may have received a token embedded within the Server Redirection PDU (see[MS-RDPEGDI] section 2.2.3). If the client does not have a routing token, it MAY construct one. In the absence of a routing token, the Microsoft RDP client sends the username (truncated to nine characters) in the following format:

Cookie: mstshash=*username*

The optional **rdpNegData** field contains an RDP Negotiation Request structure, as specified in section 2.2.1.1.1. The **requestedProtocols** field is initialized with flags describing the security protocols which the client supports (see section 5.4 for more details on Enhanced RDP Security).

3.2.5.3.2 Processing X.224 Connection Confirm PDU

The structure and fields of the X.224 Connection Confirm PDU are specified in section 2.2.1.2.

The embedded length fields within the tpktHeader (see [T123]) and **x224Ccf** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped. The Destination reference, Source reference and Class and options fields within the **x224Ccf** field MAY be ignored.

If the **rdpNegData** field is not present, it is assumed that the server does not support Enhanced RDP Security (section 5.4) and the protocol selected by the server is implicitly assumed to be **PROTOCOL_RDP (0x00000000)**. If the **rdpNegData** is present, then it MUST contain either a

RDP Negotiation Response structure or RDP Negotiation Failure structure. If any other structure is present, the connection SHOULD be dropped.

If an RDP Negotiation Failure structure is present, the failure code is extracted from the **failureCode** field and the connection SHOULD be dropped (see section 2.2.1.2.2 for a list of failure codes). If an RDP Negotiation Response structure is present, the **selectedProtocol** field is parsed to extract the selected protocol identifier (see section 2.2.1.2.1 for a list of identifiers).

If an External Security Protocol (section 5.4.5) will be used for the duration of the connection, the client MUST execute the selected protocol at this stage by calling into the relevant External Security Protocol (section 5.4.5) provider. Once the External Security Protocol (section 5.4.5) handshake has run to completion, the client MUST continue with the connection sequence by sending the MCS Connect Initial PDU to the server over the newly established secure channel (see section 3.2.5.3.3).

If Standard RDP security mechanisms (see section 5.3) are to be used, that is, the protocol selected by the server is PROTOCOL_RDP (0x00000000), then the client MUST continue with the connection sequence by sending the MCS Connect Initial PDU with GCC Conference Create Request to the server (see section 2.2.1.3).

3.2.5.3.3 Sending MCS Connect Initial PDU with GCC Conference Create Request

The structure and fields of the MCS Connect Initial PDU with GCC Conference Create Request are specified in section 2.2.1.3. A basic high-level overview of the nested structure for the MCS Connect Initial PDU is illustrated in Figure 2.

The tpktHeader field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The MCS Connect Initial PDU (embedded within the mcsCi field) is specified in detail in [T125]. The client SHOULD initialize the fields of the MCS Connect Initial PDU as follows:

Connect initial field	Value
calledDomainSelector	0x01
callingDomainSelector	0x01
upwardFlag	TRUE
targetParameters	See table which follows.
minimumParameters	See table which follows.
maximumParameters	See table which follows.
userData	GCC Conference Create Request

The targetParameters, minimumParameters, and maximumParameters domain parameter structures SHOULD be initialized as follows:

Domain parameter	targetParameters	minimumParameters	maximumParameters
maxChannelIds	34	1	65535
maxUserIds	2	1	65535
maxTokenIds	0	1	65535

Domain parameter	targetParameters	minimumParameters	maximumParameters
numPriorities	1	1	1
minThroughput	0	0	0
maxHeight	1	1	1
maxMCSPDUsize	65535	1056	65535
protocolVersion	2	2	2

The **userData** field of the MCS Connect Initial PDU contains the GCC Conference Create Request (embedded within the **gccCCrq** field). The GCC Conference Create Request is specified in detail in [T124] and appended as user data to the MCS Connect Initial PDU using the format specified in [T124] sections 9.5 and 9.6. The client SHOULD initialize the fields of the GCC Conference Create Request (section 3.3.5.3.3) as follows:

Conference create request field	Value
conferenceName	"1"
convenerPassword	Optional field, not used.
password	Optional field, not used.
lockedConference	FALSE
listedConference	FALSE
conductibleConference	FALSE
terminationMethod	automatic (0)
conductorPrivileges	Optional field, not used.
conductedPrivileges	Optional field, not used.
nonConductedPrivileges	Optional field, not used.
conferenceDescription	Optional field, not used.
callerIdentifier	Optional field, not used.
userData	Basic client settings data blocks

The **userData** field of the GCC Conference Create Request (section 3.3.5.3.3) MUST be initialized with basic client settings data blocks (see sections 2.2.1.3.2 through to 2.2.1.3.5). The client-to-server H.221 key which MUST be embedded at the start of the userData field (see [T124] section 8.7 for a description of the structure of user data) is the ANSI text "Duca".

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.2.5.3.4 Processing MCS Connect Response PDU with GCC Conference Create Response

The structure and fields of the MCS Connect Response PDU with GCC Conference Create Response are specified in section 2.2.1.4. A basic high-level overview of the nested structure for the MCS Connect Response PDU is illustrated in Figure 2.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Connect Response PDU (embedded within the **mcsCrsp** field) is specified in [T125]. The client ignores the **calledConnectId** and **domainParameters** fields of this PDU. If the **result** field is set to rt-successful (0) the client MUST send the MCS Erect Domain Request PDU to the server (see section 3.3.5.3.5). If the result field is set to any other value, the client SHOULD drop the connection.

The **mcsCrsp** field of the MCS Connect Response PDU contains the GCC Conference Create Response data (embedded within the gccCCrsp field). The GCC Conference Create Response is described in detail in [T124] and appended as user data to the MCS Connect Response PDU using the format specified in [T124] sections 9.5 and 9.6. Microsoft RDP Servers incorrectly hard-code the length of the MCS Connect Response PDU user data as 0x2A (42) bytes —the client SHOULD ignore this incorrect length and MUST NOT generate an error.

The client ignores all of the GCC Conference Create Response fields, except for the userData field. The userData field of the GCC Conference Create Response MUST contain basic server settings data blocks (see sections 2.2.1.4.2 through 2.2.1.4.4). The client MUST check that the server-to-client H.221 key embedded at the start of the **x224Data** field (see [T124] section 8.7) for a description of the structure of user data) is the ANSI text "McDn". If this is not the case, the connection SHOULD be dropped.

All of the encoded lengths within the MCS Connect Response PDU and the GCC Conference Create Response (except for those already noted) MUST also be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

Once the **mcsCrsp** and **gccCCrsp** fields have been successfully parsed the client examines the basic server settings data blocks and stores the received data in the *Received Server Data* store (see section 3.3.1.1). However, before the data is stored the Basic Server Settings Data Blocks are checked for validity.

The clientRequestedProtocols field in the Server Core Data (see section 2.2.1.4.2) is examined to ensure that it contains the same flags that the client sent to the server in the RDP Negotiation Response (see section 3.2.5.3.1). If this is not the case, the client SHOULD drop the connection. In the event that this optional field is not present, the value PROTOCOL_RDP (0) MUST be assumed.

Select settings in the Server Security Data (see section 2.2.1.4.3) are validated using the following rules:

Server security data field	Validation rule
encryptionMethod	If this field does not contain a valid Encryption Method identifier, the client

Server security data field	Validation rule
	SHOULD drop the connection. If the client does not support the selected Encryption Method it SHOULD disconnect, as further communication with the server will not be possible.
encryptionLevel	If this field does not contain a valid Encryption Level identifier, the client SHOULD drop the connection.
serverRandomLen	If this field does not contain a value of 32, the client SHOULD drop the connection.
serverCertificate	If this field does not contain a valid certificate, the client SHOULD drop the connection. Proprietary certificates (see sections 3.2.5.3.1 and 5.3.3.1) can be tested for validity using the techniques specified in section 5.3.3.1.3.

The **channelCount** and **channelIdArray** fields in the Server Network Data (section 2.2.1.4.4) MUST be examined for consistency to ensure that the packet contains enough data to extract the specified number of channel IDs. If there is not enough data, the client SHOULD drop the connection. The MCS channel IDs returned in the **channelIdArray** should be saved in the Static Virtual Channel IDs store (see section 3.2.1.2), while the **MCSChannelId** field should be saved in the I/O Channel ID store (see section 3.2.1.3). These IDs will be used by the client when sending MCS Channel Join Request PDUs (see sections 2.2.1.8 and 3.2.5.3.8).

Once the basic server settings data blocks have been processed successfully, the client MUST send the MCS Attach User Request PDU (see section 3.2.5.3.6) to the server.

3.2.5.3.5 Sending MCS Erect Domain Request PDU

The structure and fields of the MCS Erect Domain Request PDU are specified in section 2.2.1.5.

The tpktHeader field is initialized as specified in [T123], while the x224Data field is initialized as specified in [X224].

The MCS Erect Domain Request PDU (embedded within the mcsEDrq field) is described in detail in [T125]. The client SHOULD initialize the both subHeight and subinterval fields of the MCS Erect Domain Request PDU to 0x01.

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.2.5.3.6 Sending MCS Attach User Request PDU

The structure and fields of the MCS Attach User Request PDU are specified in section 2.2.1.6.

The tpktHeader field is initialized as specified in [T123], while the x224Data field is initialized as specified in [X224].

The MCS Attach User Request PDU (embedded within the mcsAUrq field) is described in detail in [T125].

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.2.5.3.7 Processing MCS Attach User Confirm PDU

The structure and fields of the MCS Attach User Confirm PDU are specified in section 2.2.1.7.

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Attach User Confirm PDU (section 2.2.1.7) (embedded within the **mcsAUcf** field) is described in detail in [T125]. If the **result** field is not set to rt-successful (0), the client SHOULD drop the connection. If the **result** field is set to rt-successful (0) but the initiator is not present, the client SHOULD drop the connection. If the initiator is present, the client stores the value in the User Channel ID (section 3.2.1.4) store.

Once the user channel ID has been extracted, the client MUST send an MCS Channel Join Request PDU for the user channel (see section 2.2.1.7).

3.2.5.3.8 Sending MCS Channel Join Request PDU(s)

The structure and fields of the MCS Channel Join Request PDU are specified in section 2.2.1.8.

Multiple MCS Channel Join Request PDUs are sent to join the following channels:

1. User Channel (the MCS channel ID is stored in the User Channel ID (section 3.2.1.4) store).

2. I/O channel (the MCS channel ID is stored in the I/O Channel ID (section 3.2.1.3) store).

3. Static Virtual Channels (the MCS channel IDs are stored in the Static Virtual Channel IDs store).

The MCS Channel Join Request PDUs (section 2.2.1.8) are sent sequentially. The first PDU is sent after receiving the MCS Attach User Confirm PDU (see section 2.2.1.7) and subsequent PDUs are sent after receiving of the MCS Channel Join Confirm PDU (see section 2.2.1.9) for the previous request. Sending of the MCS Channel Join Request PDUs (section 2.2.1.8) MUST continue until all channels have been successfully joined.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The MCS Channel Join Request PDU (embedded within the **mcsCJrq** field) is described in detail in [T125]. The initiator field is initialized with the User Channel ID (section 3.2.1.4) obtained during the processing of the MCS Attach User Confirm PDU (see section 2.2.1.7) and stored in the User Channel ID (section 3.2.1.4) store. The **channelId** field is initialized with the MCS channel ID of the channel that is being joined.

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.2.5.3.9 Processing MCS Channel Join Confirm PDU(s)

The structure and fields of the MCS Channel Join Confirm PDU are specified in section 2.2.1.9.

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Channel Join Confirm PDU (section 2.2.1.9) (embedded within the **mcsCJcf** field) is specified in detail in [T125]. If the optional **channelId** field is not present, the client SHOULD drop the connection. Furthermore, if the **result** field is not set to rt-successful (0), the client SHOULD also drop the connection. The initiator and requested fields MAY be ignored, however, the **channelId** field MUST be examined. If the value of the **channelId** field does not correspond with the value of the **channelId** field sent in the previous MCS Channel Join Request PDU the connection SHOULD be dropped.

Once the client has successfully processed the MCS Channel Join Confirm PDU (section 2.2.1.9), it MUST send a new MCS Channel Join Request PDU to the server containing the ID of the next channel which has not yet been joined. If all channels have been joined, the client MUST proceed to send one of the following PDUs:

- The Security Exchange PDU (section 2.2.1.10) if Standard RDP Security (section 5.3) is in effect and the Encryption Level (section 5.3.1) and Encryption Method returned from the server in the Server Core Data (see sections 2.2.1.4.2 and 3.2.5.3.4) are both greater than zero.
- The Client Info PDU (section 2.2.1.11) if the Encryption Level (section 5.3.1) and Encryption Method returned from the server are both zero.

3.2.5.3.10 Sending Security Exchange PDU

The structure and fields of the Security Exchange PDU are specified in section 2.2.1.10.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as detailed in [X224].

The **mcsSDrq** field is initialized as specified in [T125]. The embedded initiator field MUST be set to the MCS user channel ID (held in the User Channel ID (section 3.2.1.4) store and the embedded **channelId** field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID store). The embedded **userData** field contains the remaining fields of the Security Exchange PDU (section 2.2.1.10).

The embedded **flags** field of the basicSecurityHeader MUST contain the SEC_EXCHANGE_PKT (0x0001) flag (specified in section 2.2.8.1.1.2.1) to indicate the PDU type. If the client can handle encrypted licensing packets from the server and Standard RDP Security mechanisms (see sections 5.3 and 5.4) are being used, then the SEC_LICENSE_ENCRYPT_SC (0x0200) flag SHOULD also be included in the **flags** subfield of the **basicSecurityHeader** field.

A 32-byte random number MUST be generated and then encrypted using the public key of the server and the techniques specified in section 5.3.4.1. The public key of the server is embedded in server's certificate which is held in the **serverCertificate** field of the Server Security Data (section 2.2.1.4.3) sent in the MCS Connect Response PDU with GCC Conference Response (see section 3.2.5.3.4). Once the 32-byte random number has been successfully encrypted, it MUST be copied into the **encryptedClientRandom** field. The size of the **encryptedClientRandom** field MUST be derived as specified in section 5.3.4.1. After the encrypted client random has been copied into the **encryptedClientRandom** buffer, 8 bytes of padding (which MUST be zeroed out) will remain.

Once the client has sent the Security Exchange PDU (section 2.2.1.10)it MUST generate the session keys which will be used to encrypt, decrypt and sign data sent on the wire. The 32-byte client random and server random (transmitted in the Server Security Data (section 2.2.1.4.3)) are used to accomplish this task by employing the techniques specified in section 5.3.5. On successful generation of the session keys, the client MUST send the Client Info PDU to the server (see section 2.2.1.11).

If Enhanced RDP Security (see section 5.4) is in effect, the Security Exchange PDU MUST not be sent.

3.2.5.3.11 Sending Client Info PDU

The structure and fields of the Client Info PDU are specified in section 2.2.1.11.

The tpktHeader field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The **mcsSDrq** field is initialized as specified in [T125]—the embedded initiator field MUST be set to the MCS user channel ID (held in the User Channel ID (section 3.2.1.4) store) and the embedded channelId field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID (section 3.2.1.3)). The embedded **userData** field contains the remaining fields of the Client Info PDU (section 2.2.1.11).

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature). The **securityHeader** field MUST be present, however it will contain a Basic Security Header structure (see section 2.2.8.1.1.2.1).

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level (section 5.3.1) and Encryption Method selected by the server as part of the negotiation specified in section 5.3.2) using the methods and techniques described in 5.3.6. The format of the **securityHeader** field is selected as described in the section detailing the PDU structure and fields (see section 2.2) and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded flags field of the **securityHeader** field MUST contain the SEC_ENCRYPT (0x0008) flag.

The embedded flags field of the **securityHeader** field (which is always present) MUST contain the SEC_INFO_PKT (0x0040) flag (specified in section 2.2.8.1.1.2.1) to indicate the PDU type.

If the client is in the process of attempting an automatic reconnection operation using a cookie stored in the Automatic Reconnection Cookie store, then it MUST populate the **autoReconnectCookie** field of the Extended Info Structure (see section 2.2.1.11.1.1.1) with the contents of the cookie. The remainder of the PDU MUST be populated with client settings according to the structure and type definition in section 2.2.1.11.1.1.

3.2.5.3.12 Processing License Error PDU - Valid Client

The structure and fields of the License Error (Valid Client) PDU are specified in section 2.2.1.12.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]), **x224Data** (see [X224]), and **mcsSDin** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDin** is used to route the PDU to the appropriate target channel.

The **securityHeader** field MUST always be present and it MUST contain at least a Basic Security Header structure (see section 2.2.8.1.1.2.1). The embedded flags field of the **securityHeader** MUST contain the SEC_LICENSE_PKT (0x0080) flag (specified in section 2.2.8.1.1.2.1). If this flag is not present then the packet cannot be handled as a Licensing PDU.

If the SEC_LICENSE_ENCRYPT_CS (0x0200) flag is present, then the server is able to accept encrypted licensing packets when using Standard RDP Security (see section 5.3) mechanisms. This fact is stored in the Server Licensing Encryption Ability (section 3.2.1.9) store.

If the SEC_ENCRYPT (0x0008) flag is present, then the data following the **securityHeader** field is encrypted and it MUST be verified and decrypted using the methods and techniques described in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol (section 5.4.5) provider.

The remaining PDU fields MUST be interpreted and processed according to the description in section 2.2.1.12. If the **bMsgType** field is not set to ERROR_ALERT (0xFF) then the message is not a License Error PDU (section 2.2.1.12) and the client MAY drop the connection. However, if the client is able to process licensing PDUs, as specified in [MS-RDPELE], it MUST determine if the message is another type of licensing PDU enumerated in [MS-RDPELE] and then if so process it accordingly. If the PDU is a License Error PDU, the client MUST examine the remaining fields and ensure that they conform to the structure and values listed in 2.2.1.12. If this is not the case, the client SHOULD drop the connection.

3.2.5.3.13 Mandatory Capability Negotiation

3.2.5.3.13.1 Processing Demand Active PDU

The structure and fields of the Demand Active PDU are specified in section 2.2.1.13.1.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol (section 5.4.5) being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]), **x224Data** (see [X224]), and **mcsSDin** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDin** is used to route the PDU to the appropriate target channel.

The conditions mandating the presence of the **securityHeader** field, as well as the type of Security Header structure present in this field, are explained in section 2.2.1.13.1. If the **securityHeader** field is present, the embedded **flags** field MUST be examined for the presence of the SEC_ENCRYPT (0x0008) flag (see section 2.2.8.1.1.2.1), and if it is present the data following the **securityHeader** field MUST be verified and decrypted using the methods and techniques described in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol (section 5.4.5) provider.

The **shareControlHeader** field (which contains a Share Control Header as specified in section 2.2.8.1.1.1.1) MUST be examined to ensure that the PDU type (present in the **pduType** field) has the value PDUTYPE_DEMANDACTIVEPDU (1). The server MCS channel ID (present in the **pduSource** field) MUST be stored in the Server Channel ID store. The value of the totalLength field MUST also be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The remaining PDU fields and capability data MUST be interpreted and processed according to sections 2.2.1.13.1.1 and 2.2.7. The capabilities received from the server MUST be stored in the Server Capabilities store and MUST be used to determine what subset of RDP to send to the server. The contents of the **shareId** field MUST be stored in the Share ID store.

After successfully processing the Demand Active PDU, the client MUST send the Confirm Active PDU (section 2.2.1.13.2) to the server.

3.2.5.3.13.2 Sending Confirm Active PDU

The structure and fields of the Confirm Active PDU are specified in section 2.2.1.13.2.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The **mcsSDrq** field is initialized as described in [T125]—the embedded initiator field MUST be set to the MCS user channel ID (held in the User Channel ID store described in section 3.3.1.5) and the embedded **channelId** field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID store described in section 3.3.1.4). The embedded **userData** field contains the remaining fields of the Confirm Active PDU.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario the securityHeader field MUST NOT be present.

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the optional **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level (section 5.3.1) and Encryption Method selected by the server as part of the negotiation specified in section 5.3.2) using the methods and techniques described in 5.3.6. The format of the **securityHeader** field is selected as specified in section 2.2.1.13.2 and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded **flags** field of the **securityHeader** field MUST contain the SEC_ENCRYPT (0x0008) flag.

The remaining fields are populated as described in section 2.2.1.13.2.1, with the concatenated capability set data being inserted into the **capabilitySets** field.

After sending the Confirm Active PDU (section 2.2.1.13.2), the client MUST send the Synchronize PDU (see section 2.2.1.14) to the server.

3.2.5.3.14 Sending Synchronize PDU

The structure and fields of the Synchronize PDU are specified in section 2.2.1.14 and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The **targetUser** field SHOULD be set to the MCS server channel ID (held in the Server Channel ID store). The contents of this PDU are not compressed.

After sending the Synchronize PDU, the client MUST send the Control (Cooperate) PDU (see section 3.2.5.3.15) to the server.

3.2.5.3.15 Sending Control PDU - Cooperate

The structure and fields of the Control (Cooperate) PDU are specified in section 2.2.1.15 and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The **grantId** and **controlId** fields SHOULD be set to zero. The contents of this PDU are not compressed.

After sending the Control (Cooperate) PDU (section 2.2.1.15), the client MUST send the Control (Request Control) PDU (see section 3.2.5.3.16) to the server.

3.2.5.3.16 Sending Control PDU - Request Control

The structure and fields of the Control (Request Control) PDU are specified in section 2.2.1.16, and the techniques described in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The **grantId** and **controlId** fields SHOULD be set to zero. The contents of this PDU are not compressed.

After sending the Control (Request Control) PDU (see section 2.2.1.16), the client MUST send the Persistent Key List PDU (see section 2.2.1.17.1) to the server if the server supports Revision 2 Bitmap Caching (see Bitmap Cache Host Support Capability Set (TS_BITMAPCACHE_HOSTSUPPORT_CAPABILITYSET) (section 2.2.7.1.4) and [MS-RDPEGDI] section 3.1.1.1.1). If the server does not support Revision 2 Bitmap Caching, the client MUST proceed to send the Font List PDU (see section Sending of Font List PDU (section 3.2.5.3.18)).

3.2.5.3.17 Sending Persistent Key List PDU(s)

The structure and fields of the Persistent Key List PDU are specified in section 2.2.1.17, and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

After sending the Persistent Key List PDU, the client MUST send the Font List PDU (see section 3.2.5.3.18) to the server.

3.2.5.3.18 Sending Font List PDU

The structure and fields of the Font List PDU are specified in section 2.2.1.18 and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.3.19 Processing Synchronize PDU

The structure and fields of the Synchronize PDU are specified in section 2.2.1.19 and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU. The contents of the targetUser field MAY be ignored.

3.2.5.3.20 Processing Control PDU - Cooperate

The structure and fields of the Control (Cooperate) PDU are specified in section 2.2.1.20, and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU. The contents of the controlId and grantId fields MAY be ignored.

3.2.5.3.21 Processing Control PDU - Granted Control

The structure and fields of the Control (Granted Control) PDU are specified in section 2.2.1.21, and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU. The contents of the controlId and grantId fields MAY be ignored.

3.2.5.3.22 Processing Font Map PDU

The structure and fields of the Font Map PDU are specified in section 2.2.1.22, and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU. The contents of the **numberEntries**, **totalNumEntries**, **mapFlags** and **entrySize** fields MAY be ignored.

Once the client has successfully processed this PDU, it MAY start to send input PDUs (see section 1.3.5) to the server (see section 2.2.8).

3.2.5.4 Disconnection Sequences

3.2.5.4.1 Sending Shutdown Request PDU

The structure and fields of the Shutdown Request PDU are specified in section 2.2.2.2, and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.4.2 Processing Shutdown Request Denied PDU

The structure and fields of the Shutdown Request Denied PDU are specified in section 2.2.2.3 and the techniques described in Processing of a Basic Server-to-Client Slow-Path PDU (see section 3.2.5.2) demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client behavior is implementation dependent. If the client still wants to disconnect, it SHOULD send an MCS Disconnect Provider Ultimatum PDU (see section 3.1.5.1.1) to the server and drop the connection.

3.2.5.5 Deactivation-Reconnection Sequence

3.2.5.5.1 Processing Deactivate All PDU

The structure and fields of the Deactivate All PDU are specified in section 2.2.3.1 and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client MUST disable its graphics and input protocol handlers and prepare either for a capability renegotiation using a Deactivation-Reactivation Sequence (see section 1.3.1.3) or a disconnection.

3.2.5.6 Auto-Reconnect Sequence

3.2.5.6.1 Processing Auto-Reconnect Status PDU

The structure and fields of the Auto-Reconnect Status PDU are specified in section 2.2.4.1 and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client MAY continue with the connection (and prompt the user to manually enter their credentials for the reconnection attempt), or it MAY simply drop the connection (forcing the user to restart the client and reconnect manually). In either case, the auto-reconnect cookie (see section 5.5) MUST no longer be used.

3.2.5.7 Server Error Reporting

3.2.5.7.1 Processing Set Error Info PDU

The structure and fields of the Set Error Info PDU are specified in section 2.2.5.1, and the techniques specified in section 3.2.5.2) demonstrate how to process the contents of the PDU. The Set Error Info PDU is sent as a precursor to a server-side disconnect and informs the client of the reason for the disconnection which will follow. Once this PDU has been processed, the client MUST store the error code so that the reason for the server disconnect which will follow can be accurately reported to the user.

3.2.5.8 Keyboard and Mouse Input

3.2.5.8.1 Input Event Notifications

3.2.5.8.1.1 Sending Slow-Path Input Event PDU

The structure and fields of the Slow-Path Input Event PDU are specified in Slow Path Input Event (TS_INPUT_EVENT) (section 2.2.8.1.1.3.1), and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU.

The **slowPathInputEvents** field encapsulates a collection of input events and is populated with the following input event data:

- Keyboard Event (TS_KEYBOARD_EVENT) (section 2.2.8.1.1.3.1.1)
- Unicode Keyboard Event (TS_UNICODE_KEYBOARD_EVENT) (section 2.2.8.1.1.3.1.2)
- Mouse Event (TS_POINTER_EVENT) (section 2.2.8.1.1.3.1.3)
- Extended Mouse Event (TS_POINTERX_EVENT) (section 2.2.8.1.1.3.1.4)
- Synchronize Event (TS_SYNC_EVENT) (section 2.2.8.1.1.3.1.5)

The contents of this PDU are not compressed.

If the client has sent a Synchronize Event (section 2.2.8.1.1.3.1.5), it SHOULD subsequently send key-down events for whatever keyboard and mouse keys may be down.

3.2.5.8.1.2 Sending Fast-Path Input Event PDU

The Fast-Path Input Event PDU (section 2.2.8.1.2) has the following basic structure (see sections 5.3.8 and 5.4.4):

- fpInputHeader: Fast-Path Input Header (Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2))
- length1 and length2: Packet Length (Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2))
- fipsInformation: Optional FIPS Information (Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2))
- dataSignature: Optional Data Signature (Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2))

- numberEvents: Optional Number of Events (Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2))
- Actual PDU Contents (collection of Fast-Path input events)
  - Keyboard Event (Fast-Path Keyboard Event (TS_FP_KEYBOARD_EVENT) (section 2.2.8.1.2.2.1))
  - Unicode Keyboard Event (Fast-Path Unicode Keyboard Event (TS_FP_UNICODE_KEYBOARD_EVENT) (section 2.2.8.1.2.2.2))
  - Mouse Event (Fast-Path Mouse Event (TS_FP_POINTER_EVENT) (section 2.2.8.1.2.2.3))
  - Extended Mouse Event (Fast-Path Extended Mouse Event (TS_FP_POINTERX_EVENT) (section 2.2.8.1.2.2.4))
  - Synchronize Event (Fast-Path Synchronize Event (TS_FP_SYNC_EVENT) (section 2.2.8.1.2.2.5))

The **fpInputHeader**, **length1**, **length2** and **numberEvents** fields MUST be initialized as described in Client Fast-Path Input Event PDU (TS_FP_INPUT_PDU) (section 2.2.8.1.2). Since the PDU is in Fast-Path format, the embedded **actionCode** field of the **fpInputHeader** field MUST be set to FASTPATH_INPUT_ACTION_FASTPATH (0).

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario the fipsInformation and **dataSignature** fields MUST NOT be present.

If Standard RDP Security (section 5.3) mechanisms are in effect, the PDU data following the optional **dataSignature** field may be encrypted and signed (depending on the values of the Encryption Level (section 5.3.1) and Encryption Method selected by the server as part of the negotiation described in section 5.3.2, using the methods and techniques described in section 5.3.6. If the data MUST be encrypted, the embedded **encryptionFlags** field of the **fpInputHeader** field MUST contain the FASTPATH_INPUT_ENCRYPTED (2) flag.

The actual PDU contents, which encapsulates a collection of input events, is populated with Fast-Path event data as described from Fast-Path Keyboard Event (TS_FP_KEYBOARD_EVENT) (section 2.2.8.1.2.2.1) to Fast-Path Synchronize Event (TS_FP_SYNC_EVENT) (section 2.2.8.1.2.2.5).

3.2.5.8.2 Keyboard Status PDUs

3.2.5.8.2.1 Processing Set Keyboard Indicators PDU

The structure and fields of the Set Keyboard Indicators PDU are specified in section 2.2.8.2.1 and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client SHOULD update the local keyboard indictors.

3.2.5.8.2.2 Processing Set Keyboard IME Status PDU

The structure and fields of the Set Keyboard IME Status PDU are specified in section 2.2.8.2.2, and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client SHOULD update the state of the local Input Method Editor (IME). Non-IME aware clients MAY ignore this PDU.

3.2.5.9 Basic Output

3.2.5.9.1 Processing Slow-Path Graphics Update PDU

The structure and fields of the Slow-Path Graphics Update PDU are specified in section 2.2.9.1.1.3, and the techniques specified in section 3.2.5.2 demonstrate how to process the contents of the PDU.

The **slowPathGraphicsUpdate** field contains a single graphics update structure which can be one of the following types:

- Orders Update (see [MS-RDPEGDI] section 2.2.2.2)
- Palette Update (see section 2.2.9.1.1.3.1.1)
- Bitmap Update (see section 2.2.9.1.1.3.1.2)
- Synchronize Update (see section 2.2.9.1.1.3.1.3)

If a Slow-Path update structure is received which does not match one of the known types, the client SHOULD ignore the data in the update.

Once this PDU has been processed, the client MUST carry out any operations necessary to complete the update. In the case of a Palette Update (section 2.2.9.1.1.3.1.1), the client MUST update the global palette on all drawing surfaces. Processing of the Bitmap Update requires that the client render the attached bitmap data on the primary drawing surface as specified by the update parameters. The Synchronize Update (section 2.2.9.1.1.3.1.3) MAY be ignored by the client. Processing of the Orders Update (which contains Optimized RDP Drawing Orders) is specified in [MS-RDPEGDI] section 3.2.5.

3.2.5.9.2 Processing Slow-Path Pointer Update PDU

The structure and fields of the Slow-Path Pointer Update PDU are specified in Server Pointer Update PDU (TS_POINTER_PDU) (section 2.2.9.1.1.4), and the techniques specified in section 3.2.5.9.2 demonstrate how to process the contents of the PDU.

The **messageType** field contains an identifier that describes the type of Pointer Update data (see Server Pointer Update PDU (TS_POINTER_PDU) (section 2.2.9.1.1.4) for a list of possible values) present in the **pointerAttributeData** field:

- Pointer Position Update (see Pointer Position Update (TS_POINTERPOSATTRIBUTE) (section 2.2.9.1.1.4.2))
- System Pointer Update (see System Pointer Update (TS_SYSTEMPOINTERATTRIBUTE) (section 2.2.9.1.1.4.3))
- Color Pointer Update (see Color Pointer Update (TS_COLORPOINTERATTRIBUTE) (section 2.2.9.1.1.4.4))
- New Pointer Update (see New Pointer Update (TS_POINTERATTRIBUTE) (section 2.2.9.1.1.4.5))
- Cached Pointer Update (see Cached Pointer Update (TS_CACHEDPOINTERATTRIBUTE) (section 2.2.9.1.1.4.6))

If a Slow-Path update structure is received which does not match one of the known types, the client SHOULD ignore the data in the update.

The contents of this PDU are not compressed.

Once this PDU has been processed, the client MUST carry out any operations necessary to update the local pointer position (in the case of the Position Update) or change the shape (in the case of the System, Color, New and Cached Pointer Updates (section 2.2.9.1.1.4.6)). In the case of the Color and New Pointer Updates the new pointer image MUST also be stored in the **Pointer Image Cache** (see Pointer Image Cache (section 3.2.1.10)), in the slot specified by the **cacheIndex** field.

This necessary step ensures that the client is able to correct process future Cached Pointer Updates (section 2.2.9.1.1.4.6).

3.2.5.9.3 Processing Fast-Path Update PDU

The Fast-Path Update PDU has the following basic structure (see sections 5.3.8 and 5.4.4):

- fpOutputHeader: Fast-Path Output Header (see Server Fast-Path Update PDU (TS_FP_UPDATE_PDU) (section 2.2.9.1.2))
- length1 and length2: Packet Length (see Server Fast-Path Update PDU (TS_FP_UPDATE_PDU) (section 2.2.9.1.2))
- fipsInformation: Optional FIPS Information (see Server Fast-Path Update PDU (TS_FP_UPDATE_PDU) (section 2.2.9.1.2))
- dataSignature: Optional Data Signature (see Server Fast-Path Update PDU (TS_FP_UPDATE_PDU) (section 2.2.9.1.2))
- Actual PDU Contents (collection of Fast-Path output updates)
  - Orders Update (see [MS-RDPEGDI] section 2.2.2.3)
  - Palette Update (see Fast-Path Palette Update (TS_FP_UPDATE_PALETTE) (section 2.2.9.1.2.1.1))
  - Bitmap Update (see Fast-Path Bitmap Update (TS_FP_UPDATE_BITMAP) (section 2.2.9.1.2.1.2))
  - Synchronize Update (see Fast-Path Synchronize Update (TS_FP_UPDATE_SYNCHRONIZE) (section 2.2.9.1.2.1.3))
  - Pointer Position Update (see Fast-Path Pointer Position Update (TS_POINTERPOSATTRIBUTE) (section 2.2.9.1.2.1.4))
  - System Pointer Hidden Update (see Fast-Path System Pointer Hidden Update (TS_FP_SYSTEMPOINTERHIDDENATTRIBUTE) (section 2.2.9.1.2.1.5))
  - System Pointer Default Update (see Fast-Path System Pointer Default Update (TS_FP_SYSTEMPOINTERDEFAULTATTRIBUTE) (section 2.2.9.1.2.1.6))
  - Color Pointer Update (see Fast-Path Color Pointer Update (TS_FP_COLORPOINTERATTRIBUTE) (section 2.2.9.1.2.1.7))
  - New Pointer Update (see Fast-Path New Pointer Update (TS_FP_POINTERATTRIBUTE) (section 2.2.9.1.2.1.8))
  - Cached Pointer Update (see Fast-Path Cached Pointer Update (section 2.2.9.1.2.1.9))

If Enhanced RDP Security (section 5.4)) is in effect, the External Security Protocol being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The contents of the embedded **actionCode** field of the **fpOutputHeader** field MUST be set to FASTPATH_OUTPUT_ACTION_FASTPATH (0). If it is not set to this value, the PDU is not a Fast-Path Update PDU and MUST be processed as a Slow-Path PDU (see section 3.2.5.2).

If the embedded **encryptionFlags** field of the fpOutputHeader field contains the FASTPATH_OUTPUT_ENCRYPTED (2) flag, then the data following the optional dataSignature field (which in this case MUST be present) MUST be verified and decrypted using the methods and techniques described in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the FASTPATH_OUTPUT_ENCRYPTED (2) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol (section 5.4.5) provider.

The update structures present in the **fpOutputUpdates** field MUST be interpreted and processed according to the descriptions detailed from Fast-Path Palette Update (TS_FP_UPDATE_PALETTE) (section 2.2.9.1.2.1.1) to Fast-Path Cached Pointer Update (section 2.2.9.1.2.1.9). The contents of each individual update MAY have been compressed by the server. If this is the case, the embedded compression field of the common **updateHeader** field will contain the FASTPATH_OUTPUT_COMPRESSION_USED flag and the optional compressionFlags field will be be initialized with the compression usage information. Once this PDU has been processed, the client MUST carry out the operation appropriate to the update type, as specified in the Slow-Path versions of this PDU (see sections 3.2.5.9.1 and 3.2.5.9.2).

3.2.5.9.4 Sound

3.2.5.9.4.1 Processing Play Sound PDU

The structure and fields of the Play Sound PDU are specified in 2.2.9.1.1.5, and the techniques specified in 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client SHOULD play a sound using the frequency and duration specified by the PDU.

3.2.5.9.5 Connection Management

3.2.5.9.5.1 Processing Save Session Info PDU

The structure and fields of the Save Session Info PDU are specified in 2.2.10.1 and the techniques specified in 3.2.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the client SHOULD respond to the type of data contained in the PDU:

1. In the case of a logon notification being present in the PDU, the client MAY carry out some implementation-dependent action and if wanted, save the new user name and domain (if received) which were used to log on.

2. In the case of an auto-reconnect cookie being received in the PDU, the client SHOULD save the cookie in the Automatic Reconnection Cookie (section 3.2.1.8) store for possible use during an automatic reconnection sequence.

3. In the case of a logon error or warning notification being present in the PDU, the client SHOULD carry out some implementation-dependent action to respond to the notification.

3.2.5.10 Controlling Server Graphics Output

3.2.5.10.1 Sending Refresh Rect PDU

The structure and fields of the Refresh Rect PDU are specified in section 2.2.11.2, and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.10.2 Sending Suppress Output PDU

The structure and fields of the Suppress Output PDU are specified in section 2.2.11.3, and the techniques specified in section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.6 Timer Events

No client timer events are used.

3.2.6.1 Connection Sequence Timeout

If the RDP Connection Sequence (see sections 1.3.1.1 and 1.3.1.2) does not complete within 300 seconds, the client MAY terminate the connection to the server.

3.2.7 Other Local Events

No additional events are used.

3.2.7.1 Disconnection Due to Network Error

If the client detects that a disconnection which has taken place is due to a network error, it MAY attempt to automatically reconnect to the server using the technique specified in section 5.5. Automatic reconnection allows the client to reconnect to an existing session (after a short-term network failure has occurred) without having to resend the user's credentials to the server.

3.3 Server Details

3.3.1 Abstract Data Model

3.3.1.1 Received Client Data

The Received Client Data store contains data received from the client during execution of the Remote Desktop Protocol (section). This store is initialized when processing the MCS Connect Initial PDU with GCC Conference Create Request (see sections 2.2.1.3 and 3.3.5.3.3) and Client Info PDU (see sections 2.2.1.11 and 3.3.5.3.11).

3.3.1.2 User Channel ID

The User Channel ID store contains the MCS channel identifier allocated by the server to identify the user channel.

3.3.1.3 I/O Channel ID

The I/O Channel ID store contains the MCS channel identifier selected by the server to identify the I/O channel. This ID is communicated to the client in the Server Network Data (see sections 2.2.1.4.4 and 3.2.5.3.4).

3.3.1.4 Server Channel ID

The Server Channel ID store contains the MCS channel identifier of the server channel. In Microsoft RDP server implementations, this value is always 0x03EA.

3.3.1.5 Client Licensing Encryption Ability

The Client Licensing Encryption Ability store determines whether the client has the ability to handle encrypted licensing packets when using Standard RDP Security mechanisms (see the discussion of the SEC_LICENSE_ENCRYPT_SC flag in section 2.2.8.1.1.2.1). This fact is communicated to the server as part of the Security Exchange PDU (see sections 2.2.1.10 and 3.2.5.3.10).

3.3.1.6 Client Capabilities

The Client Capabilities store contains capability sets (see sections 1.4 and 2.2.6) received from the client in the Confirm Active PDU (see sections 2.2.1.13.2 and 3.3.5.3.13.2). The server MUST ensure that it does not violate any of the client capabilities when sending data to the client - for example, if the client does not support Fast-Path output (see section 2.2.7.1.1), the server MUST only send Slow-Path output PDUs . In effect, the server MUST ensure that all of the RDP traffic which it sends on the wire is consistent with the expectations of the client as described by the data held in the Client Capabilities Store.

3.3.1.7 Persistent Bitmap Keys

The Persistent Bitmap Keys store holds a collection of 64-bit bitmap keys, each of which uniquely identifies a bitmap image which the was sent to client using a Revision 2 Cache Bitmap (Revision 2) Secondary Drawing Order (see [MS-RDPEGDI] section 2.2.2.3.1.2.3). When the server sends a bitmap to the client it can first check the Persistent Bitmap Keys store to determine whether the client already has the bitmap in a local bitmap cache (for more details about the Persistent Bitmap Cache, see [MS-RDPEGDI] section 3.1.1.1.1) and hence save on bandwidth.

3.3.1.8 Pointer Image Cache

The Pointer Image Cache contain a collection of pointer images sent to the client in Color Pointer Updates (see sections 2.2.9.1.2.1.7, 3.3.5.9.2, and 3.3.5.9.3) and New Pointer Updates (see sections 2.2.9.1.2.1.8, 3.3.5.9.2, and 3.3.5.9.3). The client MUST maintain an identical cache and keep it in sync with the server cache. When the server needs to instruct the client to update the pointer shape to one already in the cache it sends the client a Cached Pointer Update (see sections 2.2.9.1.1.4.6, 3.3.5.9.2, and 3.3.5.9.3), hence saving bandwidth which would have been used to resend the image. The size and color depth (either variable or fixed at 24 bits-per-pixel) of the cache is negotiated in the Pointer Capability Set (see section 2.2.7.1.6).

3.3.2 Timers

3.3.2.1 Auto-Reconnect Cookie Update Timer

The Auto-Reconnect Cookie Update Timer fires at hourly intervals and triggers the creation of an Auto-Reconnect cookie (see Section 5.5). This cookie is effectively a 16-byte cryptographically secure random number contained within a Server Auto-Reconnect Packet (see section 2.2.4.2) and is sent to the client using the Save Session Info PDU (see section 2.2.10.1).

3.3.3 Initialization

No server initialization steps are specified.

3.3.4 Higher-Layer Triggered Events

No server higher-layer triggered events are used.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Constructing a Basic Server-to-Client Slow-Path PDU

The majority of server-to-client Slow-Path PDUs have the same basic structure (see sections 5.3.7.2 and 5.4.4):

- **tpktHeader**: TPKT Header (see [T123] section 8)
- **x224Data**: X.224 Data TPDU (see [X224] section 13.7)
  - **mcsSDrq**: MCS Send Data Indication PDU (see [ITU T125] section 7, Part 7)
    - **securityHeader**: Optional Security Header (see section 2.2.9.1.1.2)
    - **shareDataHeader**: Share Data Header (see section 2.2.8.1.1.1.2)
    - Actual PDU Contents (see section 2.2)

The PDUs conforming to this basic structure MAY be constructed using the same techniques.

The **tpktHeader** field is initialized as specified in [T123], while the **x224Data** field is initialized as specified in [X224].

The **mcsSDin** field is initialized as specified in [T125]. The embedded initiator field MUST be set to the MCS server channel ID (held in the Server Channel ID (section 3.2.1.5) store) and the embedded channelId field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID (section 3.2.1.3) store). The embedded userData field contains the remaining fields of the PDU.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol (section 5.4.5) MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario, the securityHeader field MUST NOT be present.

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the optional **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level (section 5.3.1) and Encryption Method selected by the server as part of the negotiation described using the methods and techniques specified in 5.3.6. The format of the **securityHeader** field is selected as described in the section detailing the PDU structure and fields (see section 2.2) and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded flags field of the **securityHeader** field MUST contain the SEC_ENCRYPT (0x0008) flag.

The **shareDataHeader** field contains a Share Data Header structure as described in section 2.2.8.1.1.1.2. The **pduSource** of the embedded Share Control Header (section 2.2.8.1.1.1.1) MUST be set to the MCS server channel ID (held in the Server Channel ID (section 3.2.1.5) store). If the contents of the PDU are to be compressed (this MUST be done before any MAC signature is constructed and encryption methods applied), the embedded **compressedType** field of the shareDataHeader MUST be initialized as described in section 2.2.8.1.1.1.2. The remaining Share Data Header and Share Control Header fields MUST be populated as specified in sections 2.2.8.1.1.1.2 and 2.2.8.1.1.1.1 and the section specifying the PDU structure and fields (see section 2.2).

The remaining fields are populated as specified in the section detailing the PDU structure and fields (see section 2.2).

3.3.5.2 Processing a Basic Client-to-Server Slow-Path PDU

The majority of client-to-server Slow-Path PDUs have the same basic structure (see sections 5.3.8 and 5.3.8):

- tpktHeader: TPKT Header (see [T123] section 8)
- x224Data: X.224 Data TPDU (see [X224] section 13.7)
    - mcsSDrq: MCS Send Data Request PDU (see [ITU T125] section 7, part 7)
        - securityHeader: Optional Security Header (see section 2.2.8.1.1.2)
        - shareDataHeader: Share Data Header (see section 2.2.8.1.1.1.2)
        - Actual PDU Contents (see section 2.2)

The PDUs conforming to this basic structure MAY be processed using the same techniques.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]), x224Data (see [X224]), and mcsSDrq (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the mcsSDrq is used to route the PDU to the appropriate target channel.

The conditions mandating the presence of the securityHeader field, as well as the type of Security Header structure present in this field, are explained in the Section specifying the PDU structure and fields (see 2.2). If the securityHeader field is present, the embedded flags field MUST be examined for the presence of the SEC_ENCRYPT (0x0008) flag (see section 2.2.8.1.1.2.1), and if it is present the data following the securityHeader field MUST be verified and decrypted using the methods and techniques specified in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol Provider.

The conditions mandating the presence of the **securityHeader** field, as well as the type of Security Header structure present in this field, are explained in the Section specifying the PDU structure and fields (see section 2.2). If the **securityHeader** field is present, the embedded flags field MUST be examined for the presence of the SEC_ENCRYPT (0x0008) flag (see section 2.2.8.1.1.2.1), and if it is present the data following the **securityHeader** field MUST be verified and decrypted using the methods and techniques specified in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol Provider.

The remaining PDU fields (if any) MUST be interpreted and processed according to the section specifying the PDU structure and fields (see section 2.2).

3.3.5.3 Normal Connection Sequence

3.3.5.3.1 Processing X.224 Connection Request PDU

The structure and fields of the X.224 Connection Request PDU are specified in section 2.2.1.1.

The embedded length fields within the tpktHeader (see [T123]) and **x224Crq** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped. Other triggers for dropping the connection include:

- The length of the X.224 Connection Request PDU is less than 11 bytes.
- The X.224 Connection Request PDU is not Class 0 (see [X224].

The Destination reference, Source reference and Class and options fields within the **x224Crq** field are ignored.

If the optional **routingToken** field exists, it SHOULD be ignored (since the routing token is actually parsed by external hardware components along the connection path).

If the **rdpNegData** field is not present, it is assumed that the client does not support Enhanced RDP Security (see section 5.4) and negotiation data MUST NOT be sent to the client as part of the X.224 Connection Confirm PDU (see 3.3.5.3.2). If the **rdpNegData** field is present, it is parsed to check that it contains a valid RDP Negotiation Request structure, as specified in section 2.2.1.1.1. If this is not the case, the connection SHOULD be dropped. If the structure is valid, the flags describing the supported security protocols in the **requestedProtocols** field are saved in the Received Client Data (section 3.3.1.1) store.

Once the MCS X.224 Connection Request PDU has been processed successfully, the server MUST send the X.224 Connection Confirm PDU (see 3.3.5.3.2) to the client.

3.3.5.3.2 Sending X.224 Connection Confirm PDU

The structure and fields of the X.224 Connection Confirm PDU are specified in section 2.2.1.2.

The tpktHeader field is initialized as specified in [T123], while the x224Ccf field is initialized as detailed in [X224] (the Destination reference is set to zero, the Source reference is set to 0x1234, and the Class and options are set to zero). Parameter fields MUST NOT be specified in the variable part of the Connection Response PDU.

The **rdpNegData** field is left empty if the client did not append any negotiation data to the X.224 Connection Request PDU (see 3.3.5.3.1). If the client did append negotiation data to the X.224 Connection Request PDU, the **rdpNegData** field SHOULD contain an RDP Negotiation Response (see section 2.2.1.2.1) or RDP Negotiation Failure (see section 2.2.1.2.2) structure.

The RDP Negotiation Response structure is sent if the server supports (and is configured to use) one of the client-requested security protocols specified in the X.224 Connection Request PDU and saved in the Received Client Data store. The **selectedProtocol** field is initialized with the selected protocol identifier (see section 2.2.1.2.2 for a list of identifiers). If the server decides to use Standard RDP Security mechanisms, it MUST set the **selectedProtocol** field to PROTOCOL_RDP (0x00000000).

The RDP Negotiation Failure structure is sent if it is not possible to continue the connection with any of the client-requested External Security Protocols. The possible failure codes along with a reason for sending each of them are listed in section 2.2.1.2.2. After sending the RDP Negotiation Failure Structure the server MAY close the connection.

If an External Security Protocol, such as TLS (see section 5.4.5.1) or CredSSP (see section 5.4.5.2), will be used for the duration of the connection, the server MUST prepare to execute the selected protocol by calling into the relevant External Security Protocol Provider after the X.224 Connection Confirm PDU (with RDP Negotiation Response) has been sent to the client.

3.3.5.3.3 Processing MCS Connect Initial PDU with GCC Conference Create Request

The structure and fields of the MCS Connect Initial PDU with GCC Conference Create Request are specified in section 2.2.1.3. A basic high-level overview of the nested structure for the MCS Connect Initial PDU is illustrated in Figure 2.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Connect Initial PDU (embedded within the **mcsCi** field) is specified in detail in [T125]. The server ignores the **calledDomainSelector**, **callingDomainSelector** and **upwardFlag** fields of this PDU.

The domain parameters (contained in the **targetParameters**, **minimumParameters** and **maximumParameters** fields) received in the MCS Connect Initial PDU are examined and the resultant parameters determined. The table which follows summarizes the rules employed by the server when negotiating the domain parameters. If the server is unable to satisfy a Negotiation Rule, then the connection SHOULD be dropped.

Domain parameter	Negotiation rule
maxChannelIds	MUST be able to negotiate a value of at least 4.
maxUserIds	MUST be able to negotiate a value of at least 3.
maxTokenIds	SHOULD use client target value.
numPriorities	MUST be able to negotiate a value of 1.
minThroughput	SHOULD use client target value.
maxHeight	MUST be able to negotiate a value of 1.
maxMCSPDUsize	MUST be able to negotiate a value between 123 and 65529.
protocolVersion	MUST be able to negotiate a value of 2.

The **userData** field of the MCS Connect Initial PDU contains the GCC Conference Create Request (embedded within the **gccCCrq** field). The GCC Conference Create Request is described in detail in [T124] and appended as user data to the MCS Connect Initial PDU using the format specified in [T124] sections 9.5 and 9.6.

If the size of the GCC Conference Create Request data is larger than 1024 bytes, then the server MUST send an MCS Connect Response PDU (see [T125]) to the client containing only a **result** field set to the value rt-unspecified-failure (14).

If the size of the GCC Conference Create Request data is smaller than 1024 bytes, processing can continue. The server MAY ignore all of the GCC Conference Create Request fields, except for the **userData** field. The **userData** field of the GCC Conference Create Request MUST contain basic client settings data blocks (see sections 2.2.1.3.2 through to 2.2.1.3.5). The server MUST check that the client-to-server H.221 key embedded at the start of the **userData** field (see [T124] section 8.7 for a description of the structure of user data) is the ANSI text "Duca". If this is not the case, the connection SHOULD be dropped.

All of the encoded lengths within the MCS Connect-Initial PDU and the GCC Conference Create Request MUST also be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

Once the **mcsCi** and **gccCCrq** fields have been successfully parsed the server examines the basic client settings data blocks in the GCC Conference Create Request user data and stores this data in the Received Client Data (section 3.3.1.1) store. However, before the data is stored, the basic client settings data blocks are checked for validity.

Select settings in the Client Core Data (see section 2.2.1.3.2) are validated using the following rules:

Client core data field	Validation rule
desktopWidth	If this field contains a width greater than 4096 pixels, a value of exactly 4096 pixels is implicitly assumed.
desktopHeight	If this field contains a width greater than 2048 pixels, a value of exactly 2048 pixels is implicitly assumed.
colorDepth	If this field does not contain a valid value, the server SHOULD drop the connection.
postBeta2ColorDepth	If this field does not contain a valid value, the server SHOULD drop the connection.
highColorDepth	If this field does not contain a valid color-depth, a value of 8 bits-per-pixel is implicitly assumed.
serverSelectedProtocol	If this field does not contain the same value that the server transmitted to the client in the RDP Negotiation Response (see 3.3.5.3.2), the server SHOULD drop the connection. In the event that this optional field is not present, the value PROTOCOL_RDP (0) MUST be assumed.

The **encryptionMethods** and **extEncryptionMethods** fields in the Client Security Data (see section 2.2.1.3.3) are examined to ensure that they contain at least one valid flag. If no valid flags are present, the connection SHOULD be dropped.

If the Client Network Data (see section 2.2.1.3.4) is included in the Settings Data, the server MUST check that the **channelCount** field is within bounds. Furthermore, the data supplied in the **channelDefArray** MUST be complete. If these two conditions are not met the connection SHOULD be dropped.

Once the basic client settings data blocks have been processed successfully, the server MUST send the MCS Connect Response PDU with GCC Conference Create Response (see 3.3.5.3.4) to the client.

3.3.5.3.4 Sending MCS Connect Response PDU with GCC Conference Create Response

The structure and fields of the MCS Connect Response PDU with GCC Conference Create Response are described in section 2.2.1.4. A basic high-level overview of the nested structure for the MCS Connect Initial PDU is illustrated in Figure 2.

The **tpktHeader** field is initialized as described in [T123], while the **x224Data** field is initialized as detailed in [X224].

The MCS Connect Response PDU (embedded within the **mcsCrsp** field) is described in detail in [T125]. The fact that the MCS Connect Response PDU will contain a GCC Conference Create Response as user data implies that processing of the MCS Connect Initial PDU with GCC Conference Create Request (see 3.3.5.3.3) was successful, and hence the server MUST set the **result** field of the MCS Connect Response PDU to rt-successful (0). The **calledConnectId** field SHOULD be set to zero, while the **domainParameters** field MUST be initialized with the parameters which were derived from processing of the MCS Connect Initial PDU (see 3.3.5.3.3 for a description of the negotiation rules).

The **userData** field of the MCS Connect Response PDU contains the GCC Conference Create Response (embedded within the **gccCCrsp** field). The GCC Conference Create Response is described in detail in [T124] and appended as user data to the MCS Connect Response PDU using the format described in sections 9.5 and 9.6 of [T124]. The server SHOULD initialize the fields of the GCC Conference Create Response as follows:

Conference Create Response Field	Value
nodeID	31219
tag	1 (length of 1 byte)
result	success (0)
userData	Basic Server Settings Data Blocks

The **userData** field of the GCC Conference Create Response MUST be initialized with basic server settings data blocks (see sections 2.2.1.4.2 through to 2.2.1.4.4). The server-to-client H.221 key which MUST be embedded at the start of the **userData** field (see section 8.7 of [T124] for a description of the structure of user data) is the ANSI text "McDn".

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.3.5.3.5 Processing MCS Erect Domain Request PDU

The structure and fields of the MCS Erect Domain Request PDU are described in section 2.2.1.6.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Erect Domain Request PDU (embedded within the **mcsEDrq** field) is described in detail in [T125]. The server MUST ensure sure that the **subHeight** and **subinterval** fields are contained within the PDU. If this is not the case, the connection SHOULD be dropped.

3.3.5.3.6 Processing MCS Attach User Request PDU

The structure and fields of the MCS Attach User Request PDU are described in section 2.2.1.6.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the tpktHeader (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Attach User Request PDU (embedded within the **mcsAUrq** field) is described in detail in [T125].

Upon the receiving the MCS Attach User Request PDU the server MUST send the MCS Attach User Confirm PDU (see 3.3.5.3.7) to the client.

3.3.5.3.7 Sending MCS Attach User Confirm PDU

The structure and fields of the MCS Attach User Confirm PDU are described in section 2.2.1.7.

The tpktHeader field is initialized as described in [T123], while the **x224Data** field is initialized as detailed in [X224].

The MCS Connect Response PDU (embedded within the **mcsCrsp** field) is described in detail in [T125]. The optional initiator field MUST be present and MUST contain an integer identifier selected by the server to identify the user channel (this identifier should be stored in the User Channel ID (section 3.3.1.2) store). If a channel identifier could not be generated the **result** field MUST be set to rt-unspecified-failure (14) and the **initiator** field SHOULD NOT be present. If a channel identifier could be generated, the **result** field MUST be set to rt-successful (0).

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.3.5.3.8 Processing MCS Channel Join Request PDU(s)

The structure and fields of the MCS Channel Join Request PDU are described in section 2.2.1.8.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]) and **x224Data** (see [X224]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The MCS Channel Join Request PDU (embedded within the **mcsCJrq** field) is described in detail in [T125].

Upon the receiving the MCS Channel Join Request PDU the server can carry out any necessary processing to mark the channel as "joined" and MUST then send the MCS Channel Join Confirm PDU (see 3.3.5.3.9) to the client to indicate the result of the join operation.

3.3.5.3.9 Sending MCS Channel Join Confirm PDU(s)

The structure and fields of the MCS Channel Join Confirm PDU are described in section 2.2.1.9.

The **tpktHeader** field is initialized as described in [T123], while the x224Data field is initialized as detailed in [X224].

The MCS Channel Join Confirm PDU (embedded within the **mcsCJcf** field) is described in detail in [T125]. The **result** field MUST be set to rt-successful (0) if the MCS channel ID in the corresponding MCS Channel Join Request PDU (see 3.3.5.3.8) was successfully joined. If an error occurred during the join (for example, too many channels, no such MCS channel ID, memory allocation error, etc.), the server MUST set the **result** field to rt-unspecified-failure (14). The remaining fields MUST be initialized as follows (these fields are essentially copied over from the MCS Channel Join Request PDU):

Channel Join Confirm Field	Value
initiator	The initiator value which was sent in the corresponding MCS Channel Join Request PDU.
requested	The MCS channel ID which was sent in the corresponding MCS Channel Join Request PDU.
channelId	The MCS channel ID which was sent in the corresponding MCS Channel Join Request PDU.

The optional **channelId** field MUST be included in the MCS Channel Join Confirm PDU sent to the client.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire.

3.3.5.3.10 Processing Security Exchange PDU

The structure and fields of the Security Exchange PDU are described in section 2.2.1.10.

The embedded length fields within the **tpktHeader** (see [T123]), x224Data (see [X224]), and **mcsSDrq** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDrq** is used to route the PDU to the appropriate target channel.

The embedded flags field of the basicSecurityHeader MUST contain the SEC_EXCHANGE_PKT (0x0001) flag (described in section 2.2.8.1.1.2.1). If this flag is not present then the packet cannot be interpreted as a Security Exchange PDU. If the SEC_LICENSE_ENCRYPT_SC (0x0200) flag is present, then the client is able to accept encrypted licensing packets when using Standard RDP Security mechanisms. This fact is stored in the Client Licensing Encryption Ability (section 3.3.1.5) store.

The encrypted client random value is extracted from the **encryptedClientRandom** field using the **length** field to determine the size of the data. If the value of the **length** field is inconsistent with the size of the received data, the connection SHOULD be dropped. The encrypted client random value is then decrypted using the methods and techniques described in section 5.3.4.2.

Once the server has extracted and decrypted the client random it MUST generate the session keys which will be used to encrypt, decrypt and sign data sent on the wire. The 32-byte client random and server random (transmitted in the Server Security Data described in section 2.2.1.4.3), are used to accomplish this task by employing the techniques described in section 5.3.5.

3.3.5.3.11 Processing Client Info PDU

The structure and fields of the Client Info PDU are specified in section 2.2.1.11.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]), x224Data (see [X224]), and **mcsSDrq** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDrq** is used to route the PDU to the appropriate target channel.

The **securityHeader** field MUST always be present and it MUST contain at least a Basic Security Header structure (see section 2.2.8.1.1.2.1). The embedded flags field of the securityHeader MUST contain the SEC_EXCHANGE_PKT (0x0001) flag (described in section 2.2.8.1.1.2.1). If this flag is not present then the packet cannot be interpreted as a Security Exchange PDU. If the SEC_ENCRYPT (0x0008) flag is present, then the data following the **securityHeader** field is encrypted and it MUST be verified and decrypted using the methods and techniques specified in section 5.3.6. (If the Encryption Level selected by the server (see Sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_NONE (0) the SEC_ENCRYPT flag could possibly be set by some Microsoft RDP client versions as a result of a code bug. In this case the Encryption Level setting MUST be respected and the value of the flag MUST be ignored.) If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol Provider.

Before reading the client settings fields, the format of the character data should be determined by testing for the presence of the INFO_UNICODE (0x00000010) flag (see section 2.2.1.11.1.1). If the flag is present, all character data MUST be interpreted as Unicode, otherwise it MUST be treated as ANSI.

All of the received client settings are stored in the Received Client Data (section 3.3.1.1) store. When store character data the server SHOULD only save the maximum allowed sizes specified in section 2.2.1.11.1.1. For example, the maximum specified size for the **AlternateShell** field is 512 bytes. If received data is larger than this size it SHOULD be truncated to 512 bytes in length (including the mandatory null terminator) when it is stored.

If there is not enough received data to completely read a variable-length field, the connection SHOULD be dropped. For example, if the **cbAlternateShell** field contains a value of 44 bytes, but only 30 bytes remain to be parsed, the connection SHOULD be dropped.

If an auto-reconnect cookie exists in the **autoReconnectCookie** field, the server SHOULD store the cookie and use it to log on the user once the connection sequence completes (for a description of how automatic reconnection works, see section 5.5). If logon with the cookie fails, the credentials supplied in the Client Info PDU SHOULD be used, or alternatively the user MAY enter credentials at a server-side prompt remoted using RDP.

Once the server has successfully processed the Client Info PDU, it can enter the Licensing Phase of the RDP Connection Sequence (see section 1.3.1.1) and carry out a licensing exchange with the client.

3.3.5.3.12 Sending License Error PDU - Valid Client

The structure and fields of the License Error (Valid Client) PDU are described in section 2.2.1.12.

The **tpktHeader** field is initialized as described in [T123], while the **x224Data** field is initialized as detailed in [X224].

The **mcsSDin** field is initialized as described in [T125] - the embedded **initiator** field MUST be set to the MCS server channel ID (held in the Server Channel ID (section 3.3.1.4) store) and the embedded **channelId** field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID (section 3.3.1.3) store). The embedded **userData** field contains the remaining fields of the Valid Client PDU.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature). The **securityHeader** field MUST be present, however it will contain a Basic Security Header structure (see section 2.2.8.1.1.2.1).

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level and Encryption Method selected by the server as part of the negotiation described in section 5.3.2 and the contents of the Client Licensing Encryption Ability (section 3.3.1.5) store using the methods and techniques described in section 5.3.6. The format of the **securityHeader** field is selected as described in section 2.2.1.12 and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded **flags** field of the **securityHeader** field MUST contain the SEC_ENCRYPT (0x0008) flag.

The embedded **flags** field of the **securityHeader** field (which is always present) MUST contain the SEC_LICENSE_PKT (0x0080) flag (described in section 2.2.8.1.1.2.1) to indicate that the message is a licensing PDU. If the server can handle encrypted licensing packets from the client and Standard RDP Security mechanisms are being used, then the SEC_LICENSE_ENCRYPT_CS (0x0200) flag SHOULD also be included in the flags subfield of the **securityHeader** field.

The remainder of the PDU MUST be populated according to the structure and type definition in section 2.2.1.12.

After sending the License Error (Valid Client) PDU, the server MUST send the Demand Active PDU (see section 3.3.5.3.13.1) to the client.

3.3.5.3.13 Mandatory Capability Negotiation

3.3.5.3.13.1 Sending Demand Active PDU

The structure and fields of the Demand Active PDU are described in section 2.2.1.13.1.

The **tpktHeader** field is initialized as described in [T123], while the **x224Data** field is initialized as detailed in [X224].

The **mcsSDin** field is initialized as described in [T125] - the embedded initiator field MUST be set to the MCS server channel ID (held in the Server Channel ID (section 3.3.1.4) store) and the embedded channelId field MUST be set to the MCS I/O channel ID (held in the I/O Channel ID (section 3.3.1.3) store). The embedded **userData** field contains the remaining fields of the Demand Active PDU.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario the **securityHeader** field MUST NOT be present.

If Standard RDP Security mechanisms (see section 5.3) are in effect, the PDU data following the optional **securityHeader** field may be encrypted and signed (depending on the values of the Encryption Level and Encryption Method selected by the server as part of the negotiation described in section 5.3.2) using the methods and techniques described in 5.3.6. The format of the **securityHeader** field is selected as described in section 2.2.1.13.1 and the fields populated with appropriate security data. If the data MUST be encrypted, the embedded flags field of the **securityHeader** field MUST contain the SEC_ENCRYPT (0x0008) flag.

The remaining fields are populated as described in section 2.2.1.13.1.1, with the concatenated capability set data being inserted into the **capabilitySets** field.

3.3.5.3.13.2 Processing Confirm Active PDU

The structure and fields of the Confirm Active PDU are described in section 2.2.1.13.2.

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The embedded length fields within the **tpktHeader** (see [T123]), **x224Data** (see [X224]), and **mcsSDrq** (see [T125]) fields MUST be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The embedded **channelId** field within the **mcsSDrq** is used to route the PDU to the appropriate target channel.

The conditions mandating the presence of the **securityHeader** field, as well as the type of Security Header structure present in this field, are explained in section 2.2.1.13.2. If the **securityHeader** field is present, the embedded flags field MUST be examined for the presence of the SEC_ENCRYPT (0x0008) flag (see section 2.2.8.1.1.2.1), and if it is present the data following the **securityHeader** field MUST be verified and decrypted using the methods and techniques described in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the SEC_ENCRYPT (0x0008) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol Provider.

The **shareControlHeader** field (which contains a Share Control Header as described in section 2.2.8.1.1.1.1) MUST be examined to ensure that the PDU type (present in the **pduType** field) has the value PDUTYPE_CONFIRMACTIVEPDU (3). The value of the **totalLength** field MUST also be examined for consistency with the received data. If there is any discrepancy, the connection SHOULD be dropped.

The remaining PDU fields and capability data MUST be interpreted and processed according to sections 2.2.1.13.2.1 and 2.2.7. The capabilities received from the client MUST be stored in the Client Capabilities (section 3.3.1.6) store and MUST be used to determine what subset of RDP to send to the client.

After successfully processing the Confirm Active PDU, the server MUST send the Synchronize PDU (see 3.3.5.3.14) to the client.

3.3.5.3.14 Processing Synchronize PDU

The structure and fields of the Synchronize PDU are described in section 2.2.1.14, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU. The contents of the **targetUser** field MAY be ignored.

3.3.5.3.15 Processing Control PDU - Cooperate

The structure and fields of the Control (Cooperate) PDU are described in section 2.2.1.15, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

3.3.5.3.16 Processing Control PDU - Request Control

The structure and fields of the Control (Request Control) PDU are described in section 2.2.1.16, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

3.3.5.3.17 Processing Persistent Key List PDU(s)

The structure and fields of the Persistent Key List PDU are described in section 2.2.1.17, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU. Note that multiple Persistent Key List PDUs may be sent in succession - the bBitMask flag indicates the sequencing.

Once the server has successfully processed the Persistent Key List PDU, it stores the 64-bit bitmap keys received from the client in the Persistent Bitmap Keys (section 3.3.1.7) store.

3.3.5.3.18 Processing Font List PDU

The structure and fields of the Control (Request Control) PDU are described in section 2.2.1.18, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU. The contents of the **numberFonts**, **totalNumFonts**, **listFlags**, and **entrySize** fields MAY be ignored.

3.3.5.3.19 Sending Synchronize PDU

The structure and fields of the Synchronize PDU are described in section 2.2.1.19, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The targetUser field does not need to be initialized and MAY be set to zero. The contents of this PDU are not compressed.

After sending the Synchronize PDU, the server MUST send the Control (Cooperate) PDU (section 3.3.5.3.20) to the client.

3.3.5.3.20 Sending Control PDU - Cooperate

The structure and fields of the Control (Cooperate) PDU are described in section 2.2.1.20, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The grantId and controlId fields SHOULD be set to zero. The contents of this PDU are not compressed.

After sending the Control (Cooperate) PDU, the server MUST send the Control (Granted Control) PDU (section 3.3.5.3.21) to the client.

3.3.5.3.21 Sending Control PDU - Granted Control

The structure and fields of the Control (Granted Control) PDU are described in section 2.2.1.21, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The **grantId** field SHOULD be set to the MCS user channel ID (held in the User Channel ID (section 3.3.1.2) store), while the **controlId** field SHOULD be set to the MCS server channel ID (held in the Server Channel ID (section 3.3.1.4) store). The contents of this PDU are not compressed.

After sending the Control (Granted Control) PDU, the server MUST send the Font Map PDU (section 3.3.5.3.22) to the client.

3.3.5.3.22 Sending Font Map PDU

The structure and fields of the Font Map PDU are described in section 2.2.1.22, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.4 Disconnection Sequences

3.3.5.4.1 Processing Shutdown Request PDU

The structure and fields of the Shutdown Request PDU are described in section 2.2.2.2, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once the server has successfully processed the Shutdown Request PDU, it MUST send the Shutdown Request Denied PDU (section 3.3.5.4.2) to the client.

3.3.5.4.2 Sending Shutdown Request Denied PDU

The structure and fields of the Shutdown Request Denied PDU are described in section 2.2.2.3, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.5 Deactivation-Reconnection Sequence

3.3.5.5.1 Sending Deactivate All PDU

The structure and fields of the Deactivate All PDU are described in section 2.2.3.1, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

Once the server has successfully transmitted the Deactivate All PDU, it SHOULD send an MCS Disconnect Provider Ultimatum (section 3.1.5.1.1) and then close the connection, or it SHOULD initiate capability renegotiation by using a Deactivation-Reactivation Sequence (see section 1.3.1.3).

3.3.5.6 Auto-Reconnect Sequence

3.3.5.6.1 Sending Auto-Reconnect Status PDU

The structure and fields of the Auto-Reconnect Status PDU are described in section 2.2.4.1, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.7 Server Error Reporting

3.3.5.7.1 Sending Set Error Info PDU

The structure and fields of the Set Error Info PDU are described in section 2.2.5.1, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed. After the PDU has been sent the server MUST disconnect the client (since the Set Error Info PDU has been sent, the client will be aware of the reason for the disconnect).

3.3.5.8 Keyboard and Mouse Input

3.3.5.8.1 Input Event Notifications

3.3.5.8.1.1 Processing Slow-Path Input Event PDU

The structure and fields of the Slow-Path Input Event PDU are described in section 2.2.8.1.1.3, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

The **slowPathInputEvents** field encapsulates a collection of input events and is populated with the following input event data:

- Keyboard Event (section 2.2.8.1.1.3.1.1)
- Unicode Keyboard Event (section 2.2.8.1.1.3.1.2)
- Mouse Event (section 2.2.8.1.1.3.1.3)
- Extended Mouse Event (section 2.2.8.1.1.3.1.4)
- Synchronize Event (section 2.2.8.1.1.3.1.5)

If a Slow-Path input event structure is received that does not match one of the known types, the server SHOULD drop the connection.

Once this PDU has been processed, the server MUST inject the input event into the remote session.

3.3.5.8.1.2 Processing Fast-Path Input Event PDU

The Fast-Path Input Event PDU has the following basic structure (see sections 5.3.8 and 5.4.4):

- **fpInputHeader**: Fast-Path Input Header (see section 2.2.8.1.2)

- **length1** and **length2**: Packet Length (see section 2.2.8.1.2)
- **fipsInformation**: Optional FIPS Information (see section 2.2.8.1.2)
- **dataSignature**: Optional Data Signature (see section 2.2.8.1.2)
- **numberEvents**: Optional Number of Events (see section 2.2.8.1.2)
- Actual PDU Contents (collection of input events)
  - Keyboard Event (section 2.2.8.1.2.2.1)
  - Unicode Keyboard Event (section 2.2.8.1.2.2.2)
  - Mouse Event (section 2.2.8.1.2.2.3)
  - Extended Mouse Event (section 2.2.8.1.2.2.4)
  - Synchronize Event (section 2.2.8.1.2.2.5)

If Enhanced RDP Security (see section 5.4) is in effect, the External Security Protocol being used to secure the connection MUST be used to decrypt and verify the integrity of the entire PDU (possibly by using some sort of MAC signature) prior to any processing taking place.

The contents of the embedded **actionCode** field of the **fpInputHeader** field MUST be set to FASTPATH_INPUT_ACTION_FASTPATH (0). If it is not set to this value the PDU is not a Fast-Path Input Event PDU and MUST be processed as a Slow-Path PDU (see section 3.3.5.2).

If the embedded **encryptionFlags** field of the **fpInputHeader** field contains the FASTPATH_INPUT_ENCRYPTED (2) flag, then the data following the optional **dataSignature** field (which in this case MUST be present) MUST be verified and decrypted using the methods and techniques described in section 5.3.6. If the MAC signature is incorrect or the data cannot be decrypted correctly, the connection SHOULD be dropped. If Enhanced RDP Security is in effect and the FASTPATH_INPUT_ENCRYPTED (2) flag is present the connection SHOULD be dropped, as double-encryption is not used within RDP in the presence of an External Security Protocol Provider.

The **numberEvents** field details the number of input events present in the **fpInputEvents** field. The input events present in this field MUST be interpreted and processed according to the descriptions detailed from sections 2.2.8.1.2.2.1 through 2.2.8.1.2.2.5. If a Fast-Path input event structure is received that does not match one of the known types, the server SHOULD drop the connection.

Once this PDU has been processed, the server MUST inject the input event into the remote session.

3.3.5.8.2 Keyboard Status PDUs

3.3.5.8.2.1 Sending Set Keyboard Indicators PDU

The structure and fields of the Set Keyboard Indicators PDU are described in section 2.2.8.2.1, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.8.2.2 Sending Set Keyboard IME Status PDU

The structure and fields of the Set Keyboard IME Status PDU are described in section 2.2.8.2.2, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.9 Basic Output

3.3.5.9.1 Sending Slow-Path Graphics Update PDU

The structure and fields of the Slow-Path Graphics Update PDU are described in section 2.2.9.1.1.3, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU.

The **slowPathGraphicsUpdate** field contains a single graphics update structure which can be one of the following types:

- Orders Update (see section 2.2.2.2 in [MS-RDPEGDI])
- Palette Update (section 2.2.9.1.1.3.1.1)
- Bitmap Update (section 2.2.9.1.1.3.1.2)
- Synchronize Update (section 2.2.9.1.1.3.1.3)

The contents of this PDU MAY be compressed by the server before any MAC signature is constructed and encryption methods applied. The Share Data Header MUST be initialized with the compression usage information (see section 3.3.5.1).

3.3.5.9.2 Sending Slow-Path Pointer Update PDU

The structure and fields of the Slow-Path Pointer Update PDU are described in section 2.2.9.1.1.4, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU.

The **messageType** field MUST be initialized with the identifier describing the type of the Pointer Update (see section 2.2.9.1.1.4 for a list of possible values), while the **pointerAttributeData** field MUST be initialized with the actual update data contained in one of the following structures:

- Pointer Position Update (section 2.2.9.1.1.4.2)
- System Pointer Update (section 2.2.9.1.1.4.3)
- Color Pointer Update (section 2.2.9.1.1.4.4)
- New Pointer Update (section 2.2.9.1.1.4.5)
- Cached Pointer Update (section 2.2.9.1.1.4.6)

When sending a Color or New Pointer Update, the server MUST save the pointer image in the Pointer Image Cache (section 3.3.1.8) and intialize the **cacheIndex** field with the index of the cache entry which was used.

The contents of this PDU are not compressed.

3.3.5.9.3 Sending Fast-Path Update PDU

The Fast-Path Update PDU has the following basic structure (see sections 5.3.8 and 5.4.4):

- **fpOutputHeader**: Fast-Path Output Header (see section 2.2.9.1.2)
- **length1** and **length2**: Packet Length (see section 2.2.9.1.2)

- **fipsInformation**: Optional FIPS Information (see section 2.2.9.1.2)
- **dataSignature**: Optional Data Signature (see section 2.2.9.1.2)
- Actual PDU Contents (collection of Fast-Path output updates)
  - Orders Update (see section 2.2.2.3 in [MS-RDPEGDI])
  - Palette Update (section 2.2.9.1.2.1.1)
  - Bitmap Update (section 2.2.9.1.2.1.2)
  - Synchronize Update (section 2.2.9.1.2.1.3)
  - Pointer Position Update (section 2.2.9.1.2.1.4)
  - System Pointer Hidden Update (section 2.2.9.1.2.1.5)
  - System Pointer Default Update (section 2.2.9.1.2.1.6)
  - Color Pointer Update (section 2.2.9.1.2.1.7)
  - New Pointer Update (section 2.2.9.1.2.1.8)
  - Cached Pointer Update (section 2.2.9.1.2.1.9)

The **fpOutputHeader**, **length1** and **length2** fields MUST be initialized as described in section 2.2.9.1.2. Since the PDU is in Fast-Path format, the embedded **actionCode** field of the **fpOutputHeader** field MUST be set to FASTPATH_OUTPUT_ACTION_FASTPATH (0).

If Enhanced RDP Security (section 5.4) is in effect, the External Security Protocol MUST be used to encrypt the entire PDU and generate a verification digest (possibly by using some sort of MAC signature) before the PDU is transmitted over the wire. Also, in this scenario the **fipsInformation** and **dataSignature** fields MUST NOT be present.

If Standard RDP Security (section 5.3) mechanisms are in effect, the PDU data following the optional **dataSignature** field may be encrypted and signed (depending on the values of the Encryption Level and Encryption Method selected by the server as part of the negotiation described in section 5.3.2) using the methods and techniques described in section 5.3.6. If the data MUST be encrypted, the embedded **encryptionFlags** field of the **fpOutputHeader** field MUST contain the FASTPATH_OUTPUT_ENCRYPTED (2) flag.

The actual PDU contents, which encapsulates a collection of output events, is populated with Fast-Path update data as described in sections 2.2.9.1.2.1.1 through 2.2.9.1.2.1.9. The contents of each individual update MAY be compressed by the server before any MAC signature is constructed and encryption methods applied. If this is the case, the embedded **compression** field of the common **updateHeader** field MUST contain the FASTPATH_OUTPUT_COMPRESSION_USED flag and the optional **compressionFlags** field MUST be be initialized with the compression usage information.

3.3.5.9.4 Sound

3.3.5.9.4.1 Sending Play Sound PDU

The structure and fields of the Play Sound PDU are described in section 2.2.9.1.1.5, and the techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.9.5 Connection Management

3.3.5.9.5.1 Sending Save Session Info PDU

The structure and fields of the Save Session Info PDU are described in section 2.2.10.1.

The three reasons for sending this PDU are:

1. Notifying the client that the user has logged on (the username and domain which were used, as well as the ID of the session to which the user connected, may be included in this notification).

2. Transmitting an auto-reconnect cookie to the client (see section 1.3.1.5).

3. Informing the client of an error or warning that occurred while the user was logging on.

The client SHOULD always be notified after the user has logged on. The INFOTYPE_LOGON (0x00000000), INFOTYPE_LOGON_LONG (0x00000001), or INFOTYPE_LOGON_PLAINNOTIFY (0x00000002) notification types MUST be used to accomplish this task.

A logon notification of type INFOTYPE_LOGON or INFOTYPE_LOGON_LONG SHOULD<2> be sent if the INFO_LOGONNOTIFY (0x00000040) flag was set by the client in the Client Info PDU (see sections 2.2.1.11 and 3.3.5.3.1) or if the username or domain used to log on to the session is different from what was sent in the Client Info PDU (the original username or domain might have been invalid, resulting in the user having to re-enter their credentials at a remoted logon prompt). The LONG_CREDENTIALS_SUPPORTED (0x00000004) flag, in the **extraFlags** field of the General Capability Set (section 2.2.7.1.1) received from the client (see section 3.3.5.3.13.2), determines whether the INFOTYPE_LOGON or INFOTYPE_LOGON_LONG type is used.

A logon notification of type INFOTYPE_LOGON_PLAINNOTIFY SHOULD be sent whenever a notification of type INFOTYPE_LOGON or INFOTYPE_LOGON_LONG would not be sent.

The techniques described in section 3.3.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.3.5.10 Controlling Server Graphics Output

3.3.5.10.1 Processing Refresh Rect PDU

The structure and fields of the Refresh Rect PDU are described in section 2.2.11.2, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST send updated graphics data for the region specified by the PDU.

3.3.5.10.2 Processing Suppress Output PDU

The structure and fields of the Suppress Output PDU are described in section 2.2.11.3, and the techniques described in section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST stop or resume sending graphics updates, depending on the value of the **suppressOutput** field in the PDU.

3.3.6 Timer Events

No server timer events are used.

3.3.6.1 Connection Sequence Timeout

If the RDP Connection Sequence (see Sections 1.3.1.1 and 1.3.1.2) does not complete within 60 seconds, the server MAY terminate the connection to the client.

3.3.7 Other Local Events

No additional events are used.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting.

4.1 Annotated Connection Sequence

4.1.1 Client X.224 Connection Request PDU

The following is an annotated dump of the Client X.224 Connection Request PDU (section 2.2.1.1).

```
00000000 03 00 00 2c 27 e0 00 00 00 00 00 43 6f 6f 6b 69     ...,'......Cooki
00000010 65 3a 20 6d 73 74 73 68 61 73 68 3d 65 6c 74 6f     e: mstshash=elto
00000020 6e 73 0d 0a 01 00 08 00 00 00 00 00                 ns..........

03 -> TPKT Header: version = 3
00 -> TPKT Header: Reserved = 0
00 -> TPKT Header: Packet length - high part
2c -> TPKT Header: Packet length - low part (total = 44 bytes)
27 -> X.224: Length indicator (39 bytes)
e0 -> X.224: Type (high nibble) = 0xe = CR TPDU; credit (low nibble)= 0
00 00 -> X.224: Destination reference = 0
00 00 -> X.224: Source reference = 0
00 -> X.224: Class and options = 0

43 6f 6f 6b 69 65 3a 20 6d 73 74 73 68 61 73 68
3d 65 6c 74 6f 6e 73 -> "Cookie: mstshash=eltons"
0d -> CR (carriage return)
0a -> LF (line feed)

01 -> RDP_NEG_REQ::type (TYPE_RDP_NEG_REQ)
00 -> RDP_NEG_REQ::flags (0)
08 00 -> RDP_NEG_REQ::length (8 bytes)
00 00 00 00 -> RDP_NEG_REQ: Requested protocols (PROTOCOL_RDP)
```

4.1.2 Server X.224 Connection Confirm PDU

The following is an annotated dump of the Server X.224 Connection Confirm PDU (section 2.2.1.2).

```
00000000 03 00 00 13 0e d0 00 00 12 34 00 02 00 08 00 01 .........4......
00000010 00 00 00                                        ...

03 -> TPKT Header: TPKT version = 3
00 -> TPKT Header: Reserved = 0
00 -> TPKT Header: Packet length - high part
13 -> TPKT Header: Packet length - low part (total = 19 bytes)
0e -> X.224: Length indicator (14 bytes)
d0 -> X.224: Type (high nibble) = 0xd = CC TPDU; credit (low nibble) = 0
00 00 -> X.224: Destination reference = 0
12 34 -> X.224: Source reference = 0x1234 (bogus value)
00 -> X.224: Class and options = 0

02 -> RDP_NEG_RSP::type (TYPE_RDP_NEG_RSP)
00 -> RDP_NEG_RSP::flags (0)
08 00 -> RDP_NEG_RSP::length (8 bytes)
00 00 00 00 -> RDP_NEG_RSP: Selected protocols (PROTOCOL_RDP)
```

4.1.3 Client MCS Connect Initial PDU with GCC Conference Create Request

The following is an annotated dump of the Client MCS Connect Initial PDU with GCC Conference Create Request (section 2.2.1.3).

```
00000000 03 00 01 a0 02 f0 80 7f 65 82 01 94 04 01 01 04 ........e.......
00000010 01 01 01 01 ff 30 19 02 01 22 02 01 02 02 01 00 .....0..."......
00000020 02 01 01 02 01 00 02 01 01 02 02 ff ff 02 01 02 ................
00000030 30 19 02 01 01 02 01 01 02 01 01 02 01 01 02 01 0...............
00000040 00 02 01 01 02 02 04 20 02 01 02 30 1c 02 02 ff ....... ...0....
00000050 ff 02 02 fc 17 02 02 ff ff 02 01 01 02 01 00 02 ................
00000060 01 01 02 02 ff ff 02 01 02 04 82 01 33 00 05 00 ............3...
00000070 14 7c 00 01 81 2a 00 08 00 10 00 01 c0 00 44 75 .|...*........Du
00000080 63 61 81 1c 01 c0 d8 00 04 00 08 00 00 05 00 04 ca..............
00000090 01 ca 03 aa 09 04 00 00 ce 0e 00 00 45 00 4c 00 ............E.L.
000000a0 54 00 4f 00 4e 00 53 00 2d 00 44 00 45 00 56 00 T.O.N.S.-.D.E.V.
000000b0 32 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 2...............
000000c0 00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00 00 ................
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 00 00 00 00 00 00 00 00 01 ca 01 00 00 00 00 00 ................
00000110 18 00 07 00 01 00 36 00 39 00 37 00 31 00 32 00 ......6.9.7.1.2.
00000120 2d 00 37 00 38 00 33 00 2d 00 30 00 33 00 35 00 -.7.8.3.-.0.3.5.
00000130 37 00 39 00 37 00 34 00 2d 00 34 00 32 00 37 00 7.9.7.4.-.4.2.7.
00000140 31 00 34 00 00 00 00 00 00 00 00 00 00 00 00 00 1.4.............
00000150 00 00 00 00 00 00 00 00 01 00 00 00 04 c0 0c 00 ................
00000160 0d 00 00 00 00 00 00 00 02 c0 0c 00 1b 00 00 00 ................
00000170 00 00 00 00 03 c0 2c 00 03 00 00 00 72 64 70 64 ......,.....rdpd
00000180 72 00 00 00 00 00 80 80 63 6c 69 70 72 64 72 00 r.......cliprdr.
00000190 00 00 a0 c0 72 64 70 73 6e 64 00 00 00 00 00 c0 ....rdpsnd......

03 -> TPKT: TPKT version = 3
00 -> TPKT: Reserved = 0
01 -> TPKT: Packet length - high part
a0 -> TPKT: Packet length - low part (total = 416 bytes)
02 -> X.224: Length indicator = 2
f0 -> X.224: Type = 0xf0 = Data TPDU
80 -> X.224: EOT

7f 65 -> BER: Application-Defined Type = APPLICATION 101 =
Connect-Initial
This is the BER encoded multiple octet variant of the ASN.1 type
field. The multiple octet variant is used when the type can be
greater than 30, and is constructed as follows:

  7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0        7 6 5 4 3 2 1 0
+-----------------+  +-----------------+     +-----------------+
| C C F 1 1 1 1 1 |  | 1 T T T T T T T | ... | 0 T T T T T T T |
+-----------------+  +-----------------+     +-----------------+
         1                    2                       n

In this case, CC = 01 which means the type is APPLICATION defined,
and F = 1 to indicate that the type is constructed (as opposed
to primitive). There is only one octet containing the type value
(the second octet, which has the form 0TTTTTTT), and hence the
type is 0x65 (MCS_TYPE_CONNECTINITIAL).

82 01 94 -> BER: Type Length = 404 bytes
This is the BER encoded definite long variant of the ASN.1 length
field. The long variant layout is constructed as follows:

  7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0        7 6 5 4 3 2 1 0
+-----------------+  +-----------------+     +-----------------+
| 1 (0 < n < 127) |  | L L L L L L L L | ... | L L L L L L L L |
+-----------------+  +-----------------+     +-----------------+
         1                   2                    n + 1

Since the most significant bit of the first byte (0x82) is set, the
low seven bits contain the number of length bytes, which means
that the number of length bytes is 2. Hence, 0x01 and 0x94 are
length bytes, which implies that the length is greater than 256
bytes and less than 65536 bytes, specifically 0x194 (404) bytes.

04 01 01 -> Connect-Initial::callingDomainSelector
The first byte (0x04) is the ASN.1 BER encoded OctetString type. The
length of the data is given by the second byte (1 byte), which
is encoded using the BER definite short variant of the ASN.1
length field. The third byte contains the value, which is 0x01.

04 01 01 -> Connect-Initial::calledDomainSelector

01 01 ff -> Connect-Initial::upwardFlag = TRUE
The first byte (0x01) is the ASN.1 BER encoded Boolean type. The
length of the data is given by the second byte (0x01, so the
length is 1 byte). The third byte contains the value, which is
0xff (TRUE).

30 19 -> Connect-Initial::targetParameters (25 bytes)
The first byte (0x30) is the ASN.1 BER encoded SequenceOf type. The
length of the sequence data is given by the second byte (0x19, so
the length is 25 bytes).

02 01 22 -> DomainParameters::maxChannelIds = 34
The first byte (0x02) is the ASN.1 BER encoded Integer type. The
length of the integer is given by the second byte (1 byte), and
the actual value is 34 (0x22).

02 01 02 -> DomainParameters::maxUserIds = 2
02 01 00 -> DomainParameters::maxTokenIds = 0
02 01 01 -> DomainParameters::numPriorities = 1
02 01 00 -> DomainParameters::minThroughput = 0
02 01 01 -> DomainParameters::maxHeight = 1
02 02 ff ff -> DomainParameters::maxMCSPDUsize = 65535
02 01 02 -> DomainParameters::protocolVersion = 2

30 19 -> Connect-Initial::minimumParameters (25 bytes)
02 01 01 -> DomainParameters::maxChannelIds = 1
02 01 01 -> DomainParameters::maxUserIds = 1
02 01 01 -> DomainParameters::maxTokenIds = 1
02 01 01 -> DomainParameters::numPriorities = 1
02 01 00 -> DomainParameters::minThroughput = 0
02 01 01 -> DomainParameters::maxHeight = 1
02 02 04 20 -> DomainParameters::maxMCSPDUsize = 1056
02 01 02 -> DomainParameters::protocolVersion = 2

30 1c -> Connect-Initial::maximumParameters (28 bytes)
0x02 0x02 0xff 0xff -> DomainParameters::maxChannelIds = 65535
0x02 0x02 0xfc 0x17 -> DomainParameters::maxUserIds = 64535
0x02 0x02 0xff 0xff -> DomainParameters::maxTokenIds = 65535
0x02 0x01 0x01 -> DomainParameters::numPriorities = 1
0x02 0x01 0x00 -> DomainParameters::minThroughput = 0
0x02 0x01 0x01 -> DomainParameters::maxHeight = 1
0x02 0x02 0xff 0xff -> DomainParameters::maxMCSPDUsize = 65535
0x02 0x01 0x02 -> DomainParameters::protocolVersion = 2

04 82 01 33 -> Connect-Initial::userData (307 bytes)
The first byte (0x04) is the ASN.1 OctetString type. The length is
encoded using the BER definite long variant format. Hence, since the
```

```
most significant bit of the second byte (0x82) is set, the low seven
bits contain the number of length bytes, which means that the number
of length bytes is 2. Hence, 0x01 and 0x33 are length bytes, which
implies that the length is greater than 256 bytes and less than
65536 bytes, specifically 0x133 (307) bytes.

PER encoded (basic aligned variant) GCC Connection Data (ConnectData):
00 05 00 14 7c 00 01 81 2a 00 08 00 10 00 01 c0
00 44 75 63 61 81 1c

0 - CHOICE: From Key select object (0) of type OBJECT IDENTIFIER
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

05 -> object length = 5 bytes

00 14 7c 00 01 -> object
The first byte gives the first two values in the sextuple (m and n),
as it is encoded as 40m + n. Hence, decoding the remaining data
yields the correct results:
OID = { 0 0 20 124 0 1 } = { itu-t(0) recommendation(0) t(20)
t124(124) version(0) 1 }
Description = Version 1 of ITU-T Recommendation T.124 (February
1998): "Generic Conference Control"

81 2a -> ConnectData::connectPDU length = 298 bytes
Since the most significant bit of the first byte (0x81) is set to 1
and the following bit is set to 0, the length is given by the low
six bits of the first byte and the second byte. Hence we get 0x12a,
which is 298 bytes.

PER encoded (basic aligned variant) GCC Conference Create Request PDU:
00 08 00 10 00 01 c0 00 44 75 63 61 81 1c

0x00:
0 - extension bit (ConnectGCCPDU)
0 - --\
0 -   | CHOICE: From ConnectGCCPDU select conferenceCreateRequest
(0) of type ConferenceCreateRequest
0 - --/
0 - extension bit (ConferenceCreateRequest)
0 - ConferenceCreateRequest::convenerPassword present
0 - ConferenceCreateRequest::password present
0 - ConferenceCreateRequest::conductorPrivileges present

0x08:
0 - ConferenceCreateRequest::conductedPrivileges present
0 - ConferenceCreateRequest::nonConductedPrivileges present
0 - ConferenceCreateRequest::conferenceDescription present
0 - ConferenceCreateRequest::callerIdentifier present
1 - ConferenceCreateRequest::userData present
0 - extension bit (ConferenceName)
0 - ConferenceName::text present
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   | ConferenceName::numeric length = 0 + 1 = 1 character
```

```
0 -   | (minimum for SimpleNumericString is 1)
0 -   |
0 -   |
0 - --/

0x10:
0 - --\
0 -   | ConferenceName::numeric = "1"
0 -   |
1 - --/
0 - ConferenceCreateRequest::lockedConference
0 - ConferenceCreateRequest::listedConference
0 - ConferenceCreateRequest::conducibleConference
0 - extension bit (TerminationMethod)

0x00:
0 - TerminationMethod::automatic
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

0x01:
0 - --\
0 -   |
0 -   |
0 -   | number of UserData sets = 1
0 -   |
0 -   |
0 -   |
1 - --/

0xc0:
1 - UserData::value present
1 - CHOICE: From Key select h221NonStandard (1) of type
H221NonStandardIdentifier
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   | h221NonStandard length = 0 + 4 = 4 octets
0 -   | (minimum for H221NonStandardIdentifier is 4)
0 -   |
0 -   |
0 - --/

44 75 63 61 -> h221NonStandard (client-to-server H.221 key) = "Duca"

81 1c -> UserData::value length = 284 bytes
Since the most significant bit of the first byte (0x81) is set to 1
and the following bit is set to 0, the length is given by the low six
bits of the first byte and the second byte. Hence we get 0x11c, which
is 284 bytes.

01 c0 d8 00 -> TS_UD_HEADER::type = CS_CORE (0xc001), length = 216
```

```
bytes

04 00 08 00 -> TS_UD_CS_CORE::version = 0x0008004
00 05 -> TS_UD_CS_CORE::desktopWidth = 1280
00 04 -> TS_UD_CS_CORE::desktopHeight = 1024
01 ca -> TS_UD_CS_CORE::colorDepth = RNS_UD_COLOR_8BPP (0xca01)
03 aa -> TS_UD_CS_CORE::SASSequence
09 04 00 00 -> TS_UD_CS_CORE::keyboardLayout = 0x409 = 1033 =
English (US)
ce 0e 00 00 -> TS_UD_CS_CORE::clientBuild = 3790

45 00 4c 00 54 00 4f 00 4e 00 53 00 2d 00 44 00
45 00 56 00 32 00 00 00 00 00 00 00 00 00 00 00 ->
TS_UD_CS_CORE::clientName = ELTONS-TEST2

04 00 00 00 -> TS_UD_CS_CORE::keyboardType
00 00 00 00 -> TS_UD_CS_CORE::keyboardSubtype
0c 00 00 00 -> TS_UD_CS_CORE::keyboardFunctionKey

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_UD_CS_CORE::imeFileName = ""

01 ca -> TS_UD_CS_CORE::postBeta2ColorDepth = RNS_UD_COLOR_8BPP
(0xca01)

01 00 -> TS_UD_CS_CORE::clientProductId
00 00 00 00 -> TS_UD_CS_CORE::serialNumber
18 00 -> TS_UD_CS_CORE::highColorDepth = 24 bpp

07 00 -> TS_UD_CS_CORE::supportedColorDepths
0x07
= 0x01 | 0x02 | 0x04
= RNS_UD_24BPP_SUPPORT | RNS_UD_16BPP_SUPPORT | RNS_UD_15BPP_SUPPORT

01 00 -> TS_UD_CS_CORE::earlyCapabilityFlags
0x01
= RNS_UD_CS_SUPPORT_ERRINFO_PDU

36 00 39 00 37 00 31 00 32 00 2d 00 37 00 38 00
33 00 2d 00 30 00 33 00 35 00 37 00 39 00 37 00
34 00 2d 00 34 00 32 00 37 00 31 00 34 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_UD_CS_CORE::clientDigProductId = "69712-783-0357974-42714"

00 00 -> TS_UD_CS_CORE::pad2octets
00 00 00 00 -> TS_UD_CS_CORE::serverSelectedProtocols

04 c0 0c 00 -> TS_UD_HEADER::type = CS_CLUSTER (0xc004), length = 12
bytes

0d 00 00 00 -> TS_UD_CS_CLUSTER::Flags = 0x0d
0x0d
= 0x03 << 2 | 0x01
= REDIRECTION_VERSION4 << 2 | REDIRECTION_SUPPORTED

00 00 00 00 -> TS_UD_CS_CLUSTER::RedirectedSessionID

02 c0 0c 00 -> TS_UD_HEADER::type = CS_SECURITY (0xc002), length =
12 bytes

1b 00 00 00 -> TS_UD_CS_SEC::encryptionMethods
0x1b
```

```
= 0x01 | 0x02 | 0x08 | 0x10
= 40BIT_ENCRYPTION_FLAG | 128BIT_ENCRYPTION_FLAG |
56BIT_ENCRYPTION_FLAG | FIPS_ENCRYPTION_FLAG

00 00 00 00 -> TS_UD_CS_SEC::extEncryptionMethods

03 c0 2c 00 -> TS_UD_HEADER::type = CS_NET (0xc003), length = 44 bytes

03 00 00 00 -> TS_UD_CS_NET::channelCount = 3
72 64 70 64 72 00 00 00 -> CHANNEL_DEF::name = "rdpdr"

00 00 80 80 -> CHANNEL_DEF::options = 0x80800000
0x80800000
= 0x80000000 | 0x00800000
= INITIALIZED | COMPRESS_RDP

63 6c 69 70 72 64 72 00 -> CHANNEL_DEF::name = "cliprdr"

00 00 a0 c0 -> CHANNEL_DEF::options = 0xc0a00000
0xc0a00000
= 0x80000000 | 0x40000000 | 0x00800000 | 0x00200000
= INITIALIZED | ENCRYPT_RDP | COMPRESS_RDP | SHOW_PROTOCOL

72 64 70 73 6e 64 00 00 -> CHANNEL_DEF::name = "rdpsnd"

00 00 00 c0 -> CHANNEL_DEF::options = 0xc0000000
0xc0000000
= 0x80000000 | 0x40000000
= INITIALIZED | ENCRYPT_RDP
```

4.1.4 Server MCS Connect Response PDU with GCC Conference Create Response

The following is an annotated dump of the Server MCS Connect Response PDU with GCC Conference Create Response (section 2.2.1.4).

```
00000000 03 00 01 51 02 f0 80 7f 66 82 01 45 0a 01 00 02 ...Q....f..E....
00000010 01 00 30 1a 02 01 22 02 01 03 02 01 00 02 01 01 ..0...".........
00000020 02 01 00 02 01 01 02 03 00 ff f8 02 01 02 04 82 ................
00000030 01 1f 00 05 00 14 7c 00 01 2a 14 76 0a 01 01 00 ......|..*.v....
00000040 01 c0 00 4d 63 44 6e 81 08 01 0c 0c 00 04 00 08 ...McDn.........
00000050 00 00 00 00 00 03 0c 10 00 eb 03 03 00 ec 03 ed ................
00000060 03 ee 03 00 00 02 0c ec 00 02 00 00 00 02 00 00 ................
00000070 00 20 00 00 00 b8 00 00 00 10 11 77 20 30 61 0a . .........w 0a.
00000080 12 e4 34 a1 1e f2 c3 9f 31 7d a4 5f 01 89 34 96 ..4.....1}._..4.
00000090 e0 ff 11 08 69 7f 1a c3 d2 01 00 00 00 01 00 00 ....i...........
000000a0 00 01 00 00 00 06 00 5c 00 52 53 41 31 48 00 00 .......\.RSA1H..
000000b0 00 00 02 00 00 3f 00 00 00 01 00 01 00 cb 81 fe .....?..........
000000c0 ba 6d 61 c3 55 05 d5 5f 2e 87 f8 71 94 d6 f1 a5 .ma.U.._...q....
000000d0 cb f1 5f 0c 3d f8 70 02 96 c4 fb 9b c8 3c 2d 55 .._.=.p......<-U
000000e0 ae e8 ff 32 75 ea 68 79 e5 a2 01 fd 31 a0 b1 1f ...2u.hy....1...
000000f0 55 a6 1f c1 f6 d1 83 88 63 26 56 12 bc 00 00 00 U.......c&V.....
00000100 00 00 00 00 00 08 00 48 00 e9 e1 d6 28 46 8b 4e .......H....(F.N
00000110 f5 0a df fd ee 21 99 ac b4 e1 8f 5f 81 57 82 ef .....!....._.W..
00000120 9d 96 52 63 27 18 29 db b3 4a fd 9a da 42 ad b5 ..Rc'.)..J...B..
00000130 69 21 89 0e 1d c0 4c 1a a8 aa 71 3e 0f 54 b9 9a i!....L...q>.T..
00000140 e4 99 68 3f 6c d6 76 84 61 00 00 00 00 00 00 00 ..h?l.v.a.......
00000150 00                                              .

03 00 01 51 -> TPKT Header (length = 337 bytes)
02 f0 80 -> X.224 Data TPDU
```

```
7f 66 -> BER: Application-Defined Type = APPLICATION 102 =
Connect-Response
82 01 45 -> BER: Type Length = 325 bytes

0a 01 00 -> Connect-Response::result = rt-successful (0)
The first byte (0x0a) is the ASN.1 BER encoded Enumerated type. The
length of the value is given by the second byte (1 byte), and the
actual value is 0 (rt-successful).

02 01 00 -> Connect-Response::calledConnectId = 0

30 1a -> Connect-Response::domainParameters (26 bytes)
02 01 22 -> DomainParameters::maxChannelIds = 34
02 01 02 -> DomainParameters::maxUserIds = 3
02 01 00 -> DomainParameters::maximumTokenIds = 0
02 01 01 -> DomainParameters::numPriorities = 1
02 01 00 -> DomainParameters::minThroughput = 0
02 01 01 -> DomainParameters::maxHeight = 1
02 03 00 ff f8 -> DomainParameters::maxMCSPDUsize = 65528
02 01 02 -> DomainParameters::protocolVersion = 2

04 82 01 1f -> Connect-Response::userData (287 bytes)

PER encoded (basic aligned variant) GCC Connection Data (ConnectData):
00 05 00 14 7c 00 01 2a 14 76 0a 01 01 00 01 c0
00 4d 63 44 6e 81 08

00 05 -> Key::object length = 5 bytes
00 14 7c 00 01 -> Key::object = { 0 0 20 124 0 1 }

2a -> ConnectData::connectPDU length = 42 bytes
Note that this length is hard-coded by the server and ignored by the
client.

PER encoded (basic aligned variant) GCC Conference Create Response
PDU:
14 76 0a 01 01 00 01 c0 00 00 4d 63 44 6e 81 08

0x14:
0 - extension bit (ConnectGCCPDU)
0 - --\
0 -   | CHOICE: From ConnectGCCPDU select conferenceCreateResponse
(1) of type ConferenceCreateResponse
1 - --/
0 - extension bit (ConferenceCreateResponse)
1 - ConferenceCreateResponse::userData present
0 - padding
0 - padding

0x76:
0 - --\
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
1 -   |
0 -   |
      | ConferenceCreateResponse::nodeID = 0x760a + 1001 = 30218 +
1001 = 31219
0x0a: | (minimum for UserID is 1001)
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
0 -   |
1 -   |
0 - --/

0x01:
0 - --\
0 -   |
0 -   |
0 -   | ConferenceCreateResponse::tag length = 1 byte
0 -   |
0 -   |
0 -   |
1 - --/

0x01:
0 - --\
0 -   |
0 -   |
0 -   | ConferenceCreateResponse::tag = 1
0 -   |
0 -   |
0 -   |
1 - --/

0x00:
0 - extension bit (Result)
0 - --\
0 -   | ConferenceCreateResponse::result = success (0)
0 - --/
0 - padding
0 - padding
0 - padding
0 - padding

0x01:
0 - --\
0 -   |
0 -   |
0 -   | number of UserData sets = 1
0 -   |
0 -   |
0 -   |
1 - --/

0xc0:
1 - UserData::value present
1 - CHOICE: From Key select h221NonStandard (1) of type
H221NonStandardIdentifier
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   | h221NonStandard length = 0 + 4 = 4 octets
0 -   | (minimum for H221NonStandardIdentifier is 4)
0 -   |
0 -   |
0 - --/
```

```
4d 63 44 6e -> h221NonStandard (server-to-client H.221 key) = "McDn"

81 08 -> UserData::value length = 264 bytes

01 0c 0c 00 -> TS_UD_HEADER::type = SC_CORE (0x0c01), length = 12
bytes

04 00 08 00 -> TS_UD_SC_CORE::version = 0x0008004
00 00 00 00 -> TS_UD_SC_CORE::clientRequestedProtocols = PROTOCOL_RDP

03 0c 10 00 -> TS_UD_HEADER::type = SC_NET (0x0c03), length = 16 bytes

eb 03 -> TS_UD_SC_NET::MCSChannelID = 0x3eb = 1003 (I/O channel)
03 00 -> TS_UD_SC_NET::channelCount = 3
ec 03 -> channel0 = 0x3ec = 1004 (rdpdr)
ed 03 -> channel1 = 0x3ed = 1005 (cliprdr)
ee 03 -> channel2 = 0x3ee = 1006 (rdpsnd)
00 00 -> padding

02 0c ec 00 -> TS_UD_HEADER::type = SC_SECURITY, length = 236

02 00 00 00 -> TS_UD_SC_SEC1::encryptionMethod =
128BIT_ENCRYPTION_FLAG
02 00 00 00 -> TS_UD_SC_SEC1::encryptionLevel =
TS_ENCRYPTION_LEVEL_CLIENT_COMPATIBLE
20 00 00 00 -> TS_UD_SC_SEC1::serverRandomLen = 32 bytes
b8 00 00 00 -> TS_UD_SC_SEC1::serverCertLen = 184 bytes

10 11 77 20 30 61 0a 12 e4 34 a1 1e f2 c3 9f 31
7d a4 5f 01 89 34 96 e0 ff 11 08 69 7f 1a c3 d2 ->
TS_UD_SC_SEC1::serverRandom

TS_UD_SC_SEC1::serverCertificate:
01 00 00 00 01 00 00 00 01 00 00 00 06 00 5c 00
52 53 41 31 48 00 00 00 00 02 00 00 3f 00 00 00
01 00 01 00 cb 81 fe ba 6d 61 c3 55 05 d5 5f 2e
87 f8 71 94 d6 f1 a5 cb f1 5f 0c 3d f8 70 02 96
c4 fb 9b c8 3c 2d 55 ae e8 ff 32 75 ea 68 79 e5
a2 01 fd 31 a0 b1 1f 55 a6 1f c1 f6 d1 83 88 63
26 56 12 bc 00 00 00 00 00 00 00 00 08 00 48 00
e9 e1 d6 28 46 8b 4e f5 0a df fd ee 21 99 ac b4
e1 8f 5f 81 57 82 ef 9d 96 52 63 27 18 29 db b3
4a fd 9a da 42 ad b5 69 21 89 0e 1d c0 4c 1a a8
aa 71 3e 0f 54 b9 9a e4 99 68 3f 6c d6 76 84 61
00 00 00 00 00 00 00 00

01 00 00 00 -> PROPRIETARYSERVERCERTIFICATE::dwVersion = 1
01 00 00 00 -> PROPRIETARYSERVERCERTIFICATE::dwSigAlgId = MD5RSA (1)
01 00 00 00 -> PROPRIETARYSERVERCERTIFICATE::dwKeyAlgId = RSAKEY (1)
06 00 -> PROPRIETARYSERVERCERTIFICATE::wPublicKeyBlobType =
BB_RSA_KEY_BLOB (6)
5c 00 -> PROPRIETARYSERVERCERTIFICATE::wPublicKeyBlobLen = 92 bytes

PROPRIETARYSERVERCERTIFICATE::PublicKeyBlob:
52 53 41 31 48 00 00 00 00 02 00 00 3f 00 00 00
01 00 01 00 cb 81 fe ba 6d 61 c3 55 05 d5 5f 2e
87 f8 71 94 d6 f1 a5 cb f1 5f 0c 3d f8 70 02 96
c4 fb 9b c8 3c 2d 55 ae e8 ff 32 75 ea 68 79 e5
a2 01 fd 31 a0 b1 1f 55 a6 1f c1 f6 d1 83 88 63
26 56 12 bc 00 00 00 00 00 00 00 00

52 53 41 31 -> RSA_PUBLIC_KEY::magic = "RSA1"
48 00 00 00 -> RSA_PUBLIC_KEY::keylen = 72 bytes
00 02 00 00 -> RSA_PUBLIC_KEY::bitlen = 0x0200 = 512 bits
```

```
3f 00 00 00 -> RSA_PUBLIC_KEY::datalen = 63 bytes
01 00 01 00 -> RSA_PUBLIC_KEY::pubExp = 0x00010001

cb 81 fe ba 6d 61 c3 55 05 d5 5f 2e 87 f8 71 94
d6 f1 a5 cb f1 5f 0c 3d f8 70 02 96 c4 fb 9b c8
3c 2d 55 ae e8 ff 32 75 ea 68 79 e5 a2 01 fd 31
a0 b1 1f 55 a6 1f c1 f6 d1 83 88 63 26 56 12 bc
00 00 00 00 00 00 00 00 -> RSA_PUBLIC_KEY::modulus

08 00 -> PROPRIETARYSERVERCERTIFICATE::wSignatureBlobType =
BB_RSA_SIGNATURE_BLOB (8)
48 00 -> PROPRIETARYSERVERCERTIFICATE::wSignatureBlobLen = 72 bytes

e9 e1 d6 28 46 8b 4e f5 0a df fd ee 21 99 ac b4
e1 8f 5f 81 57 82 ef 9d 96 52 63 27 18 29 db b3
4a fd 9a da 42 ad b5 69 21 89 0e 1d c0 4c 1a a8
aa 71 3e 0f 54 b9 9a e4 99 68 3f 6c d6 76 84 61
00 00 00 00 00 00 00 00 -> PROPRIETARYSERVERCERTIFICATE::SignatureBlob
```

4.1.5 Client MCS Erect Domain Request PDU

The following is an annotated dump of the Client MCS Erect Domain Request PDU (section 2.2.1.5).

```
00000000 03 00 00 0c 02 f0 80 04 01 00 01 00     ............

03 00 00 0c -> TPKT Header (length = 12 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
04 01 00 01 00

0x04:
0 - --\
0 -   |
0 -   | CHOICE: From DomainMCSPDU select erectDomainRequest (1) of
type ErectDomainRequest
0 -   |
0 -   |
1 - --/
0 - padding
0 - padding

0x01:
0 - --\
0 -   |
0 -   |
0 -   | ErectDomainRequest::subHeight length = 1 byte
0 -   |
0 -   |
0 -   |
1 - --/

0x00:
0 - --\
0 -   |
0 -   |
0 -   | ErectDomainRequest::subHeight = 0
```

```
0 -   |
0 -   |
0 -   |
0 - --/

0x01:
0 - --\
0 -   |
0 -   |
0 -   | ErectDomainRequest::subInterval length = 1 byte
0 -   |
0 -   |
0 -   |
1 - --/

0x00:
0 - --\
0 -   |
0 -   |
0 -   | ErectDomainRequest::subInterval = 0
0 -   |
0 -   |
0 -   |
0 - --/
```

4.1.6 Client MCS Attach User Request PDU

The following is an annotated dump of the Client MCS Attach User Request PDU (section 2.2.1.6).

```
00000000 03 00 00 08 02 f0 80 28                      .......(

03 00 00 08 -> TPKT Header (length = 8 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
28

0x28:
0 - --\
0 -   |
1 -   | CHOICE: From DomainMCSPDU select attachUserRequest (10) of
type AttachUserRequest
0 -   |
1 -   |
0 - --/
0 - padding
0 - padding
```

4.1.7 Server MCS Attach-User Confirm PDU

The following is an annotated dump of the Server MCS Attach User Confirm PDU (section 2.2.1.7).

```
00000000 03 00 00 0b 02 f0 80 2e 00 00 06          ...........
```

```
03 00 00 0b -> TPKT Header (length = 11 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
2e 00 00 06

0x2e:
0 - --\
0 -   |
1 -   | CHOICE: From DomainMCSPDU select attachUserConfirm (11) of
type AttachUserConfirm
0 -   |
1 -   |
1 - --/
1 - AttachUserConfirm::initiator present
0 - --\
      |
0x00: | AttachUserConfirm::result = rt-successful (0)
0 -   |
0 -   |
0 - --/
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
      | AttachUserConfirm::initiator = 0x0006 + 1001 = 0x03ef =
1007 (user channel)
0x06: |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
0 - --/
```

4.1.8 MCS Channel Join Request and Confirm PDUs

4.1.9 Channel 1007

4.1.9.1 Client Join Request PDU for Channel 1007 (User Channel)

The following is an annotated dump of the Client MCS Channel Join Request PDU (section 2.2.1.8).

```
00000000 03 00 00 0c 02 f0 80 38 00 06 03 ef             .......8....

03 00 00 0c -> TPKT Header (length = 12 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
38 00 06 03 ef

0x38:
0 - --\
0 -   |
1 -   | CHOICE: From DomainMCSPDU select channelJoinRequest (14) of
type ChannelJoinRequest
1 -   |
1 -   |
0 - --/
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
      | ChannelJoinRequest::initiator = 0x06 + 1001 = 1007 (0x03ef)
0x06: |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
0 - --/

0x03:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
      | ChannelJoinRequest::channelId = 0x03ef = 1007
```

```
0xef: |
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
1 -   |
1 -   |
1 - --/
```

4.1.9.2 Server Join Confirm PDU for Channel 1007 (User Channel)

The following is an annotated dump of the Client MCS Channel Join Confirm PDU (section 2.2.1.9).

```
00000000 03 00 00 0f 02 f0 80 3e 00 00 06 03 ef 03 ef    .......>.......

03 00 00 0f -> TPKT Header (length = 15 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
3e 00 00 06 03 ef 03 ef

0x3e:
0 - --\
0 -   |
1 -   | CHOICE: From DomainMCSPDU select channelJoinConfirm (15) of
type ChannelJoinConfirm
1 -   |
1 -   |
1 - --/
1 - ChannelJoinConfirm::channelId present
0 - --\
      |
0x00: | ChannelJoinConfirm::result = rt-successful (0)
0 -   |
0 -   |
0 - --/
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
      | ChannelJoinConfirm::initiator = 0x06 + 1001 = 1007 (0x03ef)
0x06: |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
0 - --/

0x03:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
      | ChannelJoinConfirm::requested = 0x03ef = 1007
0xef: |
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
1 -   |
1 -   |
1 - --/

0x03:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
      | ChannelJoinConfirm::channelId = 0x03ef = 1007
0xef: |
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
1 -   |
1 -   |
1 - --/
```

4.1.10 Channel 1003

4.1.10.1 Client Join Request PDU for Channel 1003 (I/O Channel)

The following is an annotated dump of the Client MCS Channel Join Request PDU.

```
00000000 03 00 00 0c 02 f0 80 38 00 06 03 eb             .......8....

ChannelJoinRequest::initiator = 6 + 1001 = 1007
ChannelJoinRequest::channelId = 0x03eb = 1003
```

4.1.10.2 Server Join Confirm PDU for Channel 1003 (I/O Channel)

The following is an annotated dump of the Client MCS Channel Join Confirm PDU (section 2.2.1.9).

```
00000000 03 00 00 0f 02 f0 80 3e 00 00 06 03 eb 03 eb    .......>.......

ChannelJoinConfirm::result = rt-successful (0)
ChannelJoinConfirm::initiator = 6 + 1001 = 1007
ChannelJoinConfirm::requested = 0x03eb = 1003
ChannelJoinConfirm::channelId = 0x03eb = 1003
```

4.1.11 Channel 1004

4.1.11.1 Client Join Request PDU for Channel 1004 (rdpdr Channel)

The following is an annotated dump of the Client MCS Channel Join Request PDU (section 2.2.1.8).

```
00000000 03 00 00 0c 02 f0 80 38 00 06 03 ec             .......8....

ChannelJoinRequest::initiator = 6 + 1001 = 1007
ChannelJoinRequest::channelId = 0x03ec = 1004
```

4.1.11.2 Server Join Confirm PDU for Channel 1004 (rdpdr Channel)

The following is an annotated dump of the Client MCS Channel Join Confirm PDU (section 2.2.1.9).

```
00000000 03 00 00 0f 02 f0 80 3e 00 00 06 03 ec 03 ec    .......>.......

ChannelJoinConfirm::result = rt-successful (0)
ChannelJoinConfirm::initiator = 6 + 1001 = 1007
ChannelJoinConfirm::requested = 0x03ec = 1004
ChannelJoinConfirm::channelId = 0x03ec = 1004
```

4.1.12 Channel 1005

4.1.12.1 Client Join Request PDU for Channel 1005 (cliprdr Channel)

The following is an annotated dump of the Client MCS Channel Join Request PDU (section 2.2.1.8).

```
00000000 03 00 00 0c 02 f0 80 38 00 06 03 ed             .......8....

ChannelJoinRequest::initiator = 6 + 1001 = 1007
ChannelJoinRequest::channelId = 0x03ed = 1005
```

4.1.12.2 Server Join Confirm PDU for Channel 1005 (cliprdr Channel)

The following is an annotated dump of the Client MCS Channel Join Confirm PDU (section 2.2.1.9).

```
00000000 03 00 00 0f 02 f0 80 3e 00 00 06 03 ed 03 ed    .......>.......

ChannelJoinConfirm::result = rt-successful (0)
ChannelJoinConfirm::initiator = 6 + 1001 = 1007
ChannelJoinConfirm::requested = 0x03ed = 1005
```

```
ChannelJoinConfirm::channelId = 0x03ed = 1005
```

4.1.13 Channel 1006

4.1.13.1 Client Join Request PDU for Channel 1006 (rdpsnd Channel)

The following is an annotated dump of the Client MCS Channel Join Request PDU (section 2.2.1.8).

```
00000000 03 00 00 0c 02 f0 80 38 00 06 03 ee              .......8....

ChannelJoinRequest::initiator = 6 + 1001 = 1007
ChannelJoinRequest::channelId = 0x03ee = 1006
```

4.1.13.2 Server Join Confirm PDU for Channel 1006 (rdpsnd Channel)

The following is an annotated dump of the Client MCS Channel Join Confirm PDU (section 2.2.1.9).

```
00000000 03 00 00 0f 02 f0 80 3e 00 00 06 03 ee 03 ee    .......>.......

ChannelJoinConfirm::result = rt-successful (0)
ChannelJoinConfirm::initiator = 6 + 1001 = 1007
ChannelJoinConfirm::requested = 0x03ee = 1006
ChannelJoinConfirm::channelId = 0x03ee = 1006
```

4.1.14 Client Security Exchange PDU

The following is an annotated dump of the Client Security Exchange PDU (section 2.2.1.10).

```
00000000 03 00 00 5e 02 f0 80 64 00 06 03 eb 70 50 01 02 ...^...d....pP..
00000010 00 00 48 00 00 00 91 ac 0c 8f 64 8c 39 f4 e7 ff ..H.......d.9...
00000020 0a 3b 79 11 5c 13 51 2a cb 72 8f 9d b7 42 2e f7 .;y.\.Q*.r...B..
00000030 08 4c 8e ae 55 99 62 d2 81 81 e4 66 c8 05 ea d4 .L..U.b....f....
00000040 73 06 3f c8 5f af 2a fd fc f1 64 b3 3f 0a 15 1d s.?._.*...d.?...
00000050 db 2c 10 9d 30 11 00 00 00 00 00 00 00 00       .,..0.........

03 00 00 5e -> TPKT Header (length = 94 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) SendDataRequest PDU:
64 00 06 03 eb 70 50

0x64:
0 - --\
1 -   |
1 -   | CHOICE: From DomainMCSPDU select sendDataRequest (25) of
type SendDataRequest
0 -   |
0 -   |
1 - --/
0 - padding
0 - padding

0x00:
```

```
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
      | SendDataRequest::initiator = 0x06 + 1001 = 1007
0x06: |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
0 - --/

0x03:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
      | SendDataRequest::channelId = 0x03eb = 1003
0xeb: |
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
0 -   |
1 -   |
1 - --/

0x70:
0 - --\ SendDataRequest::dataPriority = 0x01 = high
1 - --/
1 - --\ SendDataRequest::segmentation = 0x03 = (0x02 | 0x01) =
(begin | end)
1 - --/
0 - padding
0 - padding
0 - padding
0 - padding

0x50:
0 - --\
1 -   |
0 -   |
1 -   | SendDataRequest::userData length = 80 bytes
0 -   |
0 -   |
0 -   |
0 - --/

01 02 -> TS_SECURITY_HEADER::flags = 0x0201
0x0201
= 0x0200 | 0x0001
= SEC_LICENSE_ENCRYPT_SC | SEC_EXCHANGE_PKT
```

```
00 00 -> TS_SECURITY_HEADER::flagsHi = 0x0000

48 00 00 00 -> TS_SECURITY_PACKET::length = 0x48 = 72 bytes

91 ac 0c 8f 64 8c 39 f4 e7 ff 0a 3b 79 11 5c 13
51 2a cb 72 8f 9d b7 42 2e f7 08 4c 8e ae 55 99
62 d2 81 81 e4 66 c8 05 ea d4 73 06 3f c8 5f af
2a fd fc f1 64 b3 3f 0a 15 1d db 2c 10 9d 30 11 ->
TS_SECURITY_PACKET::encryptedClientRandom

00 00 00 00 00 00 00 00 -> 8-bytes of rear padding (always present)
```

4.1.15 Client Info PDU

The following is an annotated dump of the Client Info PDU (section 2.2.1.11).

```
00000000 03 00 01 ab 02 f0 80 64 00 06 03 eb 70 81 9c 48 .......d....p..H
00000010 00 00 00 45 ca 46 fa 5e a7 be bc 74 21 d3 65 e9 ...E.F.^...t!.e.
00000020 ba 76 12 7c 55 4b 9d 84 3b 3e 07 29 20 73 25 7b .v.|UK..;>.) s%{
00000030 e6 9a bb e8 41 8a a0 69 3f 26 9a cd bc a6 03 27 ....A..i?&.....'
00000040 f5 ce bb a8 c2 ff 0f 38 a3 bf 74 81 ac cb c9 08 .......8..t.....
00000050 49 0a 43 cf 91 31 36 cd ba 3d 16 4f 11 d7 69 12 I.C..16..=.O..i.
00000060 c8 e9 57 c0 b8 0f c4 72 66 79 bd 86 ba 30 60 76 ..W....rfy...0`v
00000070 b4 cd 52 5e 79 8e 88 95 f0 9a 43 20 d9 96 74 1d ..R^y.....C ..t.
00000080 5c 8a 9a e3 8a 5d d2 55 17 8c f2 66 6b 3f 3d 3a \....].U...fk?=:
00000090 e3 2a d4 ff d5 11 30 30 e2 ff e2 e4 11 0c 7f 6a .*....00.......j
000000a0 1e a3 f4 2f dd 4f 89 8c c0 ca d3 8a 49 d7 00 d9 .../.O......I...
000000b0 09 40 ab 79 1a 72 f9 89 42 af 20 aa 50 c7 cd d0 .@.y.r..B. .P...
000000c0 b8 1e ab d3 eb 10 01 82 68 9f f5 c9 05 fe 20 bb ........h..... .
000000d0 7c 68 b4 72 cd 37 53 df 43 0a 6d de cb be 5f 80 |h.r.7S.C.m..._.
000000e0 05 1e b8 f3 5d 04 0c c6 66 3b 39 5f 5d a2 da b9 ....]...f;9_]...
000000f0 ea c9 da ba 7c 9d 4e 4a 4f 4a 16 04 ea 4e 23 d3 ....|.NJOJ...N#.
00000100 6d 2c 2b 42 58 19 69 10 23 d4 e1 af 46 34 fc 23 m,+BX.i.#...F4.#
00000110 81 59 54 65 5f 6c 67 57 14 62 57 94 f1 81 86 00 .YTe_lgW.bW.....
00000120 fe 1c 27 f6 76 e2 00 ea c5 f7 b5 e9 b2 ad ef 7f ..'.v...........
00000130 87 8b 8a b0 d3 1e 43 54 4b ab f6 ba 7f 5a b9 e5 ......CTK....Z..
00000140 2d 5f 81 ab 2a 15 c4 97 bc d3 92 9a da be 8a b0 -_..*...........
00000150 fb a4 1a a0 96 26 86 23 10 1b 21 0a 91 05 22 4d .....&.#..!..."M
00000160 6c 4d 01 4c 84 f3 50 56 4f 3a e4 c0 24 bf 35 f6 lM.L..PVO:..$.5.
00000170 f5 8b 3f 20 55 98 91 05 4d ee 46 95 44 6d 06 33 ..? U...M.F.Dm.3
00000180 42 1f 9f 84 91 e7 c5 9f 04 11 de cf a5 07 5f 27 B............._'
00000190 dd c0 ac b1 a7 98 9d 6d 79 00 70 33 bf 4e 16 23 .......my.p3.N.#
000001a0 57 f5 c7 88 82 d1 c6 a3 b4 0b 29                W.........)

03 00 01 ab -> TPKT Header (length = 427 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 81 9c -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x19c = 412 bytes

48 00 -> TS_SECURITY_HEADER::flags = 0x0048
0x0048
= 0x0040 | 0x0008
= SEC_INFO_PKT | SEC_ENCRYPT

00 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
```

Release: Thursday, February 7, 2008

```
45 ca 46 fa 5e a7 be bc -> TS_SECURITY_HEADER1::dataSignature

74 21 d3 65 e9 ba 76 12 7c 55 4b 9d 84 3b 3e 07
29 20 73 25 7b e6 9a bb e8 41 8a a0 69 3f 26 9a
cd bc a6 03 27 f5 ce bb a8 c2 ff 0f 38 a3 bf 74
81 ac cb c9 08 49 0a 43 cf 91 31 36 cd ba 3d 16
4f 11 d7 69 12 c8 e9 57 c0 b8 0f c4 72 66 79 bd
86 ba 30 60 76 b4 cd 52 5e 79 8e 88 95 f0 9a 43
20 d9 96 74 1d 5c 8a 9a e3 8a 5d d2 55 17 8c f2
66 6b 3f 3d 3a e3 2a d4 ff d5 11 30 30 e2 ff e2
e4 11 0c 7f 6a 1e a3 f4 2f dd 4f 89 8c c0 ca d3
8a 49 d7 00 d9 09 40 ab 79 1a 72 f9 89 42 af 20
aa 50 c7 cd d0 b8 1e ab d3 eb 10 01 82 68 9f f5
c9 05 fe 20 bb 7c 68 b4 72 cd 37 53 df 43 0a 6d
de cb be 5f 80 05 1e b8 f3 5d 04 0c c6 66 3b 39
5f 5d a2 da b9 ea c9 da ba 7c 9d 4e 4a 4f 4a 16
04 ea 4e 23 d3 6d 2c 2b 42 58 19 69 10 23 d4 e1
af 46 34 fc 23 81 59 54 65 5f 6c 67 57 14 62 57
94 f1 81 86 00 fe 1c 27 f6 76 e2 00 ea c5 f7 b5
e9 b2 ad ef 7f 87 8b 8a b0 d3 1e 43 54 4b ab f6
ba 7f 5a b9 e5 2d 5f 81 ab 2a 15 c4 97 bc d3 92
9a da be 8a b0 fb a4 1a a0 96 26 86 23 10 1b 21
0a 91 05 22 4d 6c 4d 01 4c 84 f3 50 56 4f 3a e4
c0 24 bf 35 f6 f5 8b 3f 20 55 98 91 05 4d ee 46
95 44 6d 06 33 42 1f 9f 84 91 e7 c5 9f 04 11 de
cf a5 07 5f 27 dd c0 ac b1 a7 98 9d 6d 79 00 70
33 bf 4e 16 23 57 f5 c7 88 82 d1 c6 a3 b4 0b 29 -> Encrypted
TS_INFO_PACKET

Decrypted TS_INFO_PACKET:
00000000 09 04 09 04 b3 43 00 00 0a 00 0c 00 00 00 00 00 .....C..........
00000010 00 00 4e 00 54 00 44 00 45 00 56 00 00 00 65 00 ..N.T.D.E.V...e.
00000020 6c 00 74 00 6f 00 6e 00 73 00 00 00 00 00 00 00 l.t.o.n.s.......
00000030 00 00 02 00 1e 00 31 00 35 00 37 00 2e 00 35 00 ......1.5.7...5.
00000040 39 00 2e 00 32 00 34 00 32 00 2e 00 31 00 35 00 9...2.4.2...1.5.
00000050 36 00 00 00 84 00 43 00 3a 00 5c 00 64 00 65 00 6.....C.:.\.d.e.
00000060 70 00 6f 00 74 00 73 00 5c 00 77 00 32 00 6b 00 p.o.t.s.\.w.2.k.
00000070 33 00 5f 00 31 00 5c 00 74 00 65 00 72 00 6d 00 3._.1.\.t.e.r.m.
00000080 73 00 72 00 76 00 5c 00 6e 00 65 00 77 00 63 00 s.r.v.\.n.e.w.c.
00000090 6c 00 69 00 65 00 6e 00 74 00 5c 00 6c 00 69 00 l.i.e.n.t.\.l.i.
000000a0 62 00 5c 00 77 00 69 00 6e 00 33 00 32 00 5c 00 b.\.w.i.n.3.2.\.
000000b0 6f 00 62 00 6a 00 5c 00 69 00 33 00 38 00 36 00 o.b.j.\.i.3.8.6.
000000c0 5c 00 6d 00 73 00 74 00 73 00 63 00 61 00 78 00 \.m.s.t.s.c.a.x.
000000d0 2e 00 64 00 6c 00 6c 00 00 00 e0 01 00 00 50 00 ..d.l.l.......P.
000000e0 61 00 63 00 69 00 66 00 69 00 63 00 20 00 53 00 a.c.i.f.i.c. .S.
000000f0 74 00 61 00 6e 00 64 00 61 00 72 00 64 00 20 00 t.a.n.d.a.r.d. .
00000100 54 00 69 00 6d 00 65 00 00 00 00 00 00 00 00 00 T.i.m.e.........
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 0a 00 00 00 05 00 02 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 50 00 61 00 63 00 69 00 66 00 69 00 63 00 ..P.a.c.i.f.i.c.
00000140 20 00 44 00 61 00 79 00 6c 00 69 00 67 00 68 00  .D.a.y.l.i.g.h.
00000150 74 00 20 00 54 00 69 00 6d 00 65 00 00 00 00 00 t. .T.i.m.e.....
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000170 00 00 00 00 04 00 00 00 01 00 02 00 00 00 00 00 ................
00000180 00 00 c4 ff ff ff 00 00 00 00 01 00 00 00 00 00 ................

09 04 09 04 -> TS_INFO_PACKET::CodePage = 0x04090409
Low word = 0x0409 = 1033 = English (US)
Since the INFO_UNICODE flag is set, this is the active input locale
identifier.

b3 43 00 00 -> TS_INFO_PACKET::flags = 0x000043b3
0x000043b3
= 0x00000001 |
  0x00000002 |
  0x00000010 |
  0x00000020 |
  0x00000080 |
  0x00000100 |
  0x00000200 |
  0x00004000
= INFO_MOUSE |
  INFO_DISABLECTRLALTDEL |
  INFO_UNICODE |
  INFO_MAXIMIZESHELL |
  INFO_COMPRESSION |
  INFO_ENABLEWINDOWSKEY |
  PACKET_COMPR_TYPE_64K << 9 |
  INFO_FORCE_ENCRYPTED_CS_PDU

0a 00 -> TS_INFO_PACKET::cbDomain = 0x0a = 10 bytes (not including
the size of the mandatory NULL terminator)
0c 00 -> TS_INFO_PACKET::cbUserName = 0x0c = 12 bytes (not including
the size of the mandatory NULL terminator)
00 00 -> TS_INFO_PACKET::cbPassword = 0 bytes
00 00 -> TS_INFO_PACKET::cbAlternateShell = 0 bytes
00 00 -> TS_INFO_PACKET::cbWorkingDir = 0 bytes

4e 00 54 00 44 00 45 00 56 00 00 00 -> TS_INFO_PACKET::Domain =
"NTDEV"
65 00 6c 00 74 00 6f 00 6e 00 73 00 00 00 ->
TS_INFO_PACKET::UserName = "eltons"
00 00 -> TS_INFO_PACKET::Password = ""
00 00 -> TS_INFO_PACKET::AlternateShell = ""
00 00 -> TS_INFO_PACKET::WorkingDir = ""

02 00 -> TS_EXTENDED_INFO_PACKET::clientAddressFamily = AF_INET (2)
1e 00 -> TS_EXTENDED_INFO_PACKET::cbClientAddress = 0x1e = 30 bytes
(including the size of the mandatory NULL terminator)

31 00 35 00 37 00 2e 00 35 00 39 00 2e 00 32 00
34 00 32 00 2e 00 31 00 35 00 36 00 00 00 ->
TS_EXTENDED_INFO_PACKET::clientAddress = "157.59.242.156"

84 00 -> TS_EXTENDED_INFO_PACKET::cbClientDir = 0x84 = 132 bytes
(including the size of the mandatory NULL terminator)

43 00 3a 00 5c 00 64 00 65 00 70 00 6f 00 74 00
73 00 5c 00 77 00 32 00 6b 00 33 00 5f 00 31 00
5c 00 74 00 65 00 72 00 6d 00 73 00 72 00 76 00
5c 00 6e 00 65 00 77 00 63 00 6c 00 69 00 65 00
6e 00 74 00 5c 00 6c 00 69 00 62 00 5c 00 77 00
69 00 6e 00 33 00 32 00 5c 00 6f 00 62 00 6a 00
5c 00 69 00 33 00 38 00 36 00 5c 00 6d 00 73 00
74 00 73 00 63 00 61 00 78 00 2e 00 64 00 6c 00
6c 00 00 00 -> TS_EXTENDED_INFO_PACKET::clientDir =
"C:\depots\w2k3_1\termsrv\newclient\lib\win32\obj\i386\mstscax.dll"

e0 01 00 00 -> TIME_ZONE_INFORMATION::Bias = 0x01e0 = 480 mins = 8 hrs

50 00 61 00 63 00 69 00 66 00 69 00 63 00 20 00
53 00 74 00 61 00 6e 00 64 00 61 00 72 00 64 00
20 00 54 00 69 00 6d 00 65 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TIME_ZONE_INFORMATION::StandardName = "Pacific Standard Time"

00 00 -> TIME_ZONE_INFORMATION::StandardDate::wYear = 0
0a 00 -> TIME_ZONE_INFORMATION::StandardDate::wMonth = 0x0a =
October (10)
00 00 -> TIME_ZONE_INFORMATION::StandardDate::wDayOfWeek = Sunday (0)
```

```
05 00 -> TIME_ZONE_INFORMATION::StandardDate::wDay = 5 (last Sunday)
02 00 -> TIME_ZONE_INFORMATION::StandardDate::wHour = 2am
00 00 -> TIME_ZONE_INFORMATION::StandardDate::wMinute = 0
00 00 -> TIME_ZONE_INFORMATION::StandardDate::wSecond = 0
00 00 -> TIME_ZONE_INFORMATION::StandardDate::wMilliseconds = 0

00 00 00 00 -> TIME_ZONE_INFORMATION::StandardBias = 0

50 00 61 00 63 00 69 00 66 00 69 00 63 00 20 00
44 00 61 00 79 00 6c 00 69 00 67 00 68 00 74 00
20 00 54 00 69 00 6d 00 65 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TIME_ZONE_INFORMATION::DaylightName = "Pacific Daylight Time"

00 00 -> TIME_ZONE_INFORMATION::DaylightDate::wYear = 0
04 00 -> TIME_ZONE_INFORMATION::DaylightDate::wMonth = April (4)
00 00 -> TIME_ZONE_INFORMATION::DaylightDate::wDayOfWeek = Sunday (0)
01 00 -> TIME_ZONE_INFORMATION::DaylightDate::wDay = 1 (first Sunday)
02 00 -> TIME_ZONE_INFORMATION::DaylightDate::wHour = 2am
00 00 -> TIME_ZONE_INFORMATION::DaylightDate::wMinute = 0
00 00 -> TIME_ZONE_INFORMATION::DaylightDate::wSecond = 0
00 00 -> TIME_ZONE_INFORMATION::DaylightDate::wMilliseconds = 0

c4 ff ff ff -> TIME_ZONE_INFORMATION::DaylightBias = 0xffffffc4 =
-60 (two's complement)

00 00 00 00 -> TS_EXTENDED_INFO_PACKET::clientSessionId = 0
01 00 00 00 -> TS_EXTENDED_INFO_PACKET::performanceFlags = 0x01 =
TS_PERF_DISABLE_WALLPAPER
00 00 -> TS_EXTENDED_INFO_PACKET::cbAutoReconnectLen = 0
```

4.1.16 Server License Error PDU - Valid Client

The following is an annotated dump of the Server License Error PDU - Valid Client (section 2.2.1.12).

```
00000000 03 00 00 2a 02 f0 80 68 00 01 03 eb 70 1c 88 02 ...*...h....p...
00000010 02 03 8d 43 9a ab d5 2a 31 39 62 4d c1 ec 0d 99 ...C...*19bM....
00000020 88 e6 da ab 2c 02 72 4d 49 90                   ....,.rMI.

03 00 00 2a -> TPKT Header (length = 42 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) SendDataIndication PDU:
68 00 01 03 eb 70 1c

0x68:
0 - --\
1 -   |
1 -   | CHOICE: From DomainMCSPDU select sendDataIndication (26) of
                type SendDataIndication
0 -   |
1 -   |
0 - --/
0 - padding
0 - padding

0x00:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
      | SendDataIndication::initiator = 0x01 + 1001 = 1002 (0x03ea)
0x01: |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 - --/

0x03:
0 - --\
0 -   |
0 -   |
0 -   |
0 -   |
0 -   |
1 -   |
1 -   |
      | SendDataIndication::channelId = 0x03eb = 1003
0xeb: |
1 -   |
1 -   |
1 -   |
0 -   |
1 -   |
0 -   |
1 -   |
1 - --/

0x70:
0 - --\ SendDataIndication::dataPriority = 0x01 = high
1 - --/
1 - --\ SendDataIndication::segmentation = 0x03 = (0x02 | 0x01) = (begin | end)
1 - --/
0 - padding
0 - padding
0 - padding
0 - padding

0x1c:
0 - --\
0 -   |
0 -   |
1 -   | SendDataIndication::userData length = 28 bytes
1 -   |
1 -   |
0 -   |
0 - --/

88 02 -> TS_SECURITY_HEADER::flags = 0x0288
0x0288
= 0x0008 | 0x0080 | 0x0200
= SEC_ENCRYPT | SEC_LICENSE_PKT | SEC_LICENSE_ENCRYPT_CS

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
8d 43 9a ab d5 2a 31 39 -> TS_SECURITY_HEADER1::dataSignature

62 4d c1 ec 0d 99 88 e6 da ab 2c 02 72 4d 49 90 -> Encrypted Licensing Packet
```

```
Decrypted Licensing Packet:
00000000 ff 03 10 00 07 00 00 00 02 00 00 00 04 00 00 00 ................

ff -> LICENSE_PREAMBLE::bMsgType = ERROR_ALERT
03 -> LICENSE_PREAMBLE::bVersion = 3 (RDP 5.0 and RDP 5.1)
10 00 -> LICENSE_PREAMBLE::wMsgSize = 0x10 = 16 bytes

07 00 00 00 -> LICENSE_ERROR_MESSAGE::dwErrorCode = STATUS_VALID_CLIENT
02 00 00 00 -> LICENSE_ERROR_MESSAGE::dwStateTransition = ST_NO_TRANSITION
04 00 -> LICENSE_ERROR_MESSAGE::bbErrorInfo::wBlobType = BB_ERROR_BLOB
00 00 -> LICENSE_ERROR_MESSAGE::bbErrorInfo::wBlobLen = 0
```

4.1.17 Server Demand Active PDU

The following is an annotated dump of the Server Demand Active PDU (section 2.2.1.13.1).

```
00000000 03 00 01 82 02 f0 80 68 00 01 03 eb 70 81 73 08 .......h....p.s.
00000010 00 02 03 56 02 e1 47 ac 5c 50 d9 72 f9 c3 32 0a ...V..G.\P.r..2.
00000020 c7 23 3f 5f 78 11 de e2 af 6c 9b f3 63 32 6b 18 .#?_x....l..c2k.
00000030 15 1c e5 e2 ff e2 61 f9 1e 99 90 c5 62 9b 8f 2a ......a.....b..*
00000040 c3 de bb 6f 3e 59 01 62 4f 75 e4 5c be e7 ce 08 ...o>Y.bOu.\....
00000050 44 b1 37 9f c0 27 55 bd e5 eb 7e 63 80 6a bf 8e D.7..'U...~c.j..
00000060 0e 21 f0 c3 70 f8 e9 4f da 72 0f e5 ca 2a f3 b5 .!..p..O.r...*..
00000070 9d d7 05 de 4d 35 49 80 37 2f 8a fb 4b c2 1f f8 ....M5I.7/..K...
00000080 01 4f 2f 1d 73 7b 95 01 52 9d b1 c6 d2 03 61 51 .O/.s{..R.....aQ
00000090 da 3a 17 86 77 36 05 a2 24 63 5c af 65 67 e7 8d .:..w6..$c\.eg..
000000a0 0b a3 71 e1 ec f3 e4 a1 24 ed c8 2a 4f 5d 9f 91 ..q.....$..*O]..
000000b0 89 91 1d 69 c5 f5 48 bb 37 b2 93 e9 35 21 7e 0d ...i..H.7...5!~.
000000c0 09 27 d6 16 d6 91 57 9c 7e f9 d2 a1 c5 26 63 de .'....W.~....&c.
000000d0 78 38 f7 77 08 95 76 e3 68 bc 26 82 18 3c fb f0 x8.w..v.h.&..<..
000000e0 ba 21 02 72 55 27 fa 8c e2 59 ba 86 dd 11 12 ba .!.rU'...Y......
000000f0 7e 87 74 3e c4 7c 57 3d 50 c0 b7 0f 85 a0 7b 1d ~.t>.|W=P.....{.
00000100 86 7a 03 b3 6d ef de 1b 59 5c 4d ea 65 34 f8 bf .z..m...Y\M.e4..
00000110 f3 50 6b 24 b5 30 85 1d e6 30 3b 99 0d 0b 31 b1 .Pk$.0...0;...1.
00000120 45 10 6b af 4a 38 bc 14 9c c5 c7 a7 24 b3 f9 6a E.k.J8......$..j
00000130 3a 87 c7 39 0f 59 b7 d6 3d c4 23 d7 d3 fe c5 f3 :..9.Y..=.#.....
00000140 b6 16 e4 2c c2 c7 27 a7 31 e9 d9 84 b8 19 59 ea ...,..'.1.....Y.
00000150 a7 e1 1c d2 8d a7 00 61 e9 b5 ab 0d 53 fe e2 cc .......a....S...
00000160 1d b8 93 39 c1 d4 e4 40 b3 e4 b8 a6 46 75 11 59 ...9...@....Fu.Y
00000170 c1 cb 60 72 7a 6d a8 1a fe 9d b7 4a 06 60 99 ad ..`rzm.....J.`..
00000180 81 48                                           .H

03 00 01 82 -> TPKT Header (length = 386 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 81 73 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x173 = 371 bytes

08 00 -> TS_SECURITY_HEADER::flags = 0x0800 = SEC_ENCRYPT
02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
56 02 e1 47 ac 5c 50 d9 -> TS_SECURITY_HEADER1::dataSignature

72 f9 c3 32 0a c7 23 3f 5f 78 11 de e2 af 6c 9b
f3 63 32 6b 18 15 1c e5 e2 ff e2 61 f9 1e 99 90
c5 62 9b 8f 2a c3 de bb 6f 3e 59 01 62 4f 75 e4
5c be e7 ce 08 44 b1 37 9f c0 27 55 bd e5 eb 7e
```

```
63 80 6a bf 8e 0e 21 f0 c3 70 f8 e9 4f da 72 0f
e5 ca 2a f3 b5 9d d7 05 de 4d 35 49 80 37 2f 8a
fb 4b c2 1f f8 01 4f 2f 1d 73 7b 95 01 52 9d b1
c6 d2 03 61 51 da 3a 17 86 77 36 05 a2 24 63 5c
af 65 67 e7 8d 0b a3 71 e1 ec f3 e4 a1 24 ed c8
2a 4f 5d 9f 91 89 91 1d 69 c5 f5 48 bb 37 b2 93
e9 35 21 7e 0d 09 27 d6 16 d6 91 57 9c 7e f9 d2
a1 c5 26 63 de 78 38 f7 77 08 95 76 e3 68 bc 26
82 18 3c fb f0 ba 21 02 72 55 27 fa 8c e2 59 ba
86 dd 11 12 ba 7e 87 74 3e c4 7c 57 3d 50 c0 b7
0f 85 a0 7b 1d 86 7a 03 b3 6d ef de 1b 59 5c 4d
ea 65 34 f8 bf f3 50 6b 24 b5 30 85 1d e6 30 3b
99 0d 0b 31 b1 45 10 6b af 4a 38 bc 14 9c c5 c7
a7 24 b3 f9 6a 3a 87 c7 39 0f 59 b7 d6 3d c4 23
d7 d3 fe c5 f3 b6 16 e4 2c c2 c7 27 a7 31 e9 d9
84 b8 19 59 ea a7 e1 1c d2 8d a7 00 61 e9 b5 ab
0d 53 fe e2 cc 1d b8 93 39 c1 d4 e4 40 b3 e4 b8
a6 46 75 11 59 c1 cb 60 72 7a 6d a8 1a fe 9d b7
4a 06 60 99 ad 81 48 -> Encrypted TS_DEMAND_ACTIVE_PDU

Decrypted TS_DEMAND_ACTIVE_PDU:
00000000 67 01 11 00 ea 03 ea 03 01 00 04 00 51 01 52 44 g...........Q.RD
00000010 50 00 0d 00 00 00 09 00 08 00 ea 03 dc e2 01 00 P...............
00000020 18 00 01 00 03 00 00 02 00 00 00 00 1d 04 00 00 ................
00000030 00 00 00 00 01 01 14 00 08 00 02 00 00 00 16 00 ................
00000040 28 00 00 00 00 00 70 f6 13 f3 01 00 00 00 01 00 (.....p.........
00000050 00 00 18 00 00 00 9c f6 13 f3 61 a6 82 80 00 00 ..........a.....
00000060 00 00 00 50 91 bf 0e 00 04 00 02 00 1c 00 18 00 ...P............
00000070 01 00 01 00 01 00 00 05 00 04 00 00 01 00 01 00 ................
00000080 00 00 01 00 00 00 03 00 58 00 00 00 00 00 00 00 ........X.......
00000090 00 00 00 00 00 00 00 00 00 00 40 42 0f 00 01 00 ..........@B....
000000a0 14 00 00 00 01 00 00 00 22 00 01 01 01 01 01 00 ........".......
000000b0 00 01 01 01 01 01 00 00 00 01 01 01 01 01 01 01 ................
000000c0 01 00 01 01 01 01 00 00 00 00 a1 06 00 00 40 42 ..............@B
000000d0 0f 00 40 42 0f 00 01 00 00 00 00 00 00 00 0a 00 ..@B............
000000e0 08 00 06 00 00 00 12 00 08 00 01 00 00 00 08 00 ................
000000f0 0a 00 01 00 19 00 19 00 0d 00 58 00 35 00 00 00 ..........X.5...
00000100 a1 06 00 00 40 42 0f 00 0c f6 13 f3 93 5a 37 f3 ....@B.......Z7.
00000110 00 90 30 e1 34 1c 38 f3 40 f6 13 f3 04 00 00 00 ..0.4.8.@.......
00000120 4c 54 dc e2 08 50 dc e2 01 00 00 00 08 50 dc e2 LT...P.......P..
00000130 00 00 00 00 38 f6 13 f3 2e 05 38 f3 08 50 dc e2 ....8.....8..P..
00000140 2c f6 13 f3 00 00 00 00 08 00 0a 00 01 00 19 00 ,...............
00000150 17 00 08 00 00 00 00 00 18 00 0b 00 00 00 00 00 ................
00000160 00 00 00 00 00 00 00                            .......

67 01 -> TS_SHARECONTROLHEADER::totalLength = 0x0167 = 359 bytes
11 00 -> TS_SHARECONTROLHEADER::pduType = 0x0011
0x0011
= 0x0010 | 0x0001
= TS_PROTOCOL_VERSION | PDUTYPE_DEMANDACTIVEPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea (1002)

ea 03 01 00 -> TS_DEMAND_ACTIVE_PDU::shareId
04 00 -> TS_DEMAND_ACTIVE_PDU::lengthSourceDescriptor = 4 bytes
51 01 -> TS_DEMAND_ACTIVE_PDU::lengthCombinedCapabilities = 0x151 = 337 bytes

52 44 50 00 -> TS_DEMAND_ACTIVE_PDU::sourceDescriptor = "RDP"

0d 00 -> TS_DEMAND_ACTIVE_PDU::numberCapabilities = 13
00 00 -> TS_DEMAND_ACTIVE_PDU::pad2octets

Share Capability Set (8 bytes)
09 00 08 00 ea 03 dc e2
```

```
09 00 -> TS_SHARE_CAPABILITYSET::capabilitySetType = CAPSTYPE_SHARE
(9)
08 00 -> TS_SHARE_CAPABILITYSET::lengthCapability = 8 bytes
ea 03 -> TS_SHARE_CAPABILITYSET::nodeID = 0x03ea (1002)
dc e2 -> TS_SHARE_CAPABILITYSET::pad2octets

General Capability Set (24 bytes)
01 00 18 00 01 00 03 00 00 02 00 00 00 00 1d 04
00 00 00 00 00 00 01 01

01 00 -> TS_GENERAL_CAPABILITYSET::capabilitySetType = CAPSTYPE_GENERAL (1)
18 00 -> TS_GENERAL_CAPABILITYSET::lengthCapability = 24 bytes

01 00 -> TS_GENERAL_CAPABILITYSET::osMajorType = TS_OSMAJORTYPE_WINDOWS (1)
03 00 -> TS_GENERAL_CAPABILITYSET::osMinorType = TS_OSMINORTYPE_WINDOWS_NT (3)
00 02 -> TS_GENERAL_CAPABILITYSET::protocolVersion = TS_CAPS_PROTOCOLVERSION (0x0200)
00 00 -> TS_GENERAL_CAPABILITYSET::pad2octetsA
00 00 -> TS_GENERAL_CAPABILITYSET::generalCompressionTypes = 0
1d 04 -> TS_GENERAL_CAPABILITYSET::extraFlags = 0x041d
0x041d
= 0x0400 |
  0x0010 |
  0x0008 |
  0x0004 |
  0x0001
= NO_BITMAP_COMPRESSION_HDR |
  ENC_SALTED_CHECKSUM |
  AUTORECONNECT_SUPPORTED |
  LONG_CREDENTIALS_SUPPORTED |
  FASTPATH_OUTPUT_SUPPORTED

00 00 -> TS_GENERAL_CAPABILITYSET::updateCapabilityFlag = 0
00 00 -> TS_GENERAL_CAPABILITYSET::remoteUnshareFlag = 0
00 00 -> TS_GENERAL_CAPABILITYSET::generalCompressionLevel = 0
01 -> TS_GENERAL_CAPABILITYSET::refreshRectSupport = TRUE
01 -> TS_GENERAL_CAPABILITYSET::suppressOutputSupport = TRUE

Virtual Channel Capability Set (8 bytes)
14 00 08 00 02 00 00 00

14 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::capabilitySetType =
CAPSTYPE_VIRTUALCHANNEL (20)
08 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::lengthCapability = 8 bytes

02 00 00 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::vccaps1 = 0x00000002
= VCCAPS_COMPR_CS_8K

DrawGdiPlus Capability Set (40 bytes)
16 00 28 00 00 00 00 00 70 f6 13 f3 01 00 00 00
01 00 00 00 18 00 00 00 9c f6 13 f3 61 a6 82 80
00 00 00 00 00 50 91 bf

16 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::capabilitySetType =
CAPSTYPE_DRAWGDIPLUS (22)
28 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::lengthCapability = 40 bytes

00 00 00 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::drawGdiplusSupportLevel
= TS_DRAW_GDIPLUS_DEFAULT (0)
70 f6 13 f3 -> TS_DRAW_GDIPLUS_CAPABILITYSET::GdipVersion
(uninitialized due to bug)
01 00 00 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::drawGdiplusCacheLevel
= TS_DRAW_GDIPLUS_CACHE_LEVEL_ONE (1)

Since GDI+ is not supported (see drawGdiplusSupportLevel), the
following unitialized cache fields can be ignored.
```

```
01 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipGraphicsCacheEntries
(uninitialized due to bug)
00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectBrushCacheEntries
(uninitialized due to bug)
18 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectPenCacheEntries
(uninitialized due to bug)
00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectImageCacheEntries
(uninitialized due to bug)
9c f6 ->
TS_GDIPLUS_CACHE_ENTRIES::GdipObjectImageAttributesCacheEntries
(uninitialized due to bug)

13 f3 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipGraphicsCacheChunkSize
(uninitialized due to bug)
61 a6 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectBrushCacheChunkSize
(uninitialized due to bug)
82 80 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectPenCacheChunkSize
(uninitialized due to bug)
00 00 ->
TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectImageAttributesCacheChunkSize
(uninitialized due to bug)

00 00 ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheChunkSize
(uninitialized due to bug)
00 50 ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheTotalSize
(uninitialized due to bug)
91 bf ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheMaxSize
(uninitialized due to bug)

Font Capability Set (4 bytes)
0e 00 04 00

0e 00 -> TS_FONT_CAPABILITYSET::capabilitySetType = CAPSTYPE_FONT (14)
04 00 -> TS_FONT_CAPABILITYSET::lengthCapability = 4 bytes

Due to a bug, the TS_FONT_CAPABILITYSET capability set size is
incorrectly set to 4 bytes (it should be 8 bytes). As a result of
this bug, the fontSupportFlags and pad2octets fields are missing.

Bitmap Capability Set (28 bytes)
02 00 1c 00 18 00 01 00 01 00 01 00 00 05 00 04
00 00 01 00 01 00 00 00 01 00 00 00

02 00 -> TS_BITMAP_CAPABILITYSET::capabilitySetType =
CAPSTYPE_BITMAP (2)
1c 00 -> TS_BITMAP_CAPABILITYSET::lengthCapability = 28 bytes

18 00 -> TS_BITMAP_CAPABILITYSET::preferredBitsPerPixel = 24 bpp
01 00 -> TS_BITMAP_CAPABILITYSET::receive1BitPerPixel = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::receive4BitsPerPixel = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::receive8BitsPerPixel = TRUE
00 05 -> TS_BITMAP_CAPABILITYSET::desktopWidth = 1280 pixels
00 04 -> TS_BITMAP_CAPABILITYSET::desktopHeight = 1024 pixels
00 00 -> TS_BITMAP_CAPABILITYSET::pad2octets
01 00 -> TS_BITMAP_CAPABILITYSET::desktopResizeFlag = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::bitmapCompressionFlag = TRUE
00 -> TS_BITMAP_CAPABILITYSET::highColorFlags = 0
00 -> TS_BITMAP_CAPABILITYSET::pad1octet
01 00 -> TS_BITMAP_CAPABILITYSET::multipleRectangleSupport = TRUE
00 00 -> TS_BITMAP_CAPABILITYSET::pad2octetsB
```

```
Order Capability Set (88 bytes)
03 00 58 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 40 42 0f 00 01 00 14 00 00 00 01 00
00 00 22 00 01 01 01 01 01 00 00 01 01 01 01 01
00 00 00 01 01 01 01 01 01 01 01 00 01 01 01 01
00 00 00 00 a1 06 00 00 40 42 0f 00 40 42 0f 00
01 00 00 00 00 00 00 00

03 00 -> TS_ORDER_CAPABILITYSET::capabilitySetType =
CAPSTYPE_ORDER (3)
58 00 -> TS_ORDER_CAPABILITYSET::lengthCapability = 88 bytes

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_ORDER_CAPABILITYSET::terminalDescriptor = ""
40 42 0f 00 -> TS_ORDER_CAPABILITYSET::pad4octetsA

01 00 -> TS_ORDER_CAPABILITYSET::desktopSaveXGranularity = 1
14 00 -> TS_ORDER_CAPABILITYSET::desktopSaveYGranularity = 20
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsA
01 00 -> TS_ORDER_CAPABILITYSET::maximumOrderLevel =
ORD_LEVEL_1_ORDERS (1)
00 00 -> TS_ORDER_CAPABILITYSET::numberFonts = 0

22 00 -> TS_ORDER_CAPABILITYSET::orderFlags = 0x0022
0x0022
= 0x0020 | 0x0002
= COLORINDEXSUPPORT | NEGOTIATEORDERSUPPORT

01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_DSTBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_PATBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_SCRBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEMBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEM3BLT_INDEX] =
TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ATEXTOUT_INDEX] =
FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_AEXTTEXTOUT_INDEX]
= FALSE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_DRAWNINEGRID_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_LINETO_INDEX] = TRUE
01 ->
TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTI_DRAWNINEGRID_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_OPAQUERECT_INDEX] =
TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_SAVEBITMAP_INDEX] =
TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WTEXTOUT_INDEX] =
FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEMBLT_R2_INDEX] =
FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEM3BLT_R2_INDEX] =
FALSE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIDSTBLT_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIPATBLT_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTISCRBLT_INDEX]
= TRUE
01 ->
TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIOPAQUERECT_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_FAST_INDEX_INDEX]
= TRUE
```

```
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYGON_SC_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYGON_CB_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYLINE_INDEX] =
TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[23] = 0
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_FAST_GLYPH_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ELLIPSE_SC_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ELLIPSE_CB_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_INDEX_INDEX] = TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WEXTTEXTOUT_INDEX]
= FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WLONGTEXTOUT_INDEX]
= FALSE
00 ->
TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WLONGEXTTEXTOUT_INDEX]
= FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[24] = 0

a1 06 -> TS_ORDER_CAPABILITYSET::textFlags = 0x06a1
0x6a1
= 0x400 |
  0x200 |
  0x080 |
  0x020 |
  0x001
= TS_TEXTFLAGS_ALLOWCELLHEIGHT |
  TS_TEXTFLAGS_USEBASELINESTART |
  TS_TEXTFLAGS_CHECKFONTSIGNATURES |
  TS_TEXTFLAGS_ALLOWDELTAXSIM |
  TS_TEXTFLAGS_CHECKFONTASPECT

00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsB
40 42 0f 00 -> TS_ORDER_CAPABILITYSET::pad4octetsB
40 42 0f 00 -> TS_ORDER_CAPABILITYSET::desktopSaveSize = 0xf4240
= 1000000
01 00 -> TS_ORDER_CAPABILITYSET::pad2octetsC
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsD
00 00 -> TS_ORDER_CAPABILITYSET::textANSICodePage
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsE

Color Table Cache Capability Set (8 bytes)
0a 00 08 00 06 00 00 00

0a 00 -> TS_COLORTABLECACHE_CAPABILITYSET::capabilitySetType =
CAPSTYPE_COLORCACHE (10)
08 00 -> TS_COLORTABLECACHE_CAPABILITYSET::lengthCapability = 8 bytes

06 00 -> TS_COLORTABLECACHE_CAPABILITYSET::colorTableCacheSize = 6
00 00 -> TS_COLORTABLECACHE_CAPABILITYSET::pad2octets

Bitmap Cache Host Support Capability Set (8 bytes)
12 00 08 00 01 00 00 00

12 00 -> TS_BITMAPCACHE_CAPABILITYSET_HOSTSUPPORT::capabilitySetType
= CAPSTYPE_BITMAPCACHE_HOSTSUPPORT (18)
08 00 -> TS_BITMAPCACHE_CAPABILITYSET_HOSTSUPPORT::lengthCapability
= 8 bytes

01 -> TS_BITMAPCACHE_CAPABILITYSET_HOSTSUPPORT::CacheVersion = 1
(corresponds to rev. 2 capabilities)
```

```
00 -> TS_BITMAPCACHE_CAPABILITYSET_HOSTSUPPORT::Pad1
00 00 -> TS_BITMAPCACHE_CAPABILITYSET_HOSTSUPPORT::Pad2

Pointer Capability Set (10 bytes)
08 00 0a 00 01 00 19 00 19 00

08 00 -> TS_POINTER_CAPABILITYSET::capabilitySetType =
CAPSTYPE_POINTER (8)
0a 00 -> TS_POINTER_CAPABILITYSET::lengthCapability = 10 bytes

01 00 -> TS_POINTER_CAPABILITYSET::colorPointerFlag = TRUE
19 00 -> TS_POINTER_CAPABILITYSET::colorPointerCacheSize = 25
19 00 -> TS_POINTER_CAPABILITYSET::pointerCacheSize = 25

Input Capability Set (88 bytes)
0d 00 58 00 35 00 00 00 a1 06 00 00 40 42 0f 00
0c f6 13 f3 93 5a 37 f3 00 90 30 e1 34 1c 38 f3
40 f6 13 f3 04 00 00 00 4c 54 dc e2 08 50 dc e2
01 00 00 00 08 50 dc e2 00 00 00 00 38 f6 13 f3
2e 05 38 f3 08 50 dc e2 2c f6 13 f3 00 00 00 00
08 00 0a 00 01 00 19 00

0d 00 -> TS_INPUT_CAPABILITYSET::capabilitySetType = CAPSTYPE_INPUT
(13)
58 00 -> TS_INPUT_CAPABILITYSET::lengthCapability = 88 bytes

35 00 -> TS_INPUT_CAPABILITYSET::inputFlags = 0x0035
0x0035
= 0x0020 |
  0x0010 |
  0x0004 |
  0x0001
= INPUT_FLAG_FASTPATH_INPUT2 |
  INPUT_FLAG_VKPACKET |
  INPUT_FLAG_MOUSEX |
  INPUT_FLAG_SCANCODES

00 00 -> TS_INPUT_CAPABILITYSET::pad2octetsA
a1 06 00 00 -> TS_INPUT_CAPABILITYSET::keyboardLayout (uninitialized
due to bug)
40 42 0f 00 -> TS_INPUT_CAPABILITYSET::keyboardType (uninitialized
due to bug)
0c f6 13 f3 -> TS_INPUT_CAPABILITYSET::keyboardSubType (
uninitialized due to bug)
93 5a 37 f3 -> TS_INPUT_CAPABILITYSET::keyboardFunctionKey
(uninitialized due to bug)

00 90 30 e1 34 1c 38 f3 40 f6 13 f3 04 00 00 00
4c 54 dc e2 08 50 dc e2 01 00 00 00 08 50 dc e2
00 00 00 00 38 f6 13 f3 2e 05 38 f3 08 50 dc e2
2c f6 13 f3 00 00 00 00 08 00 0a 00 01 00 19 00 ->
TS_INPUT_CAPABILITYSET::imeFileName (uninitialized due to bug)

RAIL Capability Set (8 bytes)
17 00 08 00 00 00 00 00

17 00 -> TS_RAIL_CAPABILITYSET::capabilitySetType = CAPSTYPE_RAIL (23)
08 00 -> TS_RAIL_CAPABILITYSET::lengthCapability = 8 bytes

00 00 00 00 -> TS_RAIL_CAPABILITYSET::railSupportLevel =
TS_RAIL_LEVEL_DEFAULT (0)

Windowing Capability Set (11 bytes)
18 00 0b 00 00 00 00 00 00 00 00
```

```
18 00 -> TS_WINDOW_CAPABILITYSET::capabilitySetType =
CAPSTYPE_WINDOW (24)
0b 00 -> TS_WINDOW_CAPABILITYSET::lengthCapability = 11 bytes

00 00 00 00 -> TS_WINDOW_CAPABILITYSET::wndSupportLevel =
TS_WINDOW_LEVEL_DEFAULT (0)
00 -> TS_WINDOW_CAPABILITYSET::nIconCaches = 0
00 00 -> TS_WINDOW_CAPABILITYSET::nIconCacheEntries = 0

Remainder of Demand Active PDU:

00 00 00 00 -> TS_DEMAND_ACTIVE_PDU::sessionId = 0
```

4.1.18 Client Confirm Active PDU

The following is an annotated dump of the Client Confirm Active PDU (section 2.2.1.13.2).

```
00000000 03 00 02 07 02 f0 80 64 00 06 03 eb 70 81 f8 38 .......d....p..8
00000010 00 00 00 ab 1f 51 e7 93 17 5c 45 04 36 38 41 80 .....Q...\E.68A.
00000020 2f ad d4 d3 48 e9 88 84 05 f4 3f c4 d1 e8 9d 92 /...H.....?.....
00000030 85 ac e6 fd 25 30 6d b5 fe 0e 4b 72 e3 f4 15 9f ....%0m...Kr....
00000040 2a 01 6e 44 15 d1 b4 1b f6 96 36 40 63 39 6f 73 *.nD......6@c9os
00000050 fc 93 57 b2 a7 f8 df 44 e5 23 5d 2f 57 4a e2 df ..W....D.#]/WJ..
00000060 aa 2d bc 99 4c fd 78 e1 a4 df 57 71 07 1e d4 99 .-..L.x...Wq....
00000070 59 c8 4d ae 4f 00 90 de 56 63 3a 8c cc ca 40 60 Y.M.O...Vc:...@`
00000080 2b ae 74 c5 e2 70 e9 bb 5e 0b c6 e8 82 21 cc a3 +.t..p..^....!..
00000090 e9 61 4c 6e db 76 7a fc a4 cc 57 a5 94 d5 96 5c .aLn.vz...W....\
000000a0 b2 99 1a 2a 84 52 84 97 35 54 6b c9 7d 3e f0 c8 ...*.R..5Tk.}>..
000000b0 3c e4 3d 44 79 76 07 e6 3f 20 1d 66 2c c9 0f d2 <.=Dyv..? .f,...
000000c0 cd 3d bf 25 38 7b cd 10 7c d7 2d da 72 8b db de .=.%8{..|.-.r...
000000d0 b8 97 00 11 14 dd 22 b5 a0 b9 19 7b e5 9d e1 90 ......"....{....
000000e0 72 5f 5a 5a 48 59 a8 67 68 b5 e6 95 70 e9 d3 19 r_ZZHY.gh...p...
000000f0 4f bd d9 1c 09 03 ac fa 6e 4b f5 0a 1e 21 a6 2f O.......nK...!./
00000100 57 c0 70 80 fc a1 0f 12 58 fe 0a 89 ca fc ff cf W.p.....X.......
00000110 37 04 b1 12 fd d2 03 30 b4 c7 fe a1 ad 5e 2b 8d 7......0.....^+.
00000120 21 3d 18 6e 0c b0 18 c4 78 33 06 f0 14 67 7a 7d !=.n....x3...gz}
00000130 09 1c 6e 66 57 00 db be 95 ef bf c2 1a a7 11 5e ..nfW..........^
00000140 d2 d3 36 c8 13 8d 64 ed 0f a3 bf ce c2 6f 8e e4 ..6...d......o..
00000150 11 4f 84 e5 c5 61 68 15 44 c5 5d 53 40 24 35 26 .O...ah.D.]S@$5&
00000160 20 21 a5 cf 11 6a a2 7a 6c 3e 36 d5 93 a1 f9 5e  !...j.zl>6....^
00000170 df e6 a5 2c 94 4f 1a 22 9f 7d fd 24 b4 06 7d 70 ...,.O.".}.$..}p
00000180 f0 49 ae 04 54 9d 14 73 48 27 57 e6 38 32 0e 31 .I..T..sH'W.82.1
00000190 c5 aa d5 c9 1c 82 0d ae 18 24 9c 18 90 b4 90 8d .........$......
000001a0 f1 bd 5f fb 10 c7 0b 01 fb bc 12 56 1d 30 19 c6 .._........V.0..
000001b0 90 a1 06 17 38 ed 0f 3c 62 1e 16 0d 87 b4 90 af ....8..<b.......
000001c0 ff 08 71 ff e9 25 19 8c d4 eb 7f b4 6a 43 d4 8b ..q..%......jC..
000001d0 05 43 b8 66 59 e2 1d 23 d8 92 14 9b 3c a7 07 40 .C.fY..#....<..@
000001e0 d6 30 7b 58 3e 6e 7f c8 12 15 bc eb 9f 74 8f 9c .0{X>n.......t..
000001f0 b3 8d e2 60 34 a3 3a 8f a0 34 42 b1 18 08 a0 c5 ...`4.:..4B.....
00000200 b5 97 44 ed b5 48 82                            ..D..H.

03 00 02 07 -> TPKT Header (length = 519 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 81 f8 -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x1f8 = 504 bytes
```

```
38 00 -> TS_SECURITY_HEADER::flags = 0x0038
0x0038
= 0x0010 | 0x0020 | 0x0008
= SEC_RESET_SEQNO | SEC_IGNORE_SEQNO | SEC_ENCRYPT

00 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
ab 1f 51 e7 93 17 5c 45 -> TS_SECURITY_HEADER1::dataSignature

04 36 38 41 80 2f ad d4 d3 48 e9 88 84 05 f4 3f
c4 d1 e8 9d 92 85 ac e6 fd 25 30 6d b5 fe 0e 4b
72 e3 f4 15 9f 2a 01 6e 44 15 d1 b4 1b f6 96 36
40 63 39 6f 73 fc 93 57 b2 a7 f8 df 44 e5 23 5d
2f 57 4a e2 df aa 2d bc 99 4c fd 78 e1 a4 df 57
71 07 1e d4 99 59 c8 4d ae 4f 00 90 de 56 63 3a
8c cc ca 40 60 2b ae 74 c5 e2 70 e9 bb 5e 0b c6
e8 82 21 cc a3 e9 61 4c 6e db 76 7a fc a4 cc 57
a5 94 d5 96 5c b2 99 1a 2a 84 52 84 97 35 54 6b
c9 7d 3e f0 c8 3c e4 3d 44 79 76 07 e6 3f 20 1d
66 2c c9 0f d2 cd 3d bf 25 38 7b cd 10 7c d7 2d
da 72 8b db de b8 97 00 11 14 dd 22 b5 a0 b9 19
7b e5 9d e1 90 72 5f 5a 5a 48 59 a8 67 68 b5 e6
95 70 e9 d3 19 4f bd d9 1c 09 03 ac fa 6e 4b f5
0a 1e 21 a6 2f 57 c0 70 80 fc a1 0f 12 58 fe 0a
89 ca fc ff cf 37 04 b1 12 fd d2 03 30 b4 c7 fe
a1 ad 5e 2b 8d 21 3d 18 6e 0c b0 18 c4 78 33 06
f0 14 67 7a 7d 09 1c 6e 66 57 00 db be 95 ef bf
c2 1a a7 11 5e d2 d3 36 c8 13 8d 64 ed 0f a3 bf
ce c2 6f 8e e4 11 4f 84 e5 c5 61 68 15 44 c5 5d
53 40 24 35 26 20 21 a5 cf 11 6a a2 7a 6c 3e 36
d5 93 a1 f9 5e df e6 a5 2c 94 4f 1a 22 9f 7d fd
24 b4 06 7d 70 f0 49 ae 04 54 9d 14 73 48 27 57
e6 38 32 0e 31 c5 aa d5 c9 1c 82 0d ae 18 24 9c
18 90 b4 90 8d f1 bd 5f fb 10 c7 0b 01 fb bc 12
56 1d 30 19 c6 90 a1 06 17 38 ed 0f 3c 62 1e 16
0d 87 b4 90 af ff 08 71 ff e9 25 19 8c d4 eb 7f
b4 6a 43 d4 8b 05 43 b8 66 59 e2 1d 23 d8 92 14
9b 3c a7 07 40 d6 30 7b 58 3e 6e 7f c8 12 15 bc
eb 9f 74 8f 9c b3 8d e2 60 34 a3 3a 8f a0 34 42
b1 18 08 a0 c5 b5 97 44 ed b5 48 82 ->
Encrypted TS_CONFIRM_ACTIVE_PDU

Decrypted TS_CONFIRM_ACTIVE_PDU:
00000000 ec 01 13 00 ef 03 ea 03 01 00 ea 03 06 00 d6 01 ................
00000010 00 20 73 25 7b e6 12 00 00 00 01 00 18 00 01 00 . s%{...........
00000020 03 00 00 02 00 00 00 00 1d 04 00 00 00 00 00 00 ................
00000030 00 00 02 00 1c 00 18 00 01 00 01 00 01 00 00 05 ................
00000040 00 04 00 00 01 00 01 00 00 00 01 00 00 00 03 00 ................
00000050 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 X...............
00000060 00 00 00 00 00 00 01 00 14 00 00 00 01 00 00 00 ................
00000070 2a 00 01 01 01 01 01 00 00 01 01 01 00 01 00 00 *...............
00000080 00 01 01 01 01 01 01 01 01 00 01 01 01 00 00 00 ................
00000090 00 00 a1 06 00 00 00 00 00 00 00 84 03 00 00 00 ................
000000a0 00 00 e4 04 00 00 13 00 28 00 03 00 00 03 78 00 ........(.....x.
000000b0 00 00 78 00 00 00 fb 09 00 80 00 00 00 00 00 00 ..x.............
000000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0a 00 ................
000000d0 08 00 06 00 00 00 07 00 0c 00 00 00 00 00 00 00 ................
000000e0 00 00 05 00 0c 00 00 00 00 00 02 00 02 00 08 00 ................
000000f0 0a 00 01 00 14 00 15 00 09 00 08 00 00 00 00 00 ................
00000100 0d 00 58 00 15 00 20 00 09 04 00 00 04 00 00 00 ..X... .........
00000110 00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000150 00 00 00 00 00 00 00 00 0c 00 08 00 01 00 00 00 ................
```

```
00000160 0e 00 08 00 01 00 00 00 10 00 34 00 fe 00 04 00 ..........4.....
00000170 fe 00 04 00 fe 00 08 00 fe 00 08 00 fe 00 10 00 ................
00000180 fe 00 20 00 fe 00 40 00 fe 00 80 00 fe 00 00 01 .. ...@.........
00000190 40 00 00 08 00 01 00 01 03 00 00 00 0f 00 08 00 @...............
000001a0 01 00 00 00 11 00 0c 00 01 00 00 00 00 1e 64 00 ..............d.
000001b0 14 00 08 00 01 00 00 00 15 00 0c 00 02 00 00 00 ................
000001c0 00 0a 00 01 16 00 28 00 00 00 00 00 00 00 00 00 ......(.........
000001d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001e0 00 00 00 00 00 00 00 00 00 00 00 00             ............

ec 01 -> TS_SHARECONTROLHEADER::totalLength = 0x01ec = 492 bytes
13 00 -> TS_SHARECONTROLHEADER::pduType = 0x0013
0x0013
= 0x0010 | 0x0003
= TS_PROTOCOL_VERSION | PDUTYPE_CONFIRMACTIVEPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef (1007)
ea 03 01 00 -> TS_CONFIRM_ACTIVE_PDU::shareID = 0x000103ea
ea 03 -> TS_CONFIRM_ACTIVE_PDU::originatorID = 0x03ea (1002)
06 00 -> TS_CONFIRM_ACTIVE_PDU::lengthSourceDescriptor = 6 bytes
d6 01 -> TS_CONFIRM_ACTIVE_PDU::lengthCombinedCapabilities = 0x1d6 = 470 bytes

00 20 73 25 7b e6 -> TS_CONFIRM_ACTIVE_PDU::sourceDescriptor = ""
(uninitialized due to bug - should be "MSTSC")

12 00 -> TS_CONFIRM_ACTIVE_PDU::numberCapabilities = 18
00 00 -> TS_CONFIRM_ACTIVE_PDU::pad2octets

General Capability Set (24 bytes)
01 00 18 00 01 00 03 00 00 02 00 00 00 00 1d 04
00 00 00 00 00 00 00 00

01 00 -> TS_GENERAL_CAPABILITYSET::capabilitySetType = CAPSTYPE_GENERAL (1)
18 00 -> TS_GENERAL_CAPABILITYSET::lengthCapability = 24 bytes

01 00 -> TS_GENERAL_CAPABILITYSET::osMajorType = TS_OSMAJORTYPE_WINDOWS (1)
03 00 -> TS_GENERAL_CAPABILITYSET::osMinorType = TS_OSMINORTYPE_WINDOWS_NT (3)
00 02 -> TS_GENERAL_CAPABILITYSET::protocolVersion = TS_CAPS_PROTOCOLVERSION (0x0200)
00 00 -> TS_GENERAL_CAPABILITYSET::pad2octetsA
00 00 -> TS_GENERAL_CAPABILITYSET::generalCompressionTypes = 0

1d 04 -> TS_GENERAL_CAPABILITYSET::extraFlags = 0x041d
0x041d
= 0x0400 |
  0x0010 |
  0x0008 |
  0x0004 |
  0x0001
= NO_BITMAP_COMPRESSION_HDR |
  ENC_SALTED_CHECKSUM |
  AUTORECONNECT_SUPPORTED |
  LONG_CREDENTIALS_SUPPORTED |
  FASTPATH_OUTPUT_SUPPORTED

00 00 -> TS_GENERAL_CAPABILITYSET::updateCapabilityFlag = 0
00 00 -> TS_GENERAL_CAPABILITYSET::remoteUnshareFlag = 0
00 00 -> TS_GENERAL_CAPABILITYSET::generalCompressionLevel = 0

00 -> TS_GENERAL_CAPABILITYSET::refreshRectSupport = FALSE
00 -> TS_GENERAL_CAPABILITYSET::suppressOutputSupport = FALSE

Bitmap Capability Set (28 bytes)
02 00 1c 00 18 00 01 00 01 00 01 00 00 05 00 04
00 00 01 00 01 00 00 00 01 00 00 00
```

```
02 00 -> TS_BITMAP_CAPABILITYSET::capabilitySetType = CAPSTYPE_BITMAP (2)
1c 00 -> TS_BITMAP_CAPABILITYSET::lengthCapability = 28 bytes

18 00 -> TS_BITMAP_CAPABILITYSET::preferredBitsPerPixel = 24 bpp
01 00 -> TS_BITMAP_CAPABILITYSET::receive1BitPerPixel = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::receive4BitsPerPixel = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::receive8BitsPerPixel = TRUE
00 05 -> TS_BITMAP_CAPABILITYSET::desktopWidth = 1280 pixels
00 04 -> TS_BITMAP_CAPABILITYSET::desktopHeight = 1024 pixels
00 00 -> TS_BITMAP_CAPABILITYSET::pad2octets
01 00 -> TS_BITMAP_CAPABILITYSET::desktopResizeFlag = TRUE
01 00 -> TS_BITMAP_CAPABILITYSET::bitmapCompressionFlag = TRUE
00 -> TS_BITMAP_CAPABILITYSET::highColorFlags = 0
00 -> TS_BITMAP_CAPABILITYSET::pad1octet
01 00 -> TS_BITMAP_CAPABILITYSET::multipleRectangleSupport = TRUE
00 00 -> TS_BITMAP_CAPABILITYSET::pad2octetsB

Order Capability Set (88 bytes)
03 00 58 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 14 00 00 00 01 00
00 00 2a 00 01 01 01 01 01 00 00 01 01 01 00 01
00 00 00 01 01 01 01 01 01 01 01 00 01 01 01 00
00 00 00 00 a1 06 00 00 00 00 00 00 00 84 03 00
00 00 00 00 e4 04 00 00

03 00 -> TS_ORDER_CAPABILITYSET::capabilitySetType = CAPSTYPE_ORDER (3)
58 00 -> TS_ORDER_CAPABILITYSET::lengthCapability = 88 bytes

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_ORDER_CAPABILITYSET::terminalDescriptor = ""
00 00 00 00 -> TS_ORDER_CAPABILITYSET::pad4octetsA

01 00 -> TS_ORDER_CAPABILITYSET::desktopSaveXGranularity = 1
14 00 -> TS_ORDER_CAPABILITYSET::desktopSaveYGranularity = 20
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsA
01 00 -> TS_ORDER_CAPABILITYSET::maximumOrderLevel = ORD_LEVEL_1_ORDERS (1)
00 00 -> TS_ORDER_CAPABILITYSET::numberFonts = 0

2a 00 -> TS_ORDER_CAPABILITYSET::orderFlags = 0x002a
0x002a
= 0x0020 |
  0x0008 |
  0x0002
= COLORINDEXSUPPORT |
  ZEROBOUNDSDELTASSUPPORT |
  NEGOTIATEORDERSUPPORT

01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_DSTBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_PATBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_SCRBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEMBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEM3BLT_INDEX] =
TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ATEXTOUT_INDEX] = FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_AEXTTEXTOUT_INDEX] = FALSE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_DRAWNINEGRID_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_LINETO_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTI_DRAWNINEGRID_INDEX] = TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_OPAQUERECT_INDEX] = FALSE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_SAVEBITMAP_INDEX] = TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WTEXTOUT_INDEX] = FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEMBLT_R2_INDEX] = FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MEM3BLT_R2_INDEX] = FALSE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIDSTBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIPATBLT_INDEX] = TRUE
```

```
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTISCRBLT_INDEX] = TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_MULTIOPAQUERECT_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_FAST_INDEX_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYGON_SC_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYGON_CB_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_POLYLINE_INDEX]
= TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[23] = 0
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_FAST_GLYPH_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ELLIPSE_SC_INDEX]
= TRUE
01 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_ELLIPSE_CB_INDEX]
= TRUE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_INDEX_INDEX] = FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WEXTTEXTOUT_INDEX]
= FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WLONGTEXTOUT_INDEX]
= FALSE
00 ->
TS_ORDER_CAPABILITYSET::orderSupport[TS_NEG_WLONGEXTTEXTOUT_INDEX]
= FALSE
00 -> TS_ORDER_CAPABILITYSET::orderSupport[24] = 0

a1 06 -> TS_ORDER_CAPABILITYSET::textFlags = 0x06a1
0x6a1
= 0x400 |
  0x200 |
  0x080 |
  0x020 |
  0x001
= TS_TEXTFLAGS_ALLOWCELLHEIGHT |
  TS_TEXTFLAGS_USEBASELINESTART |
  TS_TEXTFLAGS_CHECKFONTSIGNATURES |
  TS_TEXTFLAGS_ALLOWDELTAXSIM |
  TS_TEXTFLAGS_CHECKFONTASPECT

00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsB
00 00 00 00 -> TS_ORDER_CAPABILITYSET::pad4octetsB
00 84 03 00 -> TS_ORDER_CAPABILITYSET::desktopSaveSize = 0x38400 =
230400
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsC
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsD
e4 04 -> TS_ORDER_CAPABILITYSET::textANSICodePage = 0x04e4 = ANSI -
Latin I (1252)
00 00 -> TS_ORDER_CAPABILITYSET::pad2octetsE

Bitmap Cache Rev. 2 Capability Set (40 bytes)
13 00 28 00 03 00 00 03 78 00 00 00 78 00 00 00
fb 09 00 80 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

13 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::capabilitySetType =
CAPSTYPE_BITMAPCACHE_REV2 (19)
28 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::lengthCapability =
40 bytes

03 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CacheFlags = = 0x0003
0x0003
= 0x0001 | 0x0002
= PERSISTENT_KEYS_EXPECTED_FLAG | ALLOW_CACHE_WAITING_LIST_FLAG
```

```
00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::Pad2
03 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::NumCellCaches = 3

78 00 00 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CellCacheInfo[0] =
0x00000078
TS_BITMAPCACHE_CELL_CACHE_INFO::NumEntries = 0x78 = 120
TS_BITMAPCACHE_CELL_CACHE_INFO::k = FALSE

78 00 00 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CellCacheInfo[1] =
0x00000078
TS_BITMAPCACHE_CELL_CACHE_INFO::NumEntries = 0x78 = 120
TS_BITMAPCACHE_CELL_CACHE_INFO::k = FALSE

fb 09 00 80 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CellCacheInfo[2] =
0x800009fb
TS_BITMAPCACHE_CELL_CACHE_INFO::NumEntries = 0x9fb = 2555
TS_BITMAPCACHE_CELL_CACHE_INFO::k = TRUE

00 00 00 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CellCacheInfo[3] =
0x00000000
00 00 00 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::CellCacheInfo[4] =
0x00000000

00 00 00 00 00 00 00 00 00 00 00 00 -> TS_BITMAPCACHE_CAPABILITYSET_REV2::pad3

Color Table Cache Capability Set (8 bytes)
0a 00 08 00 06 00 00 00

0a 00 -> TS_COLORTABLECACHE_CAPABILITYSET::capabilitySetType =
CAPSTYPE_COLORCACHE (10)
08 00 -> TS_COLORTABLECACHE_CAPABILITYSET::lengthCapability = 8 bytes

06 00 -> TS_COLORTABLECACHE_CAPABILITYSET::colorTableCacheSize = 6
00 00 -> TS_COLORTABLECACHE_CAPABILITYSET::pad2octets = 0

Window Activation Capability Set (12 bytes)
07 00 0c 00 00 00 00 00 00 00 00 00

07 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::capabilitySetType =
CAPSTYPE_ACTIVATION (7)
0c 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::lengthCapability =
12 bytes

00 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::helpKeyFlag = 0
00 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::helpKeyIndexFlag = 0
00 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::helpExtendedKeyFlag = 0
00 00 -> TS_WINDOWACTIVATION_CAPABILITYSET::windowManagerKeyFlag = 0

Control Capability Set (12 bytes)
05 00 0c 00 00 00 00 00 02 00 02 00

05 00 -> TS_CONTROL_CAPABILITYSET::capabilitySetType =
CAPSTYPE_CONTROL (5)
0c 00 -> TS_CONTROL_CAPABILITYSET::lengthCapability = 12 bytes

00 00 -> TS_CONTROL_CAPABILITYSET::controlFlags = 0
00 00 -> TS_CONTROL_CAPABILITYSET::remoteDetachFlag = 0
02 00 -> TS_CONTROL_CAPABILITYSET::controlInterest =
CONTROLPRIORITY_NEVER (2)
02 00 -> TS_CONTROL_CAPABILITYSET::detachInterest =
CONTROLPRIORITY_NEVER (2)

Pointer Capability Set (10 bytes)
08 00 0a 00 01 00 14 00 15 00
```

```
08 00 -> TS_POINTER_CAPABILITYSET::capabilitySetType =
CAPSTYPE_POINTER (8)
0a 00 -> TS_POINTER_CAPABILITYSET::lengthCapability = 10 bytes

01 00 -> TS_POINTER_CAPABILITYSET::colorPointerFlag = TRUE
14 00 -> TS_POINTER_CAPABILITYSET::colorPointerCacheSize = 20
15 00 -> TS_POINTER_CAPABILITYSET::pointerCacheSize = 21

Share Capability Set (8 bytes)
09 00 08 00 00 00 00 00

09 00 -> TS_SHARE_CAPABILITYSET::capabilitySetType =
CAPSTYPE_SHARE (9)
08 00 -> TS_SHARE_CAPABILITYSET::lengthCapability = 8 bytes

00 00 -> TS_SHARE_CAPABILITYSET::nodeID = 0
00 00 -> TS_SHARE_CAPABILITYSET::pad2octets

Input Capability Set (88 bytes)
0d 00 58 00 15 00 20 00 09 04 00 00 04 00 00 00
00 00 00 00 0c 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

0d 00 -> TS_INPUT_CAPABILITYSET::capabilitySetType =
CAPSTYPE_INPUT (13)
58 00 -> TS_INPUT_CAPABILITYSET::lengthCapability = 88 bytes

0d 00 -> TS_INPUT_CAPABILITYSET::capabilitySetType =
CAPSTYPE_INPUT (13)
58 00 -> TS_INPUT_CAPABILITYSET::lengthCapability = 88 bytes

15 00 -> TS_INPUT_CAPABILITYSET::inputFlags = 0x0015
0x0015
= 0x0010 |
  0x0004 |
  0x0001
= INPUT_FLAG_VKPACKET |
  INPUT_FLAG_MOUSEX |
  INPUT_FLAG_SCANCODES

20 00 -> TS_INPUT_CAPABILITYSET::pad2octetsA
09 04 00 00 -> TS_INPUT_CAPABILITYSET::keyboardLayout = 0x00000409 =
English (United States)
04 00 00 00 -> TS_INPUT_CAPABILITYSET::keyboardType = 4 =
IBM enhanced (101- or 102-key) keyboard
00 00 00 00 -> TS_INPUT_CAPABILITYSET::keyboardSubType = 0
0c 00 00 00 -> TS_INPUT_CAPABILITYSET::keyboardFunctionKey = 0x0c =
12

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ->
TS_INPUT_CAPABILITYSET::imeFileName

Sound Capability Set (8 bytes)
0c 00 08 00 01 00 00 00

0c 00 -> TS_SOUND_CAPABILITYSET::capabilitySetType = CAPSTYPE_SOUND
(12)
08 00 -> TS_SOUND_CAPABILITYSET::lengthCapability = 8 bytes
```

```
01 00 -> TS_SOUND_CAPABILITYSET::soundFlags = 0x0001 =
SOUND_FLAG_BEEPS
00 00 -> TS_SOUND_CAPABILITYSET::pad2octetsA

Font Capability Set (8 bytes)
0e 00 08 00 01 00 00 00

0e 00 -> TS_FONT_CAPABILITYSET::capabilitySetType = CAPSTYPE_FONT (14)
08 00 -> TS_FONT_CAPABILITYSET::lengthCapability = 8 bytes

01 00 -> TS_FONT_CAPABILITYSET::fontSupportFlags = 0x0001 =
FONTSUPPORT_FONTLIST
00 00 -> TS_FONT_CAPABILITYSET::pad2octets

Glyphe Cache Capability Set (52 bytes)
10 00 34 00 fe 00 04 00 fe 00 04 00 fe 00 08 00
fe 00 08 00 fe 00 10 00 fe 00 20 00 fe 00 40 00
fe 00 80 00 fe 00 00 01 40 00 00 08 00 01 00 01
03 00 00 00

10 00 -> TS_GLYPHCACHE_CAPABILITYSET::capabilitySetType =
CAPSTYPE_GLYPHCACHE (16)
34 00 -> TS_GLYPHCACHE_CAPABILITYSET::lengthCapability = 52 bytes

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[0]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
04 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 4

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[1]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
04 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 4

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[2]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
08 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 8

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[3]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
08 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 8

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[4]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
10 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 16

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[5]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
20 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 32

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[6]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
40 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 64

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[7]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
80 00 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 128

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[8]:
fe 00 -> TS_CACHE_DEFINITION::CacheEntries = 254
00 01 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 256

TS_GLYPHCACHE_CAPABILITYSET::GlyphCache[9]:
40 00 -> TS_CACHE_DEFINITION::CacheEntries = 64
00 08 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 256
```

```
TS_GLYPHCACHE_CAPABILITYSET::FragCache:
00 01 -> TS_CACHE_DEFINITION::CacheEntries = 256
00 01 -> TS_CACHE_DEFINITION::CacheMaximumCellSize = 256

03 00 -> TS_GLYPHCACHE_CAPABILITYSET::GlyphSupportLevel =
GLYPH_SUPPORT_ENCODE (3)
00 00 -> TS_GLYPHCACHE_CAPABILITYSET::pad2octets

Brush Capability Set (8 bytes)
0f 00 08 00 01 00 00 00

0f 00 -> TS_BRUSH_CAPABILITYSET::capabilitySetType = CAPSTYPE_BRUSH
(15)
08 00 -> TS_BRUSH_CAPABILITYSET::lengthCapability = 8 bytes

01 00 00 00 -> TS_BRUSH_CAPABILITYSET::brushSupportLevel =
BRUSH_COLOR_8x8 (1)

Offscreen Bitmap Cache Capability Set (12 bytes)
11 00 0c 00 01 00 00 00 00 1e 64 00

11 00 -> TS_OFFSCREEN_CAPABILITYSET::capabilitySetType =
CAPSTYPE_OFFSCREENCACHE (17)
0c 00 -> TS_OFFSCREEN_CAPABILITYSET::lengthCapability = 12 bytes

01 00 00 00 -> TS_OFFSCREEN_CAPABILITYSET::offscreenSupportLevel =
TRUE (1)
00 1e -> TS_OFFSCREEN_CAPABILITYSET::offscreenCacheSize = 7680
64 00 -> TS_OFFSCREEN_CAPABILITYSET::offscreenCacheEntries = 100

Virtual Channel Capability Set (8 bytes)
14 00 08 00 01 00 00 00

14 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::capabilitySetType =
CAPSTYPE_VIRTUALCHANNEL (20)
08 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::lengthCapability = 8 bytes

01 00 00 00 -> TS_VIRTUALCHANNEL_CAPABILITYSET::vccaps1 =
0x00000001 = VCCAPS_COMPR_SC

DrawNineGridCache Capability Set (12 bytes)
15 00 0c 00 02 00 00 00 00 0a 00 01

15 00 -> TS_DRAW_NINEGRID_CAPABILITYSET::capabilitySetType =
CAPSTYPE_DRAWNINEGRIDCACHE (21)
0c 00 -> TS_DRAW_NINEGRID_CAPABILITYSET::lengthCapability = 12 bytes

02 00 00 00 ->
TS_DRAW_NINEGRID_CAPABILITYSET::drawNineGridSupportLevel =
DRAW_NINEGRID_SUPPORTED_REV2 (2)
00 0a -> TS_DRAW_NINEGRID_CAPABILITYSET::drawNineGridCacheSize = 2560
00 01 -> TS_DRAW_NINEGRID_CAPABILITYSET::drawNineGridCacheEntries =
256

DrawGdiPlus Capability Set (40 bytes)
16 00 28 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

16 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::capabilitySetType =
CAPSTYPE_DRAWGDIPLUS (22)
28 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::lengthCapability = 40 bytes

00 00 00 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::drawGdiplusSupportLevel
= TS_DRAW_GDIPLUS_DEFAULT (0)
```

```
00 00 00 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::GdipVersion = 0
01 00 00 00 -> TS_DRAW_GDIPLUS_CAPABILITYSET::drawGdiplusCacheLevel
= TS_DRAW_GDIPLUS_CACHE_LEVEL_DEFAULT (0)

00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipGraphicsCacheEntries
00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectBrushCacheEntries
00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectPenCacheEntries
00 00 -> TS_GDIPLUS_CACHE_ENTRIES::GdipObjectImageCacheEntries
00 00 ->
TS_GDIPLUS_CACHE_ENTRIES::GdipObjectImageAttributesCacheEntries

00 00 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipGraphicsCacheChunkSize
00 00 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectBrushCacheChunkSize
00 00 -> TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectPenCacheChunkSize
00 00 ->
TS_GDIPLUS_CACHE_CHUNK_SIZE::GdipObjectImageAttributesCacheChunkSize

00 00 ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheChunkSize
00 00 ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheTotalSize
00 00 ->
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES::GdipObjectImageCacheMaxSize
```

4.1.19 Client Synchronize PDU

The following is an annotated dump of the Client Synchronize PDU (section 2.2.1.14).

```
00000000 03 00 00 30 02 f0 80 64 00 06 03 eb 70 22 28 00 ...0...d....p"(.
00000010 81 f8 59 ff cb 2f 73 57 2b 42 db 88 2e 23 a9 97 ..Y../sW+B...#..
00000020 c2 b1 f5 74 bc 49 cc 8a d8 fd 60 8a 7a f6 44 75 ...t.I....`.z.Du

03 00 00 30 -> TPKT Header (length = 48 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 22 -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x22 = 34 bytes

28 00 -> TS_SECURITY_HEADER::flags = 0x0028
0x0028
= 0x0020 | 0x0008
= SEC_IGNORE_SEQNO | SEC_ENCRYPT

81 f8 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
59 ff cb 2f 73 57 2b 42 -> TS_SECURITY_HEADER1::dataSignature

db 88 2e 23 a9 97 c2 b1 f5 74 bc 49 cc 8a d8 fd
60 8a 7a f6 44 75 -> Encrypted TS_SYNCHRONIZE_PDU

Decrypted TS_SYNCHRONIZE_PDU:
00000000 16 00 17 00 ef 03 ea 03 01 00 00 01 08 00 1f 00 ................
00000010 00 00 01 00 ea 03                               ......

16 00 -> TS_SHARECONTROLHEADER::totalLength = 0x0016 = 22 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
```

```
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
08 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0008 = 8 bytes
1f -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SYNCHRONIZE (31)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0
```

4.1.20 Client Control PDU - Cooperate

The following is an annotated dump of the Client Control PDU - Cooperate (section 2.2.1.15).

```
00000000 03 00 00 34 02 f0 80 64 00 06 03 eb 70 26 08 00 ...4...d....p&..
00000010 81 f8 04 03 de f7 91 a3 7c af 3f 7a 62 4e 3b fe ........|.?zbN;.
00000020 b6 7a 28 bf 0d 4f 31 27 03 b9 4a f1 e6 26 f0 bd .z(..O1'..J..&..
00000030 c5 71 0a 53                                     .q.S

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x26 = 38 bytes

08 00 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
81 f8 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
04 03 de f7 91 a3 7c af -> TS_SECURITY_HEADER1::dataSignature

3f 7a 62 4e 3b fe b6 7a 28 bf 0d 4f 31 27 03 b9
4a f1 e6 26 f0 bd c5 71 0a 53 -> Encrypted TS_CONTROL_PDU

Decrypted TS_CONTROL_PDU:
00000000 1a 00 17 00 ef 03 ea 03 01 00 00 01 0c 00 14 00 ................
00000010 00 00 04 00 00 00 00 00 00 00                   ..........

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
0c 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x000c = 12 bytes
14 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_CONTROL (20)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

04 00 -> TS_CONTROL_PDU::action = CTRLACTION_COOPERATE (4)
00 00 -> TS_CONTROL_PDU::grantId = 0
00 00 00 00 -> TS_CONTROL_PDU::controlId = 0
```

4.1.21 Client Control PDU - Request Control

The following is an annotated dump of the Client Control PDU - Request Control (section 2.2.1.16).

```
00000000 03 00 00 34 02 f0 80 64 00 06 03 eb 70 26 08 00 ...4...d....p&..
00000010 81 f8 3b 8b b4 72 56 ff d1 d6 4b 17 1e ae f6 8d ..;..rV...K.....
00000020 dd 75 a0 a3 16 97 29 12 b7 cf 14 c9 11 0b d8 c8 .u....).........
00000030 fa a1 81 3a                                     ...:

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x26 = 38 bytes

08 00 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
81 f8 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
3b 8b b4 72 56 ff d1 d6 -> TS_SECURITY_HEADER1::dataSignature

4b 17 1e ae f6 8d dd 75 a0 a3 16 97 29 12 b7 cf
14 c9 11 0b d8 c8 fa a1 81 3a -> Encrypted TS_CONTROL_PDU

Decrypted TS_CONTROL_PDU:
00000000 1a 00 17 00 ef 03 ea 03 01 00 00 01 0c 00 14 00 ................
00000010 00 00 01 00 00 00 00 00 00 00                   ..........

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
0c 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x000c = 12 bytes
14 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_CONTROL (20)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

01 00 -> TS_CONTROL_PDU::action = CTRLACTION_REQUEST_CONTROL (1)
00 00 -> TS_CONTROL_PDU::grantId = 0
00 00 00 00 -> TS_CONTROL_PDU::controlId = 0
```

4.1.22 Client Persistent Key List PDU

The following is an annotated dump of the Client Persistent Key List PDU (section 2.2.1.17).

```
00000000 03 00 01 0d 02 f0 80 64 00 06 03 eb 70 80 fe 08 .......d....p...
00000010 00 90 16 ce c6 4a 69 d9 d3 49 9e 10 a5 04 0f cf .....Ji..I......
```

```
00000020 ab 4f 6a 3b da 31 03 4f 29 bd 64 3e 98 46 ec 0a .Oj;.1.O).d>.F..
00000030 1d cd 9c ad 13 58 a3 bd 8b 9d ae f1 e9 9d 43 96 .....X........C.
00000040 53 f5 d0 b7 50 88 f3 81 f1 cb ad 17 55 75 9c 5f S...P.......Uu._
00000050 ef ec a9 35 40 b3 74 06 d1 ae d1 15 9f ed 91 49 ...5@.t........I
00000060 a6 3d 1f c1 31 b1 17 58 da 0e 24 df 1f 87 86 39 .=..1..X..$....9
00000070 d1 46 66 ea 0e 98 d0 4b 5b 7b 01 b9 8a e8 68 32 .Ff....K[{....h2
00000080 80 da b9 58 a6 9f 4f b5 ba 79 04 ae d9 63 c0 6a ...X..O..y...c.j
00000090 a8 81 51 97 25 0b 3f c3 d2 47 fa 0a 7a 22 1f bd ..Q.%.?..G..z"..
000000a0 5f 4e b8 00 ea 32 06 e6 af 15 e4 6f b3 d3 c1 4c _N...2.....o...L
000000b0 cb 0a 8e dd a7 29 07 03 59 c1 c1 08 1b aa 56 3c .....)..Y.....V<
000000c0 f5 d0 89 e3 cd cf 26 8b 65 59 0a cb 7e 81 b6 33 ......&.eY..~..3
000000d0 bb 4d 9a 13 80 e7 57 2a 0d 1d 11 b4 18 c4 31 2f .M....W*......1/
000000e0 4f 89 77 09 94 2e c3 8e bf fd 6a 39 2b 47 74 0e O.w.......j9+Gt.
000000f0 12 74 ec 45 14 c3 6b 27 d6 b6 93 11 a4 bc 46 de .t.E..k'......F.
00000100 69 4a b4 54 c7 24 24 99 8f 60 b7 21 59          iJ.T.$$..`.!Y

03 00 01 0d -> TPKT Header (length = 269 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 80 fe -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0xfe = 254 bytes

08 00 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
90 16 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
ce c6 4a 69 d9 d3 49 9e -> TS_SECURITY_HEADER1::dataSignature

10 a5 04 0f cf ab 4f 6a 3b da 31 03 4f 29 bd 64
3e 98 46 ec 0a 1d cd 9c ad 13 58 a3 bd 8b 9d ae
f1 e9 9d 43 96 53 f5 d0 b7 50 88 f3 81 f1 cb ad
17 55 75 9c 5f ef ec a9 35 40 b3 74 06 d1 ae d1
15 9f ed 91 49 a6 3d 1f c1 31 b1 17 58 da 0e 24
df 1f 87 86 39 d1 46 66 ea 0e 98 d0 4b 5b 7b 01
b9 8a e8 68 32 80 da b9 58 a6 9f 4f b5 ba 79 04
ae d9 63 c0 6a a8 81 51 97 25 0b 3f c3 d2 47 fa
0a 7a 22 1f bd 5f 4e b8 00 ea 32 06 e6 af 15 e4
6f b3 d3 c1 4c cb 0a 8e dd a7 29 07 03 59 c1 c1
08 1b aa 56 3c f5 d0 89 e3 cd cf 26 8b 65 59 0a
cb 7e 81 b6 33 bb 4d 9a 13 80 e7 57 2a 0d 1d 11
b4 18 c4 31 2f 4f 89 77 09 94 2e c3 8e bf fd 6a
39 2b 47 74 0e 12 74 ec 45 14 c3 6b 27 d6 b6 93
11 a4 bc 46 de 69 4a b4 54 c7 24 24 99 8f 60 b7
21 59 -> Encrypted TS_BITMAPCACHE_PERSISTENT_LIST

Decrypted TS_BITMAPCACHE_PERSISTENT_LIST:
00000000 f2 00 17 00 ef 03 ea 03 01 00 00 01 00 00 2b 00 ..............+.
00000010 00 00 00 00 00 00 19 00 00 00 00 00 00 00 00 00 ................
00000020 19 00 00 00 00 00 03 00 00 00 a3 1e 51 16 48 29 ............Q.H)
00000030 22 78 61 f7 89 9c cd a9 66 a8 44 4e b7 bd b4 6d "xa.....f.DN...m
00000040 9e f6 39 91 64 af bc c3 70 02 9f aa fa fd 6e ba ..9.d...p.....n.
00000050 58 dc 7b af de 06 56 3a c2 ce 68 ba 54 b6 bf 9e X.{...V:..h.T...
00000060 bc d6 d1 22 c0 98 63 e9 41 fe 38 6c 50 35 0e db ..."..c.A.8lP5..
00000070 b3 f5 45 cc 18 2d 30 44 fc 88 e5 c3 5d 23 63 f6 ..E..-0D....]#c.
00000080 cf 53 0a a8 01 b6 10 51 a5 28 70 81 6c 59 19 29 .S.....Q.(p.lY.)
00000090 00 c9 e2 b5 e7 a7 46 04 4e 1b 72 8d 4a dd 81 bb ......F.N.r.J...
000000a0 14 16 53 6a 4e 3c 48 72 66 c9 6c 77 4b 4a 32 48 ..SjN<Hrf.lwKJ2H
000000b0 2c c6 02 54 56 f2 81 c9 85 56 2c 0a 3d 54 86 9d ,..TV....V,.=T..
000000c0 2b 97 63 0f 0a 36 f8 63 79 3e c9 70 41 4b ec a8 +.c..6.cy>.pAK..
000000d0 7c 7b 79 28 b6 b4 a6 43 24 de cb 9c ff a2 29 3c |{y(...C$.....)<
000000e0 02 56 64 df 80 b0 0d 6e e7 1a 83 c7 54 31 aa 8a .Vd....n....T1..
```

```
000000f0 90 b3                                          ..

f2 00 -> TS_SHARECONTROLHEADER::totalLength = 0x00f2 = 242 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
00 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0
2b -> TS_SHAREDATAHEADER::pduType2 =
PDUTYPE2_BITMAPCACHE_PERSISTENT_LIST (43)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::numEntries[0] = 0
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::numEntries[1] = 0
19 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::numEntries[2] = 0x19 = 25
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::numEntries[3] = 0
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::numEntries[4] = 0

00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::totalEntries[0] = 0
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::totalEntries[1] = 0
19 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::totalEntries[2] = 0x19 = 25
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::totalEntries[3] = 0
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::totalEntries[4] = 0

03 -> TS_BITMAPCACHE_PERSISTENT_LIST::bBitMask = 0x03
0x03
= 0x01 | 0x02
= PERSIST_FIRST_PDU | PERSIST_LAST_PDU

00 -> TS_BITMAPCACHE_PERSISTENT_LIST::Pad2
00 00 -> TS_BITMAPCACHE_PERSISTENT_LIST::Pad3

TS_BITMAPCACHE_PERSISTENT_LIST::entries:
a3 1e 51 16 -> Cache 2, Key 0, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
48 29 22 78 -> Cache 2, Key 0, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
61 f7 89 9c -> Cache 2, Key 1, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
cd a9 66 a8 -> Cache 2, Key 1, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
44 4e b7 bd -> Cache 2, Key 2, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
b4 6d 9e f6 -> Cache 2, Key 2, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
39 91 64 af -> Cache 2, Key 3, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
bc c3 70 02 -> Cache 2, Key 3, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
9f aa fa fd -> Cache 2, Key 4, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
6e ba 58 dc -> Cache 2, Key 4, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
7b af de 06 -> Cache 2, Key 5, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
56 3a c2 ce -> Cache 2, Key 5, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
68 ba 54 b6 -> Cache 2, Key 6, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
```

```
bf 9e bc d6 -> Cache 2, Key 6, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
d1 22 c0 98 -> Cache 2, Key 7, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
63 e9 41 fe -> Cache 2, Key 7, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
38 6c 50 35 -> Cache 2, Key 8, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
0e db b3 f5 -> Cache 2, Key 8, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
45 cc 18 2d -> Cache 2, Key 9, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
30 44 fc 88 -> Cache 2, Key 9, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
e5 c3 5d 23 -> Cache 2, Key 10, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
63 f6 cf 53 -> Cache 2, Key 10, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
0a a8 01 b6 -> Cache 2, Key 11, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
10 51 a5 28 -> Cache 2, Key 11, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
70 81 6c 59 -> Cache 2, Key 12, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
19 29 00 c9 -> Cache 2, Key 12, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
e2 b5 e7 a7 -> Cache 2, Key 13, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
46 04 4e 1b -> Cache 2, Key 13, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
72 8d 4a dd -> Cache 2, Key 14, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
81 bb 14 16 -> Cache 2, Key 14, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
53 6a 4e 3c -> Cache 2, Key 15, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
48 72 66 c9 -> Cache 2, Key 15, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
6c 77 4b 4a -> Cache 2, Key 16, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
32 48 2c c6 -> Cache 2, Key 16, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
02 54 56 f2 -> Cache 2, Key 17, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
81 c9 85 56 -> Cache 2, Key 17, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
2c 0a 3d 54 -> Cache 2, Key 18, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
86 9d 2b 97 -> Cache 2, Key 18, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
63 0f 0a 36 -> Cache 2, Key 19, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
f8 63 79 3e -> Cache 2, Key 19, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
c9 70 41 4b -> Cache 2, Key 20, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
ec a8 7c 7b -> Cache 2, Key 20, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
79 28 b6 b4 -> Cache 2, Key 21, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
a6 43 24 de -> Cache 2, Key 21, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
cb 9c ff a2 -> Cache 2, Key 22, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
29 3c 02 56 -> Cache 2, Key 22, High 32-bits
```

```
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
64 df 80 b0 -> Cache 2, Key 23, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
0d 6e e7 1a -> Cache 2, Key 23, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
83 c7 54 31 -> Cache 2, Key 24, Low 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key1)
aa 8a 90 b3 -> Cache 2, Key 24, High 32-bits
(TS_BITMAPCACHE_PERSISTENT_LIST_ENTRY::Key2)
```

4.1.23 Client Font List PDU

The following is an annotated dump of the Client Font List PDU (section 2.2.1.18).

```
00000000 03 00 00 34 02 f0 80 64 00 06 03 eb 70 26 08 00 ...4...d....p&..
00000010 80 fe 98 19 5c fb 92 92 f5 97 18 b2 b7 c3 13 dc ....\...........
00000020 03 fb 64 45 c0 43 6d 91 37 26 fd 8e 71 e6 f2 2a ..dE.Cm.7&..q..*
00000030 1e ae 35 03                                     ..5.

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x26 = 38 bytes

08 00 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
80 fe -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
98 19 5c fb 92 92 f5 97 -> TS_SECURITY_HEADER1::dataSignature

18 b2 b7 c3 13 dc 03 fb 64 45 c0 43 6d 91 37 26
fd 8e 71 e6 f2 2a 1e ae 35 03 -> Encrypted TS_FONT_LIST_PDU

Decrypted TS_FONT_LIST_PDU:
00000000 1a 00 17 00 ef 03 ea 03 01 00 00 01 3b da 27 00 ............;.'.
00000010 00 00 00 00 00 00 03 00 32 00                   ........2.

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
3b da -> TS_SHAREDATAHEADER::uncompressedLength (uninitialized due
to bug)
27 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_FONTLIST (39)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

00 00 -> TS_FONT_LIST_PDU::numberEntries = 0
00 00 -> TS_FONT_LIST_PDU::totalNumEntries = 0
03 00 -> TS_FONT_LIST_PDU::listFlags = 0x0003 = 0x0002 | 0x0001 =
FONTLIST_LAST | FONTLIST_FIRST
```

```
32 00 -> TS_FONT_LIST_PDU::entrySize = 0x0032 = 50 bytes
```

4.1.24 Server Synchronize PDU

The following is an annotated dump of the Server Synchronize PDU (section 2.2.1.19).

```
00000000 03 00 00 30 02 f0 80 68 00 01 03 eb 70 22 08 08 ...0...h....p"..
00000010 02 03 f4 4e d1 9e b4 53 b6 e6 d7 be cc c2 2b 18 ...N...S......+.
00000020 a2 cf 5c 9f 59 de c6 02 e2 ff 36 69 b7 ff 0e 27 ..\.Y.....6i...'

03 00 00 30 -> TPKT Header (length = 48 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 22 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x22 = 34 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
f4 4e d1 9e b4 53 b6 e6 -> TS_SECURITY_HEADER1::dataSignature

d7 be cc c2 2b 18 a2 cf 5c 9f 59 de c6 02 e2 ff
36 69 b7 ff 0e 27 -> Encrypted TS_SYNCHRONIZE_PDU

Decrypted TS_SYNCHRONIZE_PDU:
00000000 16 00 17 00 ea 03 ea 03 01 00 14 00 16 00 1f 00 ................
00000010 00 00 01 00 63 44                               ....cD

16 00 -> TS_SHARECONTROLHEADER::totalLength = 0x0016 = 22 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
14 -> TS_SHAREDATAHEADER::pad1
00 -> TS_SHAREDATAHEADER::streamId = STREAM_UNDEFINED (0)
16 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0016 = 22 bytes
1f -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SYNCHRONIZE (31)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

01 00 -> TS_SYNCHRONIZE_PDU::messageType = SYNCMSGTYPE_SYNC (1)
63 44 -> TS_SYNCHRONIZE_PDU::targetUser (uninitialized due to bug)
```

4.1.25 Server Control PDU - Cooperate

The following is an annotated dump of the Server Control PDU - Cooperate (section 2.2.1.20).

```
00000000 03 00 00 34 02 f0 80 68 00 01 03 eb 70 26 08 08 ...4...h....p&..
00000010 02 03 1c 2c 1b a6 84 ae 6d 6d 1f ad 25 6d 8b 61 ...,....mm..%m.a
00000020 11 f1 b2 0e 12 e6 e8 6b 43 af b0 4e c8 79 73 46 .......kC..N.ysF
00000030 31 ee 05 f9                                     1...

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x26 = 38 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
1c 2c 1b a6 84 ae 6d 6d -> TS_SECURITY_HEADER1::dataSignature

1f ad 25 6d 8b 61 11 f1 b2 0e 12 e6 e8 6b 43 af
b0 4e c8 79 73 46 31 ee 05 f9 -> Encrypted TS_CONTROL_PDU

Decrypted TS_CONTROL_PDU:
00000000 1a 00 17 00 ea 03 ea 03 01 00 b5 02 1a 00 14 00 ................
00000010 00 00 04 00 00 00 00 00 00 00                   ..........

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
b5 -> TS_SHAREDATAHEADER::pad1
02 -> TS_SHAREDATAHEADER::streamId = STREAM_MED (2)
1a 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x001a = 26 bytes
14 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_CONTROL (20)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

04 00 -> TS_CONTROL_PDU::action = CTRLACTION_COOPERATE (4)
00 00 -> TS_CONTROL_PDU::grantId = 0
00 00 00 00 -> TS_CONTROL_PDU::controlId = 0
```

4.1.26 Server Control PDU - Granted Control

The following is an annotated dump of the Server Control PDU - Granted Control.

```
00000000 03 00 00 34 02 f0 80 68 00 01 03 eb 70 26 08 08 ...4...h....p&..
00000010 02 03 c3 90 ba eb 39 68 dd ed 60 54 ad 97 a5 a5 ......9h..`T....
00000020 ec 44 e6 63 45 20 bd c9 66 4e 12 de 01 d3 3c 39 .D.cE ..fN....<9
00000030 09 0c 99 f8                                     ....

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU
```

```
68 00 01 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x26 = 38 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
c3 90 ba eb 39 68 dd ed -> TS_SECURITY_HEADER1::dataSignature

60 54 ad 97 a5 a5 ec 44 e6 63 45 20 bd c9 66 4e
12 de 01 d3 3c 39 09 0c 99 f8 -> Encrypted TS_CONTROL_PDU

Decrypted TS_CONTROL_PDU:
00000000 1a 00 17 00 ea 03 ea 03 01 00 12 02 1a 00 14 00 ................
00000010 00 00 02 00 ef 03 ea 03 00 00                   ..........

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
12 -> TS_SHAREDATAHEADER::pad1
02 -> TS_SHAREDATAHEADER::streamId = STREAM_MED (2)
1a 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x001a = 26 bytes
14 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_CONTROL (20)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

02 00 -> TS_CONTROL_PDU::action = CTRLACTION_GRANTED_CONTROL (2)
ef 03 -> TS_CONTROL_PDU::grantId = 0x03ef = 1007
ea 03 00 00 -> TS_CONTROL_PDU::controlId = 0x03ea = 1002
```

4.1.27 Server Font Map PDU

The following is an annotated dump of the Server Font Map PDU (section 2.2.1.22).

```
00000000 03 00 00 34 02 f0 80 68 00 01 03 eb 70 26 08 08 ...4...h....p&..
00000010 02 03 41 e9 b7 a2 62 9e bb d3 a0 be 09 9e d4 de ..A...b.........
00000020 8c 6d b6 79 64 4c bf 9d 21 46 32 7f 3b e4 dc 7f .m.ydL..!F2.;...
00000030 08 39 23 c1                                     .9#.

03 00 00 34 -> TPKT Header (length = 52 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 26 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
```

```
userData length = 0x26 = 38 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
41 e9 b7 a2 62 9e bb d3 -> TS_SECURITY_HEADER1::dataSignature

a0 be 09 9e d4 de 8c 6d b6 79 64 4c bf 9d 21 46
32 7f 3b e4 dc 7f 08 39 23 c1 -> Encrypted TS_FONT_MAP_PDU

Decrypted TS_FONT_MAP_PDU:
00000000 1a 00 17 00 ea 03 ea 03 01 00 45 02 1a 00 28 00 ..........E...(.
00000010 00 00 00 00 00 00 03 00 04 00                   ..........

1a 00 -> TS_SHARECONTROLHEADER::totalLength = 0x001a = 26 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 01 00 -> TS_SHAREDATAHEADER::shareID = 0x000103ea
45 -> TS_SHAREDATAHEADER::pad1
02 -> TS_SHAREDATAHEADER::streamId = STREAM_MED (2)
1a 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x001a = 26 bytes
28 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_FONTMAP (40)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

00 00 -> TS_FONT_MAP_PDU_DATA::numberEntries = 0
00 00 -> TS_FONT_MAP_PDU_DATA::totalNumEntries = 0

03 00 -> TS_FONT_MAP_PDU_DATA::mapFlags = 0x0003
0x0003
= 0x0002 | 0x0001
= FONTMAP_LAST | FONTMAP_FIRST

04 00 -> TS_FONT_MAP_PDU_DATA::entrySize = 4 bytes
```

4.2 Annotated User-Initiated (on Client) Disconnection Sequence

4.2.1 MCS Disconnect Provider Ultimatum PDU

The following is an annotated dump of the MCS Disconnect Provider Ultimatum PDU (section 2.2.2.1).

```
00000000 03 00 00 09 02 f0 80 21 80                        .......!.

03 00 00 09 -> TPKT Header (length = 9 bytes)
02 f0 80 -> X.224 Data TPDU

PER encoded (basic aligned variant) PDU contents:
21 80

0x21:
0 - --\
0 -    |
1 -    | CHOICE: From DomainMCSPDU select disconnectProviderUltimatum
(8) of type DisconnectProviderUltimatum
0 -    |
0 -    |
0 - --/
0 - --\
1 -    |
       | DisconnectProviderUltimatum::reason = rn-user-requested (3)
0x80: |
1 - --/
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
0 - padding
```

4.2.2 Client Shutdown Request PDU

The following is an annotated dump of the Client Shutdown Request PDU (section 2.2.2.2).

```
00000000 03 00 00 2c 02 f0 80 64 00 06 03 eb 70 1e 08 08 ...,...d....p...
00000010 70 52 ca 3d ba 05 20 60 e6 57 43 2c f1 41 f0 3b pR.=.. `.WC,.A.;
00000020 0c a0 33 ff 04 55 d4 e6 9b 3c 28 f6             ..3..U...<(.

03 00 00 2c -> TPKT Header (length = 44 bytes)
02 f0 80 -> X.224 Data TPDU

64 00 06 03 eb 70 1e -> PER encoded (basic aligned variant)
SendDataRequest
initiator = 1007 (0x03ef)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x1e = 30 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
70 52 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
ca 3d ba 05 20 60 e6 57 -> TS_SECURITY_HEADER1::dataSignature

43 2c f1 41 f0 3b 0c a0 33 ff 04 55 d4 e6 9b 3c
28 f6 -> Encrypted TS_SHUTDOWN_REQ_PDU

Decrypted TS_SHUTDOWN_REQ_PDU:
12 00 17 00 ef 03 ea 03 02 00 00 01 04 00 24 00
00 00

12 00 -> TS_SHARECONTROLHEADER::totalLength = 0x0012 = 18 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ef 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ef = 1007
ea 03 02 00 -> TS_SHAREDATAHEADER::shareID = 0x000203ea
00 -> TS_SHAREDATAHEADER::pad1
```

```
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
04 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0004 = 4 bytes
24 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SHUTDOWN_REQUEST (36)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0
```

4.2.3 Server Shutdown Request Denied PDU

The following is an annotated dump of the Server Shutdown Request Denied PDU (section 2.2.2.3).

```
00000000 03 00 00 24 02 f0 80 68 00 01 03 eb 70 1e 08 08 ...$...h....p...
00000010 10 00 31 19 b0 6c e3 cf 5e 0a df b6 5f 69 ce 41 ..1..l..^..._i.A
00000020 e3 23 f1 f6 50 4a 59 2e af e8 80 fb             .#..PJY.....

03 00 00 24 -> TPKT Header (length = 36 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 1e -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x1e = 30 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0008 = SEC_ENCRYPT
10 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
31 19 b0 6c e3 cf 5e 0a -> TS_SECURITY_HEADER1::dataSignature

df b6 5f 69 ce 41 e3 23 f1 f6 50 4a 59 2e af e8
80 fb -> Encrypted TS_SHUTDOWN_DENIED_PDU

Decrypted TS_SHUTDOWN_DENIED_PDU:
12 00 17 00 ea 03 ea 03 02 00 a6 02 12 00 25 00
00 00

12 00 -> TS_SHARECONTROLHEADER::totalLength = 0x0012 = 18 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 02 00 -> TS_SHAREDATAHEADER::shareID = 0x000203ea
a6  -> TS_SHAREDATAHEADER::pad1
02 -> TS_SHAREDATAHEADER::streamId = STREAM_MED (2)
12 00 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0012 = 18 bytes
25 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SHUTDOWN_DENIED (37)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0
```

4.3 Annotated Save Session Info PDU

4.3.1 Logon Info Version 2

The following is an annotated dump of Save Session Info PDU containing a Logon Info Version 2 structure, as specified in Section 2.2.10.1.1.2.

```
00000000 03 00 02 8b 02 f0 80 68 00 01 03 eb 70 82 7c 08 .......h....p.|.
00000010 08 00 00 6e 4b c4 ce 9e 4a 69 c4 0a f9 41 2e 6b ...nK...Ji...A.k
00000020 28 f5 95 7e ca c3 87 37 43 4c da 68 84 12 11 a1 (..~...7CL.h....
00000030 b8 5c 28 b2 78 15 30 98 c2 20 00 36 ef e6 6c 91 .\(.x.0.. .6..l.
00000040 60 d2 c7 51 f7 de 49 c3 0c 3e 5b 51 89 7f a3 b3 `..Q..I..>[Q....
00000050 d6 58 30 50 7b 1b ed 47 b6 8a fe 4f e2 e3 7b 65 .X0P{..G...O..{e
00000060 08 52 ed bf 52 16 8c 8b 42 4e 31 a0 8c 8b 59 f9 .R..R...BN1...Y.
00000070 84 66 58 b4 f8 a0 b6 49 15 01 b4 00 56 bd fe 7e .fX....I....V..~
00000080 dd ea 4a e1 9a 5a 41 dc e0 9b 1d d6 ca 09 54 94 ..J..ZA.......T.
00000090 93 48 04 40 f3 6b 17 9b 81 a2 3d 66 2e c2 00 70 .H.@.k....=f...p
000000a0 8f c5 5e 12 a5 54 98 77 4b 74 22 07 a8 09 5b 4f ..^..T.wKt"...[O
000000b0 d6 04 50 6f 90 88 1f 6d 66 a6 19 31 59 f3 68 74 ..Po...mf..1Y.ht
000000c0 16 25 51 b1 25 97 7b 3b e2 c9 ae 99 0d 8b 61 77 .%Q.%.{;......aw
000000d0 3a c7 1c 2e 20 73 93 c3 c6 2b c2 2a d6 0c b6 9c :... s...+.*....
000000e0 72 b0 2d f1 4b 3d 9c 6c e0 22 2d d3 83 b2 a3 b9 r.-.K=.l."-.....
000000f0 6e 4f ee 0c f4 98 d7 8c 19 65 1a c6 be c4 9b d9 nO.......e......
00000100 b4 3f 30 0d df bf 31 9e 33 50 e2 20 a3 9b 1d e2 .?0...1.3P. ....
00000110 46 3c b0 dc 07 29 d8 0b ed c3 68 0a 2c d9 3f ff F<...)....h.,.?.
00000120 3b f2 96 be b6 cf cf 8f 36 d2 86 71 be f7 01 31 ;.......6..q...1
00000130 5c 61 e7 83 2e 0e 7b 3c 76 18 69 52 39 6e 94 6d \a....{<v.iR9n.m
00000140 e6 63 00 7f 2e 9f f3 bd 86 43 36 25 d5 1c 77 ed .c.......C6%..w.
00000150 45 c1 7f f8 41 23 1f 25 f8 0a f2 6d 6d ac 98 d5 E...A#.%...mm...
00000160 9e d8 3b e4 63 35 67 54 4e c6 8d 50 30 a4 ee af ..;.c5gTN..P0...
00000170 84 a4 63 80 9e 62 f3 f2 94 8e 2f a3 f9 71 06 99 ..c..b..../..q..
00000180 3f 25 c8 6d 84 57 1a 5c 51 ef 88 9e e6 60 87 13 ?%.m.W.\Q....`..
00000190 d9 dd 5c 16 d1 0a bc 99 ec c9 d0 fe ad 3b f7 a4 ..\..........;..
000001a0 28 7e 41 e5 a1 85 fd ed 92 52 13 7e 1f fa 0d 3f (~A......R.~...?
000001b0 05 13 86 05 b2 1c fb 5f 76 a5 4c 47 da 4b 2b 1a ......._v.LG.K+.
000001c0 88 7f 5d ae c9 c5 03 08 79 6a 96 96 9f 7a 11 be ..].....yj...z..
000001d0 5a 66 c5 21 f4 a4 bc a0 0f 04 b7 9c 1b 71 9e c4 Zf.!.........q..
000001e0 d7 b3 60 52 33 a1 c6 76 de cf 05 f1 71 dd 4a aa ..`R3..v....q.J.
000001f0 3d d6 db 2e a7 f9 45 95 f6 06 d5 a6 3a 49 d7 73 =.....E.....:I.s
00000200 c5 af 42 c1 f5 6a 86 2b f1 ad 04 4e 1c 7c 00 35 ..B..j.+...N.|.5
00000210 77 12 c1 7e 6a bd 07 e8 61 fa 78 70 d6 d6 10 f1 w..~j...a.xp....
00000220 35 53 d8 47 03 a8 7a 49 57 12 5d 96 3a 6d 1c 86 5S.G..zIW.].:m..
00000230 f6 72 28 c8 5c 87 72 49 3c 0f 9c 07 48 ef 12 5e .r(.\.rI<...H..^
00000240 14 77 38 01 d0 bf 5e 90 e1 9a 89 f2 fa c6 06 02 .w8...^.........
00000250 4d 90 fa fd d7 12 bd e6 7e d6 08 15 82 98 b1 c1 M.......~.......
00000260 84 1b d2 9e 29 41 c0 19 96 16 82 4f 16 ee 5e 86 ....)A.....O..^.
00000270 9a 1c 2d 1f 85 c3 46 65 ed 31 d4 a9 47 e5 e4 64 ..-...Fe.1..G..d
00000280 d9 40 0f 78 4e 47 91 ec d7 39 c6                .@.xNG...9.

03 00 02 8b -> TPKT Header (length = 651 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 82 7c -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x27c = 636 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

00 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)

6e 4b c4 ce 9e 4a 69 c4 -> TS_SECURITY_HEADER1::dataSignature

0a f9 41 2e 6b 28 f5 95 7e ca c3 87 37 43 4c da
```

```
68 84 12 11 a1 b8 5c 28 b2 78 15 30 98 c2 20 00
36 ef e6 6c 91 60 d2 c7 51 f7 de 49 c3 0c 3e 5b
51 89 7f a3 b3 d6 58 30 50 7b 1b ed 47 b6 8a fe
4f e2 e3 7b 65 08 52 ed bf 52 16 8c 8b 42 4e 31
a0 8c 8b 59 f9 84 66 58 b4 f8 a0 b6 49 15 01 b4
00 56 bd fe 7e dd ea 4a e1 9a 5a 41 dc e0 9b 1d
d6 ca 09 54 94 93 48 04 40 f3 6b 17 9b 81 a2 3d
66 2e c2 00 70 8f c5 5e 12 a5 54 98 77 4b 74 22
07 a8 09 5b 4f d6 04 50 6f 90 88 1f 6d 66 a6 19
31 59 f3 68 74 16 25 51 b1 25 97 7b 3b e2 c9 ae
99 0d 8b 61 77 3a c7 1c 2e 20 73 93 c3 c6 2b c2
2a d6 0c b6 9c 72 b0 2d f1 4b 3d 9c 6c e0 22 2d
d3 83 b2 a3 b9 6e 4f ee 0c f4 98 d7 8c 19 65 1a
c6 be c4 9b d9 b4 3f 30 0d df bf 31 9e 33 50 e2
20 a3 9b 1d e2 46 3c b0 dc 07 29 d8 0b ed c3 68
0a 2c d9 3f ff 3b f2 96 be b6 cf cf 8f 36 d2 86
71 be f7 01 31 5c 61 e7 83 2e 0e 7b 3c 76 18 69
52 39 6e 94 6d e6 63 00 7f 2e 9f f3 bd 86 43 36
25 d5 1c 77 ed 45 c1 7f f8 41 23 1f 25 f8 0a f2
6d 6d ac 98 d5 9e d8 3b e4 63 35 67 54 4e c6 8d
50 30 a4 ee af 84 a4 63 80 9e 62 f3 f2 94 8e 2f
a3 f9 71 06 99 3f 25 c8 6d 84 57 1a 5c 51 ef 88
9e e6 60 87 13 d9 dd 5c 16 d1 0a bc 99 ec c9 d0
fe ad 3b f7 a4 28 7e 41 e5 a1 85 fd ed 92 52 13
7e 1f fa 0d 3f 05 13 86 05 b2 1c fb 5f 76 a5 4c
47 da 4b 2b 1a 88 7f 5d ae c9 c5 03 08 79 6a 96
96 9f 7a 11 be 5a 66 c5 21 f4 a4 bc a0 0f 04 b7
9c 1b 71 9e c4 d7 b3 60 52 33 a1 c6 76 de cf 05
f1 71 dd 4a aa 3d d6 db 2e a7 f9 45 95 f6 06 d5
a6 3a 49 d7 73 c5 af 42 c1 f5 6a 86 2b f1 ad 04
4e 1c 7c 00 35 77 12 c1 7e 6a bd 07 e8 61 fa 78
70 d6 d6 10 f1 35 53 d8 47 03 a8 7a 49 57 12 5d
96 3a 6d 1c 86 f6 72 28 c8 5c 87 72 49 3c 0f 9c
07 48 ef 12 5e 14 77 38 01 d0 bf 5e 90 e1 9a 89
f2 fa c6 06 02 4d 90 fa fd d7 12 bd e6 7e d6 08
15 82 98 b1 c1 84 1b d2 9e 29 41 c0 19 96 16 82
4f 16 ee 5e 86 9a 1c 2d 1f 85 c3 46 65 ed 31 d4
a9 47 e5 e4 64 d9 40 0f 78 4e 47 91 ec d7 39 c6 -> Encrypted
TS_SAVE_SESSION_INFO_PDU_DATA

Decrypted TS_SAVE_SESSION_INFO_PDU_DATA:
00000000 70 02 17 00 ea 03 ea 03 02 00 00 01 70 02 26 00 p...........p.&.
00000010 00 00 01 00 00 00 01 00 12 00 00 00 02 00 00 00 ................
00000020 0c 00 00 00 0e 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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 00 00 00 00 ................
00000090 00 00 00 00 00 00 00 00 00 00 00 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 00 00 00 00 00 00 ................
000000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000250 00 00 00 00 00 00 4e 00 54 00 44 00 45 00 56 00 ......N.T.D.E.V.
00000260 00 00 65 00 6c 00 74 00 6f 00 6e 00 73 00 00 00 ..e.l.t.o.n.s...
```

```
70 02 -> TS_SHARECONTROLHEADER::totalLength = 0x0270 = 624 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 02 00 -> TS_SHAREDATAHEADER::shareID = 0x000203ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
70 02 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0270 = 624 bytes
26 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SAVE_SESSION_INFO (38)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

01 00 00 00 -> TS_SAVE_SESSION_INFO_PDU_DATA::infoType =
INFOTYPE_LOGON_LONG (1)

01 00 -> TS_LOGON_INFO_VERSION_2::Version
12 00 00 00 -> TS_LOGON_INFO_VERSION_2::Size
02 00 00 00 -> TS_LOGON_INFO_VERSION_2::SessionId

0c 00 00 00 -> TS_LOGON_INFO_VERSION_2::cbDomain
0e 00 00 00 -> TS_LOGON_INFO_VERSION_2::cbUserName

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_LOGON_INFO_VERSION_2::Pad (558 bytes)

4e 00 54 00 44 00 45 00 56 00 00 00 ->
TS_LOGON_INFO_VERSION_2::Domain = ""NTDEV
65 00 6c 00 74 00 6f 00 6e 00 73 00 00 00 ->
TS_LOGON_INFO_VERSION_2::UserName = "username"
```

4.3.2 Plain Notify

The following is an annotated dump of Save Session Info PDU containing a Plain Notify structure, as specified in Section 2.2.10.1.1.3.

```
00000000 03 00 02 71 02 f0 80 68 00 01 03 eb 70 82 62 08 ...q...h....p.b.
00000010 08 02 03 90 94 9a cc a2 38 22 3b 03 6e a4 a2 e3 ........8";.n...
00000020 1c 4d 55 aa 56 d3 ca f8 e6 52 99 1e b5 f1 a0 42 .MU.V....R.....B
00000030 4e 89 64 83 54 1f da 89 a7 f5 53 8b 61 bb 73 b5 N.d.T.....S.a.s.
00000040 58 d4 6b bc 28 c2 84 c3 90 b4 45 b5 97 d5 d2 05 X.k.(.....E.....
00000050 bc 66 a4 d4 73 31 7e 0e 4d 42 12 0a 95 88 18 ff .f..s1~.MB......
00000060 f6 87 07 71 38 5b 3e 48 e6 d4 d0 2f c2 80 4c 7f ...q8[>H.../..L.
00000070 7d 88 78 5f ec 06 cf 8d cb 91 d6 d3 7c 56 45 59 }.x_........|VEY
00000080 7c 26 05 ed 14 92 a4 a5 a7 d8 98 1b f0 bf be b0 |&..............
00000090 bf e3 35 e8 38 8a ad 12 ec e1 72 9c 89 0a 1e a5 ..5.8.....r.....
000000a0 dc 19 48 5e 2a 7f 9e d0 11 92 70 cc 01 45 50 d5 ..H^*.....p..EP.
000000b0 1e c7 f9 ff 74 c1 74 45 04 4e 4f 5d 49 ce 41 b3 ....t.tE.NO]I.A.
000000c0 ed 7f 5c 0e bb 37 50 d0 f7 79 e9 d7 c0 55 4a 1c ..\..7P..y...UJ.
000000d0 54 29 84 62 3f c9 68 04 5f b3 51 41 89 2b 36 a6 T).b?.h._.QA.+6.
000000e0 65 0a 4e da 92 61 38 a5 73 16 a5 b4 cd 87 db 84 e.N..a8.s.......
000000f0 10 3e b9 1f ad 3e df 50 37 5b 8e ac cb e9 e5 51 .>...>.P7[.....Q
00000100 90 bf e1 e5 0f 16 f2 70 b9 dc 89 2a 46 53 c1 fa .......p...*FS..
00000110 e2 ef 0a bb ce 16 a1 2a 2d 24 1e 21 fe b9 b6 54 .......*-$.!...T
00000120 2a 6e ff e5 b7 d3 84 52 19 dd 41 eb eb 4b 81 ab *n.....R..A..K..
00000130 20 11 8c 18 19 45 e9 23 00 58 a5 71 94 6c c0 58  ....E.#.X.q.l.X
00000140 70 9b 1d 75 f6 e4 f7 18 17 f9 8c 1d e9 c1 9b 76 p..u...........v
00000150 21 a3 6e f6 3e 4b 82 54 f2 16 96 21 0e 1c 54 e9 !.n.>K.T...!..T.
00000160 d1 65 18 0f e5 f9 45 bf d7 f9 24 a9 7e 3e 6a 73 .e....E...$.~>js
00000170 23 fc 3c 0a 04 52 c4 ee fa 13 64 21 a1 47 2d 4a #.<..R....d!.G-J
00000180 4f 00 c0 80 8b 9c a6 ec e9 94 57 a4 3d 88 77 e5 O.........W.=.w.
00000190 b6 71 e6 a1 15 a4 c6 02 64 a1 af 34 b9 73 87 e1 .q......d..4.s..
000001a0 22 1b 33 a5 bf bb 7e 96 bc 31 92 f8 4a bc ab f8 ".3...~..1..J...
000001b0 3f 5b 85 1b 23 75 46 45 b7 31 08 45 ca de 1f df ?[..#uFE.1.E....
000001c0 49 3e 37 f1 2e af 16 d2 5c 3e 2e 30 68 36 d1 ae I>7.....\>.0h6..
000001d0 9e 0d bf ff 53 ce 96 f6 6f 31 60 f1 40 e0 6f 0c ....S...o1`.@.o.
000001e0 a1 b3 b3 6b 04 99 a1 f6 b9 cf 69 21 e4 a2 bc 07 ...k......i!....
000001f0 81 c4 36 dc 9e 99 9d 50 da 62 55 71 f0 5d 3d fd ..6....P.bUq.]=.
00000200 08 73 54 b6 cb 48 dd 5d 54 1a 08 09 ae 9f 98 b0 .sT..H.]T.......
00000210 3b e3 2a a8 e8 61 1f 4f e5 11 d4 4f 8e e0 96 8d ;.*..a.O...O....
00000220 c8 ed d1 9e f2 27 1f c6 79 dc a2 df 52 01 21 be .....'..y...R.!.
00000230 13 7f c6 55 bb 08 b1 d3 2d de e3 7b 8b 11 95 53 ...U....-..{...S
00000240 af 4b bf 80 e9 5f 54 d4 96 f1 da 35 ee d4 50 e8 .K..._T....5..P.
00000250 28 58 aa 59 86 db f3 e5 44 a3 8b 3c 40 fd f5 b5 (X.Y....D..<@...
00000260 9f 1d b8 1c 30 43 52 9f 4b 34 4b c7 59 6b b6 06 ....0CR.K4K.Yk..
00000270 e7                                              .

03 00 02 71 -> TPKT Header (length = 625 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 82 62 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x262 = 610 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

02 03 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)

90 94 9a cc a2 38 22 3b -> TS_SECURITY_HEADER1::dataSignature

03 6e a4 a2 e3 1c 4d 55 aa 56 d3 ca f8 e6 52 99
1e b5 f1 a0 42 4e 89 64 83 54 1f da 89 a7 f5 53
8b 61 bb 73 b5 58 d4 6b bc 28 c2 84 c3 90 b4 45
b5 97 d5 d2 05 bc 66 a4 d4 73 31 7e 0e 4d 42 12
0a 95 88 18 ff f6 87 07 71 38 5b 3e 48 e6 d4 d0
2f c2 80 4c 7f 7d 88 78 5f ec 06 cf 8d cb 91 d6
d3 7c 56 45 59 7c 26 05 ed 14 92 a4 a5 a7 d8 98
1b f0 bf be b0 bf e3 35 e8 38 8a ad 12 ec e1 72
9c 89 0a 1e a5 dc 19 48 5e 2a 7f 9e d0 11 92 70
cc 01 45 50 d5 1e c7 f9 ff 74 c1 74 45 04 4e 4f
5d 49 ce 41 b3 ed 7f 5c 0e bb 37 50 d0 f7 79 e9
d7 c0 55 4a 1c 54 29 84 62 3f c9 68 04 5f b3 51
41 89 2b 36 a6 65 0a 4e da 92 61 38 a5 73 16 a5
b4 cd 87 db 84 10 3e b9 1f ad 3e df 50 37 5b 8e
ac cb e9 e5 51 90 bf e1 e5 0f 16 f2 70 b9 dc 89
2a 46 53 c1 fa e2 ef 0a bb ce 16 a1 2a 2d 24 1e
21 fe b9 b6 54 2a 6e ff e5 b7 d3 84 52 19 dd 41
eb eb 4b 81 ab 20 11 8c 18 19 45 e9 23 00 58 a5
71 94 6c c0 58 70 9b 1d 75 f6 e4 f7 18 17 f9 8c
1d e9 c1 9b 76 21 a3 6e f6 3e 4b 82 54 f2 16 96
21 0e 1c 54 e9 d1 65 18 0f e5 f9 45 bf d7 f9 24
a9 7e 3e 6a 73 23 fc 3c 0a 04 52 c4 ee fa 13 64
21 a1 47 2d 4a 4f 00 c0 80 8b 9c a6 ec e9 94 57
a4 3d 88 77 e5 b6 71 e6 a1 15 a4 c6 02 64 a1 af
34 b9 73 87 e1 22 1b 33 a5 bf bb 7e 96 bc 31 92
f8 4a bc ab f8 3f 5b 85 1b 23 75 46 45 b7 31 08
45 ca de 1f df 49 3e 37 f1 2e af 16 d2 5c 3e 2e
30 68 36 d1 ae 9e 0d bf ff 53 ce 96 f6 6f 31 60
f1 40 e0 6f 0c a1 b3 b3 6b 04 99 a1 f6 b9 cf 69
21 e4 a2 bc 07 81 c4 36 dc 9e 99 9d 50 da 62 55
71 f0 5d 3d fd 08 73 54 b6 cb 48 dd 5d 54 1a 08
09 ae 9f 98 b0 3b e3 2a a8 e8 61 1f 4f e5 11 d4
4f 8e e0 96 8d c8 ed d1 9e f2 27 1f c6 79 dc a2
df 52 01 21 be 13 7f c6 55 bb 08 b1 d3 2d de e3
7b 8b 11 95 53 af 4b bf 80 e9 5f 54 d4 96 f1 da
35 ee d4 50 e8 28 58 aa 59 86 db f3 e5 44 a3 8b
3c 40 fd f5 b5 9f 1d b8 1c 30 43 52 9f 4b 34 4b
c7 59 6b b6 06 e7 -> Encrypted TS_SAVE_SESSION_INFO_PDU_DATA

Decrypted TS_SAVE_SESSION_INFO_PDU_DATA:
00000 56 02 17 00 ea 03 ea 03 02 00 00 01 56 02 26 00 V...........V.&.
```

```
00010 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00250 00 00 00 00 00 00                               ......
```

```
56 02 -> TS_SHARECONTROLHEADER::totalLength = 0x0256 = 598 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 02 00 -> TS_SHAREDATAHEADER::shareID = 0x000203ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
56 02 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0256 = 598 bytes
26 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SAVE_SESSION_INFO (38)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

02 00 00 00 -> TS_SAVE_SESSION_INFO_PDU_DATA::infoType =
INFOTYPE_LOGON_PLAINNOTIFY (2)

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ->
TS_PLAIN_NOTIFY::Pad (576 bytes)
```

4.3.3 Logon Info Extended

The following is an annotated dump of Save Session Info PDU containing a Logon Info Extended structure, as specified in Section 2.2.10.1.1.4.

```
00000000 03 00 02 91 02 f0 80 68 00 01 03 eb 70 82 82 08 .......h....p...
00000010 08 00 00 a6 70 37 7e 91 62 c5 1d c4 a0 a9 67 53 ....p7~.b.....gS
00000020 c0 fa c3 ee 78 9d 89 70 8e 6b e4 0e d9 2f 44 39 ....x..p.k.../D9
00000030 97 20 3d 78 77 9e 53 44 4d 91 f3 71 3e 78 60 7b . =xw.SDM..q>x`{
00000040 6b c6 05 3c 4a f6 2e 92 00 3c 63 81 ce e7 da 37 k..<J....<c....7
00000050 33 07 70 af a3 8c f8 3a a1 cd dd 02 60 8b 85 35 3.p....:....`..5
00000060 57 7b 6e dd 69 84 22 68 11 46 74 e6 ae 17 18 8d W{n.i."h.Ft.....
00000070 df 94 52 6b 82 1e b9 77 73 07 1e 0c 76 d4 83 87 ..Rk...ws...v...
00000080 38 34 4c f5 3e cf 4f 75 d2 53 bf db 3d fb e4 77 84L.>.Ou.S..=..w
00000090 92 c9 fc 43 dc 06 96 c0 ad c7 dc 48 11 83 2a 40 ...C.......H..*@
000000a0 d4 58 3c cd 7e 6e bb d8 a4 f1 a1 6d c5 6e 98 90 .X<.~n.....m.n..
000000b0 e6 0f 73 02 6a f2 d3 05 af ee 01 e2 cb 5d 8c ae ..s.j........]..
000000c0 a4 66 4b c6 36 c4 5e 61 a2 fd c3 cd 2f 8c fb a9 .fK.6.^a..../...
000000d0 34 bb 55 61 92 a8 bf b4 2a aa ff 3a 35 3e 62 4b 4.Ua....*..:5>bK
000000e0 14 bc ae 11 36 c8 f4 14 c2 ce 86 0f 6c d8 36 57 ....6.......l.6W
000000f0 d6 d4 4e c4 f4 62 54 86 46 e6 c3 a7 fe 6a b5 53 ..N..bT.F....j.S
00000100 49 8a a6 72 13 fb e5 60 2f 3c 21 4b 76 54 99 e8 I..r...`/<!KvT..
00000110 c1 83 6c 89 e4 2d 57 ad 15 61 f4 06 bf 87 c8 a6 ..l..-W..a......
00000120 69 5a f4 ec 6d de c6 af df f8 82 be 42 d0 21 85 iZ..m.......B.!.
00000130 59 e3 80 9f a6 18 5c 83 3b b5 29 9b c2 f6 ee 13 Y.....\.;.).....
00000140 2e 53 5c ea ee 2f e4 52 93 58 90 e1 2b fb c1 9d .S\../.R.X..+...
00000150 2d 64 95 61 8a 22 36 00 45 ea 56 b5 39 e6 de fe -d.a."6.E.V.9...
00000160 82 dc 67 ec 1d da 2d a3 17 27 22 c2 39 44 2f 04 ..g...-..'".9D/.
00000170 8d 8b ff 84 27 f0 9c 18 2a d2 69 a0 af fd 6a e0 ....'...*.i...j.
00000180 3d ab ce f7 4b 6b 5d 8e bf 49 24 b4 71 ec 70 5e =...Kk]..I$.q.p^
00000190 14 42 cf 0c 8b 45 b6 7d 77 b1 23 0c 87 3b fa f0 .B...E.}w.#..;..
000001a0 44 13 31 b4 16 84 db 03 c7 04 dd 23 b7 5c 95 c7 D.1........#.\..
000001b0 29 50 5d d6 dd 21 39 85 18 1b dd fa 1c a2 0a 66 )P]..!9........f
```

```
000001c0 a6 75 e0 e5 e4 f0 0e 20 9d 39 9f 07 eb 2c 7f fc .u..... .9...,..
000001d0 3b f2 88 e0 88 dd 9f 3c 1d b2 36 8b 90 81 b1 63 ;......<..6....c
000001e0 3f 31 40 2b 91 a7 1b f3 59 bf 90 53 68 c2 5a 99 ?1@+....Y..Sh.Z.
000001f0 4d 2e 2d 59 b7 bc f9 ba 05 45 18 2c 3c 16 ae d9 M.-Y.....E.,<...
00000200 0d f1 35 fd 0d 12 51 08 50 18 d2 38 07 52 4c cb ..5...Q.P..8.RL.
00000210 8c 16 b9 5a 57 2a 8e 7c ee d7 82 56 27 a8 f0 1d ...ZW*.|...V'...
00000220 9b e8 be 06 a3 ac c3 b8 61 d6 e3 70 05 5a 14 68 ........a..p.Z.h
00000230 19 4f 78 a5 5a 0d 0a 13 e5 e4 78 04 46 00 cb ba .Ox.Z.....x.F...
00000240 53 b2 10 a4 6c d9 7b 07 34 44 52 fb e8 65 49 57 S...l.{.4DR..eIW
00000250 f9 96 6e 0f 53 30 b7 31 93 15 a1 cb 60 ba 6a c4 ..n.S0.1....`.j.
00000260 dc 29 ac 11 8c 37 91 eb b3 97 b8 51 88 5d 11 f9 .)...7.....Q.]..
00000270 79 8b 3e 38 8e 88 3d 54 0d fa 83 58 2f ef bc 80 y.>8..=T...X/...
00000280 2b 78 8c b8 91 c2 a2 21 36 85 00 ae ef 2e c6 28 +x.....!6......(
00000290 3d                                              =

03 00 02 91 -> TPKT Header (length = 657 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 82 82 -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x282 = 642 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

00 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)

a6 70 37 7e 91 62 c5 1d -> TS_SECURITY_HEADER1::dataSignature

c4 a0 a9 67 53 c0 fa c3 ee 78 9d 89 70 8e 6b e4
0e d9 2f 44 39 97 20 3d 78 77 9e 53 44 4d 91 f3
71 3e 78 60 7b 6b c6 05 3c 4a f6 2e 92 00 3c 63
81 ce e7 da 37 33 07 70 af a3 8c f8 3a a1 cd dd
02 60 8b 85 35 57 7b 6e dd 69 84 22 68 11 46 74
e6 ae 17 18 8d df 94 52 6b 82 1e b9 77 73 07 1e
0c 76 d4 83 87 38 34 4c f5 3e cf 4f 75 d2 53 bf
db 3d fb e4 77 92 c9 fc 43 dc 06 96 c0 ad c7 dc
48 11 83 2a 40 d4 58 3c cd 7e 6e bb d8 a4 f1 a1
6d c5 6e 98 90 e6 0f 73 02 6a f2 d3 05 af ee 01
e2 cb 5d 8c ae a4 66 4b c6 36 c4 5e 61 a2 fd c3
cd 2f 8c fb a9 34 bb 55 61 92 a8 bf b4 2a aa ff
3a 35 3e 62 4b 14 bc ae 11 36 c8 f4 14 c2 ce 86
0f 6c d8 36 57 d6 d4 4e c4 f4 62 54 86 46 e6 c3
a7 fe 6a b5 53 49 8a a6 72 13 fb e5 60 2f 3c 21
4b 76 54 99 e8 c1 83 6c 89 e4 2d 57 ad 15 61 f4
06 bf 87 c8 a6 69 5a f4 ec 6d de c6 af df f8 82
be 42 d0 21 85 59 e3 80 9f a6 18 5c 83 3b b5 29
9b c2 f6 ee 13 2e 53 5c ea ee 2f e4 52 93 58 90
e1 2b fb c1 9d 2d 64 95 61 8a 22 36 00 45 ea 56
b5 39 e6 de fe 82 dc 67 ec 1d da 2d a3 17 27 22
c2 39 44 2f 04 8d 8b ff 84 27 f0 9c 18 2a d2 69
a0 af fd 6a e0 3d ab ce f7 4b 6b 5d 8e bf 49 24
b4 71 ec 70 5e 14 42 cf 0c 8b 45 b6 7d 77 b1 23
0c 87 3b fa f0 44 13 31 b4 16 84 db 03 c7 04 dd
23 b7 5c 95 c7 29 50 5d d6 dd 21 39 85 18 1b dd
fa 1c a2 0a 66 a6 75 e0 e5 e4 f0 0e 20 9d 39 9f
07 eb 2c 7f fc 3b f2 88 e0 88 dd 9f 3c 1d b2 36
8b 90 81 b1 63 3f 31 40 2b 91 a7 1b f3 59 bf 90
```

```
53 68 c2 5a 99 4d 2e 2d 59 b7 bc f9 ba 05 45 18
2c 3c 16 ae d9 0d f1 35 fd 0d 12 51 08 50 18 d2
38 07 52 4c cb 8c 16 b9 5a 57 2a 8e 7c ee d7 82
56 27 a8 f0 1d 9b e8 be 06 a3 ac c3 b8 61 d6 e3
70 05 5a 14 68 19 4f 78 a5 5a 0d 0a 13 e5 e4 78
04 46 00 cb ba 53 b2 10 a4 6c d9 7b 07 34 44 52
fb e8 65 49 57 f9 96 6e 0f 53 30 b7 31 93 15 a1
cb 60 ba 6a c4 dc 29 ac 11 8c 37 91 eb b3 97 b8
51 88 5d 11 f9 79 8b 3e 38 8e 88 3d 54 0d fa 83
58 2f ef bc 80 2b 78 8c b8 91 c2 a2 21 36 85 00
ae ef 2e c6 28 3d -> Encrypted TS_SAVE_SESSION_INFO_PDU_DATA

Decrypted TS_SAVE_SESSION_INFO_PDU_DATA:
00000 76 02 17 00 ea 03 ea 03 02 00 00 01 76 02 26 00 v...........v.&.
00010 00 00 03 00 00 00 26 00 01 00 00 00 1c 00 00 00 ......&.........
00020 1c 00 00 00 01 00 00 00 02 00 00 00 a8 02 e7 25 ...............%
00030 e2 4c 82 b7 52 a5 53 50 34 98 a1 a8 00 00 00 00 .L..R.SP4.......
00040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00270 00 00 00 00 00 00                               ......

76 02 -> TS_SHARECONTROLHEADER::totalLength = 0x0276 = 630 bytes
17 00 -> TS_SHARECONTROLHEADER::pduType = 0x0017
0x0017
= 0x0010 | 0x0007
= TS_PROTOCOL_VERSION | PDUTYPE_DATAPDU

ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea = 1002
ea 03 02 00 -> TS_SHAREDATAHEADER::shareID = 0x000203ea
00 -> TS_SHAREDATAHEADER::pad1
01 -> TS_SHAREDATAHEADER::streamId = STREAM_LOW (1)
76 02 -> TS_SHAREDATAHEADER::uncompressedLength = 0x0276 = 630 bytes
```

```
26 -> TS_SHAREDATAHEADER::pduType2 = PDUTYPE2_SAVE_SESSION_INFO (38)
00 -> TS_SHAREDATAHEADER::generalCompressedType = 0
00 00 -> TS_SHAREDATAHEADER::generalCompressedLength = 0

03 00 00 00 -> TS_SAVE_SESSION_INFO_PDU_DATA::infoType =
INFOTYPE_LOGON_EXTENDED_INFO (3)

26 00 -> TS_LOGON_INFO_EXTENDED::Length = 0x26 = 38
01 00 00 00 -> TS_LOGON_INFO_EXTENDED::FieldsPresent =
LOGON_EX_AUTORECONNECTCOOKIE (1)

1c 00 00 00 -> TS_LOGON_INFO_FIELD::cbFieldData = 28

1c 00 00 00 -> ARC_SC_PRIVATE_PACKET::cbLen = 28
01 00 00 00 -> ARC_SC_PRIVATE_PACKET::Version
02 00 00 00 -> ARC_SC_PRIVATE_PACKET::LogonId
a8 02 e7 25 e2 4c 82 b7 52 a5 53 50 34 98 a1 a8 ->
ARC_SC_PRIVATE_PACKET::ArcRandomBits

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 ->
TS_LOGON_INFO_EXTENDED::Pad (570 bytes)
```

4.4 Annotated Server-to-Client Virtual Channel PDU

The following is an annotated dump of the Virtual Channel PDU (section 2.2.6.1).

```
00000000 03 00 00 2e 02 f0 80 68 00 01 03 ed f0 20 08 08 .......h..... ..
```

```
00000010 01 00 47 bd eb cb 29 51 ae 0a f6 07 33 ce fc a5 ..G...)Q....3...
00000020 f7 09 de 67 4e a3 2a 2c 38 29                   ...gN.*,8)

03 00 00 2a -> TPKT Header (length = 42 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 ed f0 1c -> PER encoded (basic aligned variant)
SendDataIndication
initiator = 1007 (0x03ef)
channelId = 1005 (0x03ed) = "cliprdr"
dataPriority = low
segmentation = begin | end
userData length = 0x1c = 28 bytes

08 08 -> TS_SECURITY_HEADER::flags = 0x0808
0x0808
= 0x0800 | 0x0008
= SEC_SECURE_CHECKSUM | SEC_ENCRYPT

01 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does
not contain SEC_FLAGSHI_VALID (0x8000)
47 bd eb cb 29 51 ae 0a -> TS_SECURITY_HEADER::dataSignature

f6 07 33 ce fc a5 f7 09 de 67 4e a3 2a 2c 38 29 -> Encrypted static
virtual channel data

Decrypted static virtual channel data:
00000000 08 00 00 00 03 00 00 00 03 00 01 00 00 00 00 00 ................

08 00 00 00 -> CHANNEL_PDU_HEADER::length = 8 bytes

03 00 00 00 -> CHANNEL_PDU_HEADER::flags = 0x00000003
0x00000003
= 0x00000002 | 0x00000001
= CHANNEL_FLAG_FIRST | CHANNEL_FLAG_LAST

03 00 01 00 00 00 00 00 -> Channel data to be processed by the
"cliprdr" handler
```

4.5 Java Code to Encrypt and Decrypt a Sample Client Random

The following Java code illustrates how to encrypt and decrypt with RSA.

```
import java.math.BigInteger;

public class RdpRsaEncrypt
{
    //
    // Print out the contents of a byte array in hexadecimal.
    //
    private static void PrintBytes(
        byte[] bytes
        )
    {
        int cBytes = bytes.length;
        int iByte = 0;

        for (;;) {
            for (int i = 0; i < 8; i++) {
                String hex = Integer.toHexString(bytes[iByte++] & 0xff);
                if (hex.length() == 1) {
                    hex = "0" + hex;
                }

                System.out.print("0x" + hex + " ");
                if (iByte >= cBytes) {
                    System.out.println();
                    return;
                }
            }
            System.out.println();
        }
    }

    //
    // Reverse the order of the values in a byte array.
    //
    public static void ReverseByteArray(
        byte[] array
        )
    {
        int i, j;
        byte temp;

        for (i = 0, j = array.length - 1; i < j; i++, j--) {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    //
    // Use RSA to encrypt data.
    //
    public static byte[] RsaEncrypt(
        byte[] modulusBytes,
        byte[] exponentBytes,
        byte[] dataBytes
        )
    {
        //
        // Reverse the passed in byte arrays and then use these to
        // create the BigIntegers for the RSA computation.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(exponentBytes);
        ReverseByteArray(dataBytes);

        BigInteger modulus = new BigInteger(
            1,
            modulusBytes
            );
        BigInteger exponent = new BigInteger(
            1,
            exponentBytes
            );
        BigInteger data = new BigInteger(
            1,
            dataBytes
            );

        //
        // Perform RSA encryption:
        // ciphertext = plaintext^exponent % modulus.
        //
        BigInteger cipherText = data.modPow(
            exponent,
                modulus
                );

            //
            // Reverse the generated ciphertext.
            //
            byte[] cipherTextBytes = cipherText.toByteArray();
            ReverseByteArray(cipherTextBytes);

            //
            // Undo the reversal of the passed in byte arrays.
            //
            ReverseByteArray(modulusBytes);
            ReverseByteArray(exponentBytes);
            ReverseByteArray(dataBytes);

            return cipherTextBytes;
        }

        //
        // Use RSA to decrypt data.
        //
        public static byte[] RsaDecrypt(
            byte[] modulusBytes,
            byte[] privateExponentBytes,
            byte[] encryptedDataBytes
            )
        {
            //
            // Reverse the passed in byte arrays and then use these to
    create the
            // BigIntegers for the RSA computation.
            //
            ReverseByteArray(modulusBytes);
            ReverseByteArray(privateExponentBytes);
            ReverseByteArray(encryptedDataBytes);

            BigInteger modulus = new BigInteger(
                1,
                modulusBytes
                );
            BigInteger privateExponent = new BigInteger(
                1,
                privateExponentBytes
                );
            BigInteger encryptedData = new BigInteger(
                1,
                encryptedDataBytes
                );

            //
            // Perform RSA encryption:
            // plaintext = ciphertext^privateExponent % modulus.
            //
            BigInteger decryptedData = encryptedData.modPow(
                privateExponent,
                modulus
                );

            //
            // Reverse the generated plaintext.
            //
            byte[] decryptedDataBytes = decryptedData.toByteArray();
            ReverseByteArray(decryptedDataBytes);
        //
        // Undo the reversal of the passed in byte arrays.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(privateExponentBytes);
        ReverseByteArray(encryptedDataBytes);

        return decryptedDataBytes;
    }

    //
    // Main routine.
    //
    public static void main(
        String[] args
        )
    {
        //
        // Modulus bytes obtained straight from the wire in the
        // proprietary certificate (in little endian format).
        // This is for a 512-bit key set.
        //
        byte[] modulusBytes =
        {
            (byte) 0x37, (byte) 0xa8, (byte) 0x70, (byte) 0xfe,
            (byte) 0x9a, (byte) 0xb9, (byte) 0xa8, (byte) 0x54,
            (byte) 0xcb, (byte) 0x98, (byte) 0x79, (byte) 0x44,
            (byte) 0x7a, (byte) 0xb9, (byte) 0xeb, (byte) 0x38,
            (byte) 0x06, (byte) 0xea, (byte) 0x26, (byte) 0xa1,
            (byte) 0x47, (byte) 0xea, (byte) 0x19, (byte) 0x70,
            (byte) 0x5d, (byte) 0xf3, (byte) 0x52, (byte) 0x88,
            (byte) 0x70, (byte) 0x21, (byte) 0xb5, (byte) 0x9e,
            (byte) 0x50, (byte) 0xb4, (byte) 0xe1, (byte) 0xf5,
            (byte) 0x1a, (byte) 0xd8, (byte) 0x2d, (byte) 0x51,
            (byte) 0x4d, (byte) 0x1a, (byte) 0xad, (byte) 0x79,
            (byte) 0x7c, (byte) 0x89, (byte) 0x46, (byte) 0xb0,
            (byte) 0xcc, (byte) 0x66, (byte) 0x74, (byte) 0x02,
            (byte) 0xd8, (byte) 0x28, (byte) 0x5d, (byte) 0x9d,
            (byte) 0xd7, (byte) 0xca, (byte) 0xfc, (byte) 0x60,
            (byte) 0x0f, (byte) 0x38, (byte) 0xf9, (byte) 0xb3
        };

        //
        // Exponent bytes (in little endian order) obtained straight
        // from the wire (in the proprietary certificate).
        //
        byte[] exponentBytes =
        {
            (byte) 0x01, (byte) 0x00, (byte) 0x01, (byte) 0x00
        };

        //
        // Private exponent of the private key generated by the
        // server (in little endian format).
        //
        byte[] privateExponentBytes =
        {
            (byte) 0xc1, (byte) 0x07, (byte) 0xe7, (byte) 0xd4,
            (byte) 0xd3, (byte) 0x38, (byte) 0x8d, (byte) 0x36,
            (byte) 0xf5, (byte) 0x9e, (byte) 0x8b, (byte) 0x96,
            (byte) 0x0d, (byte) 0x55, (byte) 0x65, (byte) 0x08,
            (byte) 0x28, (byte) 0x25, (byte) 0xa3, (byte) 0x2e,
            (byte) 0xc7, (byte) 0x68, (byte) 0xd6, (byte) 0x44,
            (byte) 0x85, (byte) 0x2d, (byte) 0x32, (byte) 0xf6,
            (byte) 0x72, (byte) 0xa8, (byte) 0x9b, (byte) 0xba,
            (byte) 0x5e, (byte) 0x82, (byte) 0x82, (byte) 0xf0,
            (byte) 0x5c, (byte) 0x0c, (byte) 0xeb, (byte) 0x6b,
            (byte) 0x12, (byte) 0x6a, (byte) 0xa7, (byte) 0x45,
            (byte) 0x15, (byte) 0xce, (byte) 0x41, (byte) 0xe0,
            (byte) 0x03, (byte) 0xe5, (byte) 0xe6, (byte) 0x6d,
            (byte) 0xdf, (byte) 0xfd, (byte) 0x58, (byte) 0x61,
            (byte) 0x0b, (byte) 0x07, (byte) 0xa4, (byte) 0x7b,
            (byte) 0xb3, (byte) 0xf3, (byte) 0x71, (byte) 0x94
        };

        //
        // Sample 32-byte client random.
        //
        byte[] clientRandomBytes =
        {
            (byte) 0xff, (byte) 0xee, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0xff
        };

        System.out.println("Client random:");
        PrintBytes(clientRandomBytes);

        //
        // Perform encryption.
        //
        byte[] encryptedClientRandomBytes = RsaEncrypt(
            modulusBytes,
            exponentBytes,
            clientRandomBytes
            );

        System.out.println("Encrypted client random to send to
server:");
        PrintBytes(encryptedClientRandomBytes);

        //
        // Perform decryption.
        //
        byte[] decryptedClientRandomBytes = RsaDecrypt(
            modulusBytes,
            privateExponentBytes,
            encryptedClientRandomBytes
            );

        System.out.println("Decrypted client random:");
        PrintBytes(decryptedClientRandomBytes);
    }
};
```

4.6 Java Code to Sign a Sample Proprietary Certificate Hash

The following Java code illustrates how to sign a Proprietary Certificate Hash with RSA.

```
import java.math.BigInteger;
```

```
public class RdpRsaSign
{
    //
    // Print out the contents of a byte array in hexadecimal.
    //
    private static void PrintBytes(
        byte[] bytes
        )
    {
        int cBytes = bytes.length;
        int iByte = 0;

        for (;;) {
            for (int i = 0; i < 8; i++) {
                String hex = Integer.toHexString(bytes[iByte++] & 0xff);
                if (hex.length() == 1) {
                    hex = "0" + hex;
                }

                System.out.print("0x" + hex + " ");
                if (iByte >= cBytes) {
                    System.out.println();
                    return;
                }
            }
            System.out.println();
        }
    }

    //
    // Reverse the order of the values in a byte array.
    //
    public static void ReverseByteArray(
        byte[] array
        )
    {
        int i, j;
        byte temp;

        for (i = 0, j = array.length - 1; i < j; i++, j--) {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    //
    // Use RSA to encrypt data.
    //
    public static byte[] RsaEncrypt(
        byte[] modulusBytes,
        byte[] exponentBytes,
        byte[] dataBytes
        )
    {
        //
        // Reverse the passed in byte arrays and then use these to
        // create the BigIntegers for the RSA computation.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(exponentBytes);
        ReverseByteArray(dataBytes);

        BigInteger modulus = new BigInteger(
            1,
            modulusBytes
            );
        BigInteger exponent = new BigInteger(
            1,
            exponentBytes
            );
        BigInteger data = new BigInteger(
            1,
            dataBytes
            );

        //
        // Perform RSA encryption:
        // ciphertext = plaintext^exponent % modulus.
        //
        BigInteger cipherText = data.modPow(
            exponent,
            modulus
            );

        //
        // Reverse the generated ciphertext.
        //
        byte[] cipherTextBytes = cipherText.toByteArray();
        ReverseByteArray(cipherTextBytes);

        //
        // Undo the reversal of the passed in byte arrays.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(exponentBytes);
        ReverseByteArray(dataBytes);

        return cipherTextBytes;
    }

    //
    // Use RSA to decrypt data.
    //
    public static byte[] RsaDecrypt(
        byte[] modulusBytes,
        byte[] privateExponentBytes,
        byte[] encryptedDataBytes
        )
    {
        //
        // Reverse the passed in byte arrays and then use these to
create the
        // BigIntegers for the RSA computation.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(privateExponentBytes);
        ReverseByteArray(encryptedDataBytes);

        BigInteger modulus = new BigInteger(
            1,
            modulusBytes
            );
        BigInteger privateExponent = new BigInteger(
            1,
            privateExponentBytes
            );
        BigInteger encryptedData = new BigInteger(
            1,
            encryptedDataBytes
            );

        //
        // Perform RSA encryption:
        // plaintext = ciphertext^privateExponent % modulus.
        //
        BigInteger decryptedData = encryptedData.modPow(
            privateExponent,
            modulus
            );

        //
        // Reverse the generated plaintext.
        //
        byte[] decryptedDataBytes = decryptedData.toByteArray();
        ReverseByteArray(decryptedDataBytes);

        //
        // Undo the reversal of the passed in byte arrays.
        //
        ReverseByteArray(modulusBytes);
        ReverseByteArray(privateExponentBytes);
        ReverseByteArray(encryptedDataBytes);

        return decryptedDataBytes;
    }

    //
    // Main routine.
    //
    public static void main(
        String[] args
        )
    {
        //
        // Modulus bytes obtained straight from the wire in the
        // proprietary certificate (in little endian format).
        // This is for a 512-bit key set.
        //
        byte[] modulusBytes =
        {
            (byte) 0x3d, (byte) 0x3a, (byte) 0x5e, (byte) 0xbd,
            (byte) 0x72, (byte) 0x43, (byte) 0x3e, (byte) 0xc9,
            (byte) 0x4d, (byte) 0xbb, (byte) 0xc1, (byte) 0x1e,
            (byte) 0x4a, (byte) 0xba, (byte) 0x5f, (byte) 0xcb,
            (byte) 0x3e, (byte) 0x88, (byte) 0x20, (byte) 0x87,
            (byte) 0xef, (byte) 0xf5, (byte) 0xc1, (byte) 0xe2,
            (byte) 0xd7, (byte) 0xb7, (byte) 0x6b, (byte) 0x9a,
            (byte) 0xf2, (byte) 0x52, (byte) 0x45, (byte) 0x95,
            (byte) 0xce, (byte) 0x63, (byte) 0x65, (byte) 0x6b,
            (byte) 0x58, (byte) 0x3a, (byte) 0xfe, (byte) 0xef,
            (byte) 0x7c, (byte) 0xe7, (byte) 0xbf, (byte) 0xfe,
            (byte) 0x3d, (byte) 0xf6, (byte) 0x5c, (byte) 0x7d,
            (byte) 0x6c, (byte) 0x5e, (byte) 0x06, (byte) 0x09,
            (byte) 0x1a, (byte) 0xf5, (byte) 0x61, (byte) 0xbb,
            (byte) 0x20, (byte) 0x93, (byte) 0x09, (byte) 0x5f,
            (byte) 0x05, (byte) 0x6d, (byte) 0xea, (byte) 0x87,
        };

        //
        // Exponent bytes (in little endian order) obtained straight
        // from the wire (in the proprietary certificate).
        //
        byte[] exponentBytes =
        {
            (byte) 0x5b, (byte) 0x7b, (byte) 0x88, (byte) 0xc0
        };

        //
        // Private exponent of the private key generated by the
        // server (in little endian format).
        //
        byte[] privateExponentBytes =
        {
            (byte) 0x87, (byte) 0xa7, (byte) 0x19, (byte) 0x32,
            (byte) 0xda, (byte) 0x11, (byte) 0x87, (byte) 0x55,
            (byte) 0x58, (byte) 0x00, (byte) 0x16, (byte) 0x16,
            (byte) 0x25, (byte) 0x65, (byte) 0x68, (byte) 0xf8,
            (byte) 0x24, (byte) 0x3e, (byte) 0xe6, (byte) 0xfa,
            (byte) 0xe9, (byte) 0x67, (byte) 0x49, (byte) 0x94,
            (byte) 0xcf, (byte) 0x92, (byte) 0xcc, (byte) 0x33,
            (byte) 0x99, (byte) 0xe8, (byte) 0x08, (byte) 0x60,
            (byte) 0x17, (byte) 0x9a, (byte) 0x12, (byte) 0x9f,
            (byte) 0x24, (byte) 0xdd, (byte) 0xb1, (byte) 0x24,
            (byte) 0x99, (byte) 0xc7, (byte) 0x3a, (byte) 0xb8,
            (byte) 0x0a, (byte) 0x7b, (byte) 0x0d, (byte) 0xdd,
            (byte) 0x35, (byte) 0x07, (byte) 0x79, (byte) 0x17,
            (byte) 0x0b, (byte) 0x51, (byte) 0x9b, (byte) 0xb3,
            (byte) 0xc7, (byte) 0x10, (byte) 0x01, (byte) 0x13,
            (byte) 0xe7, (byte) 0x3f, (byte) 0xf3, (byte) 0x5f
        };

        //
        // Sample hash of a proprietary certificate.
        //
        byte[] hashBytes =
        {
            (byte) 0xf5, (byte) 0xcc, (byte) 0x18, (byte) 0xee,
            (byte) 0x45, (byte) 0xe9, (byte) 0x4d, (byte) 0xa6,
            (byte) 0x79, (byte) 0x02, (byte) 0xca, (byte) 0x76,
            (byte) 0x51, (byte) 0x33, (byte) 0xe1, (byte) 0x7f,
            (byte) 0x00, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
            (byte) 0xff, (byte) 0xff, (byte) 0x01
        };

        System.out.println("Hash:");
        PrintBytes(hashBytes);

        //
        // Perform decryption (signing).
        //
        byte[] signedHashBytes = RsaDecrypt(
            modulusBytes,
            privateExponentBytes,
            hashBytes
            );

        System.out.println("Signed hash bytes:");
        PrintBytes(signedHashBytes);
        //
        // Perform encryption (verification).
        //
        byte[] verifiedHashBytes = RsaEncrypt(
            modulusBytes,
            exponentBytes,
            signedHashBytes
            );

        System.out.println("Verified hash bytes:");
        PrintBytes(verifiedHashBytes);
    }
};
```

5 Security

The following sections specify security considerations for implementers of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification.

5.1 Security Considerations for Implementers

See sections 5.3 through 5.5 for complete details of RDP security considerations.

5.2 Index of Security Parameters

None.

5.3 Standard RDP Security

5.3.1 Encryption Levels

Standard RDP Security supports four levels of encryption: Low, Client Compatible, High and FIPS Compliant. The required Encryption Level is configured on the server. All Microsoft RDP servers using Standard RDP Security use one of these four encryption levels and cannot be configured to run under any other level.

1. Low: All data sent from the client to the server is protected by encryption based on the maximum key strength supported by the client.

2. Client Compatible: All data sent between the client and the server is protected by encryption based on the maximum key strength supported by the client.

3. High: All data sent between the client and server is protected by encryption based on the server's maximum key strength. Clients that do not support this level of encryption cannot connect.

4. FIPS: All data sent between the client and server is protected using Federal Information Processing Standard 140-1 validated encryption methods. Clients that do not support this level of encryption cannot connect.

5.3.2 Negotiating the Cryptographic Configuration

Clients advertise their cryptographic support (for use with Standard RDP Security mechanisms) in the Client Security Data (section 2.2.1.3.3), sent to the server as part of the Basic Settings Exchange phase of the connection sequence (see section 1.3.1.1). Upon receiving the client data the server will determine the cryptographic configuration to use for the session based on its configured Encryption Level and then send this selection to the client in the Server Security Data (section 2.2.1.4.3), as part of the Basic Settings Exchange phase. The client will use this information to configure its cryptographic modules.

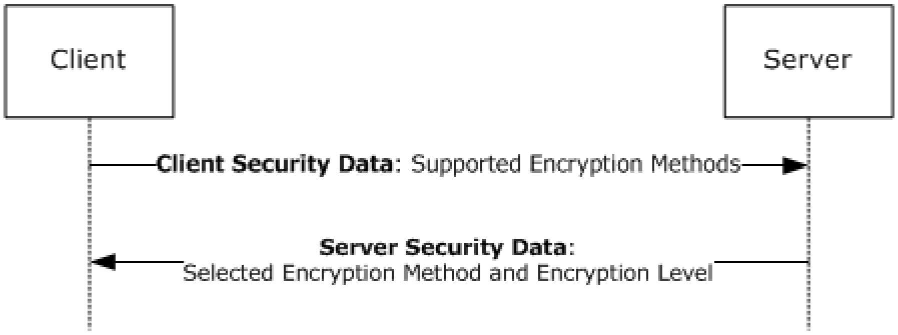

Client
Server
Client Security Data: Supported Encryption Methods
Server Security Data:
Selected Encryption Method and Encryption Level

Figure 6: Determining the cryptographic configuration for a session

The Encryption Method and Encryption Level (see section 5.3.1) are closely related. If the Encryption Level is zero, then the Encryption Method must be zero as well (the converse is also true). Essentially this means that if no encryption is being used for the session (an Encryption Level of zero), then there should be no Encryption Method being applied to the data. If the Encryption Level is greater than zero (encryption is in force for at least client-to-server traffic) then the Encryption Method should be greater than zero (the converse is also true). Essentially this means that if encryption is in force for the session, then an Encryption Method must be defined which specifies how to encrypt the data. Furthermore, if the Encryption Level is set to FIPS, then the Encryption Method should select only FIPS compatible methods.

If the server determines that no encryption is necessary for the session it can send the client a value of zero for the selected Encryption Method and Encryption Level. In this scenario the Security Commencement phase of the connection sequence see section 5.4.2.3 is not executed, with the result that the client does not send the Security Exchange PDU (section 2.2.1.10). This PDU can be dropped as the Client Random (see section 5.3.4) is redundant since no security keys need to be generated. Furthermore, because no security measures are in effect, the Security Header (see section 5.3.8) will not be included with any data sent on the wire, except for the Client Info (see section 3.2.5.3.11) and licensing PDUs (see [MS-RDPELE]) which always contain the Security Header (see section 2.2.9.1.1.2). To protect the confidentiality of user data Microsoft RDP servers never disable encryption on the wire when running with Standard RDP Security enabled.

5.3.3 Server Certificates

5.3.3.1 Proprietary Certificates

Proprietary Certificates are used exclusively by RDP 4.0 servers and servers which have not received an X.509 certificate from a Domain or Enterprise License Server. Every server creates a public/private key pair and then generates and stores a Proprietary Certificate containing the public key at least once at system start-up time. The certificate is only generated when one does not already exist.

Client **SKUs** of the Windows Operating System which support Remote Desktop (such as Windows XP Professional) and Server SKUs (such as Windows Server 2003) running in Remote Administration mode never contact a Licensing Server, and as a result only use Proprietary Certificates. RDP 4.0 clients and servers only support Proprietary Certificates.

The server sends the Proprietary Certificate to the client in the Server Security Data (section 2.2.1.4.3) during the Basic Settings Exchange phase of the connection sequence (see section 1.3.1.1). The Proprietary Certificate structure is detailed in section 2.2.1.4.3.1.1.

5.3.3.1.1 Terminal Services Signing Key

The modulus, private exponent, and public exponent of the 512-bit Terminal Services asymmetric key used for signing Proprietary Certificates with the RSA algorithm are detailed below:

64-byte Modulus (n):

```
0x3d, 0x3a, 0x5e, 0xbd, 0x72, 0x43, 0x3e, 0xc9,
0x4d, 0xbb, 0xc1, 0x1e, 0x4a, 0xba, 0x5f, 0xcb,
0x3e, 0x88, 0x20, 0x87, 0xef, 0xf5, 0xc1, 0xe2,
0xd7, 0xb7, 0x6b, 0x9a, 0xf2, 0x52, 0x45, 0x95,
0xce, 0x63, 0x65, 0x6b, 0x58, 0x3a, 0xfe, 0xef,
0x7c, 0xe7, 0xbf, 0xfe, 0x3d, 0xf6, 0x5c, 0x7d,
0x6c, 0x5e, 0x06, 0x09, 0x1a, 0xf5, 0x61, 0xbb,
0x20, 0x93, 0x09, 0x5f, 0x05, 0x6d, 0xea, 0x87
```

64-byte Private Exponent (d):

```
0x87, 0xa7, 0x19, 0x32, 0xda, 0x11, 0x87, 0x55,
0x58, 0x00, 0x16, 0x16, 0x25, 0x65, 0x68, 0xf8,
0x24, 0x3e, 0xe6, 0xfa, 0xe9, 0x67, 0x49, 0x94,
0xcf, 0x92, 0xcc, 0x33, 0x99, 0xe8, 0x08, 0x60,
0x17, 0x9a, 0x12, 0x9f, 0x24, 0xdd, 0xb1, 0x24,
0x99, 0xc7, 0x3a, 0xb8, 0x0a, 0x7b, 0x0d, 0xdd,
0x35, 0x07, 0x79, 0x17, 0x0b, 0x51, 0x9b, 0xb3,
0xc7, 0x10, 0x01, 0x13, 0xe7, 0x3f, 0xf3, 0x5f
```

4-byte Public Exponent (e):

```
0x5b, 0x7b, 0x88, 0xc0
```

The enumerated integers are in little-endian byte order. The public key is the pair (e, n), while the private key is the pair (d, n).

5.3.3.1.2 Signing a Proprietary Certificate

The Proprietary Certificate is signed by using RSA to encrypt the hash of the first six fields with the Terminal Services private signing key (specified in section 5.3.3.1.1) and then appending the result to the end of the certificate. Mathematically the signing operation is formulated as follows:

```
s = m^d mod n
```

Where,

```
s = signature;
m = hash of first six fields of certificate
d = private exponent; n = modulus
```

The structure of the Proprietary Certificate is detailed in section 2.2.1.4.3.1.1. The structure of the public key embedded in the certificate is described in 2.2.1.4.3.1.1.1. An example of public key bytes (in little-endian format) follows:

```
0x52 0x53 0x41 0x31: magic (0x31415352)
0x48 0x00 0x00 0x00: keylen (72 bytes)
0x00 0x02 0x00 0x00: bitlen (512 bits)
0x3f 0x00 0x00 0x00: datalen (63 bytes)
0x01 0x00 0x01 0x00: pubExp (0x00010001)


0xaf 0xfe 0x36 0xf2 0xc5 0xa1 0x44 0x2e
0x47 0xc1 0x31 0xa7 0xdb 0xc6 0x67 0x02
0x64 0x71 0x5c 0x00 0xc9 0xb6 0xb3 0x04
0xd0 0x89 0x9f 0xe7 0x6b 0x24 0xe8 0xe8
0xe5 0x2d 0x0b 0x13 0xa9 0x0c 0x6d 0x4d
0x91 0x5e 0xe8 0xf6 0xb3 0x17 0x17 0xe3
0x9f 0xc5 0x4d 0x4a 0xba 0xfa 0xb9 0x2a
0x1b 0xfb 0x10 0xdd 0x91 0x8c 0x60 0xb7: modulus
```

A 128-bit MD5 hash over the first six fields of the proprietary certificate (which are all in little-endian format) appears as follows:

```
PublicKeyBlob = wBlobType + wBlobLen + PublicKeyBytes
hash = MD5(dwVersion + dwSigAlgID + dwKeyAlgID + PublicKeyBlob)
```

An array of 63 bytes is then created and initialized as follows:

```
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0x00 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0x01
```

The 128-bit MD5 hash is copied into the first sixteen bytes of the array. For example, assume that the generated hash is:

```
0xf5 0xcc 0x18 0xee 0x45 0xe9 0x4d 0xa6
0x79 0x02 0xca 0x76 0x51 0x33 0xe1 0x7f
```

The byte array will appear as follows after copying in the sixteen bytes of the MD5 hash:

```
0xf5 0xcc 0x18 0xee 0x45 0xe9 0x4d 0xa6
0x79 0x02 0xca 0x76 0x51 0x33 0xe1 0x7f
```

```
0x00 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0x01
```

The 63-byte array is then treated as an unsigned little-endian integer and signed with the Terminal Services private key by using RSA. The resultant signature must be in little-endian format before appending it to the Proprietary Certificate structure. The final structure of the certificate must conform to the specification in section 2.2.1.4.3.1.1. This means that fields 7 through to 9 will be the signature blob type, the number of bytes in the signature and the actual signature bytes respectively. The blob type and number of bytes must be in little-endian format.

5.3.3.1.3 Validating a Proprietary Certificate

Verification of the Proprietary Certificate signature is carried out by decrypting the signature with the Terminal Services public signing key and then verifying that this result is the same as the MD5 hash of the first six fields of the certificate.

```
m = s^e mod n
```

Where,

```
m = decrypted signature; s = signature
e = public exponent; n = modulus
```

The structure of the Proprietary Certificate is detailed in section 2.2.1.4.3.1.1. A 128-bit MD5 hash over the first six fields (which are all little-endian integers of varying lengths) appears as follows:

```
PublicKeyBlob = wBlobType + wBlobLen + PublicKeyBytes
hash = MD5(dwVersion + dwSigAlgID + dwKeyAlgID + PublicKeyBlob)
```

Next, the actual signature bytes are treated as an unsigned little-endian integer and decrypted with the Terminal Services public key by using RSA. The bytes which result from the decryption must be sorted in little-endian order to form an array of 63 bytes. The 17th byte of the array should be 0x00, the 18th through to the 62nd byte should be 0xFF, while the 63rd byte should be 0x01. An example of a successfully decrypted signature is:

```
0xf5 0xcc 0x18 0xee 0x45 0xe9 0x4d 0xa6
0x79 0x02 0xca 0x76 0x51 0x33 0xe1 0x7f
0x00 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
0xff 0xff 0xff 0xff 0xff 0xff 0x01
```

The first sixteen bytes of the array are then compared to the hash which was generated over the Proprietary Certificate and if they match, the signature has been successfully verified.

5.3.3.2 X.509 Certificate Chains

X.509-compliant certificates are issued to servers upon request by Domain or Enterprise License Servers and are required to issue client licenses (see [MS-RDPELE] for more information on RDP Licensing). An X.509 Certificate Chain consists of a collection of certificates concatenated together in root-certificate-first order. This eliminates the need to scan the chain to the end to get the root certificate for starting chain validation. The last certificate is the certificate of the server; the second-to-last is the license server's certificate, and so forth. More details on the structure of the chain and the component certificates are in section of [MS-RDPELE]

Servers send the X.509 Certificate Chain to clients in the Server Security Data (section 2.2.1.4.3) settings block during the Basic Settings Exchange phase of the connection sequence (see section 1.3.1.1). A server which has not yet been issued an X.509 Certificate Chain will fall back to using a Proprietary Certificate (section 2.2.1.4.3.1.1). Proprietary Certificates are always used when an RDP 4.0 client connects to a server (the client version can be determined from the Client Core Data (section 2.2.1.3.2).

5.3.4 Client and Server Random Values

The client and server both generate a 32-byte random value using a cryptographically-safe pseudorandom number generator.

The server sends the random value which it generated (along with its public key embedded in a certificate) to the client in the Server Security Data (section 2.2.1.4.3) during the Basic Settings Exchange phase of the connection sequence (see section 1.3.1.1).

If RDP Standard Security is in effect and encryption is being used, then the client sends its random value to the server (encrypted with the server's public key) in the Security Exchange PDU (section 2.2.1.10) as part of the RDP Security Commencement phase of the connection sequence (see section 1.3.1.1).

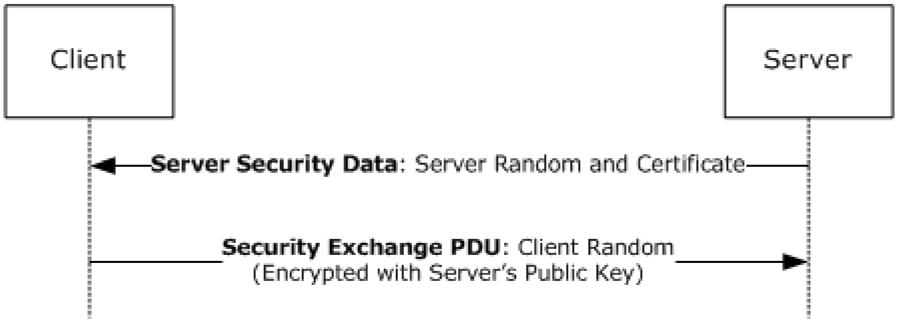

Client
Server
Server Security Data: Server Random and Certificate
Security Exchange PDU: Client Random
(Encrypted with Server's Public Key)

Figure 7: Client and Server Random Value Exchange

The two random values are used by the client and server to generate session keys to secure the connection.

5.3.4.1 Encrypting Client Random

The client random is encrypted by the client with the server's public key (obtained from the Server Security Data (section 2.2.1.4.3)) using RSA. Mathematically the encryption operation is formulated as follows:

```
c = r^e mod n
```

Where,

```
c = encrypted client random; r = unencrypted client random
e = public exponent; n = modulus
```

The client random value should be interpreted as an unsigned little-endian integer value when performing the encryption. The resultant encrypted client random should be copied into a zeroed out buffer which is of size:

```
(bitlen / 8) + 8
```

For example, if the public key of the server is 512 bits long, then the zeroed out buffer should be 72 bytes. This value can also be obtained from the keylen field in the public key structure (see section 2.2.1.4.3.1.1.1). The buffer is sent to the server in the Security Exchange PDU (section 2.2.1.10).

5.3.4.2 Decrypting Client Random

The server can decrypt the client random as it possesses the private exponent of the public/private key pair which it generated. Mathematically the decryption operation is formulated as follows:

```
r = c^d mod n
```

Where,

```
r = unencrypted client random; c = encrypted client random;
d = private exponent; n = modulus
```

The encrypted client random is obtained from the Security Exchange PDU (section 2.2.1.10). The encrypted client random value should be interpreted as an unsigned little-endian integer value when performing the decryption operation.

5.3.5 Session Key Generation

RDP uses three symmetric session keys derived from the client and server random values (see section 5.3.4). Client-to-server traffic is encrypted with one of these keys (known as the client's encryption key and server's decryption key), server-to-client traffic with another (known as the server's encryption key and client's decryption key) and the final key is used to generate a message authentication code (MAC) over the data to help ensure its integrity. The generated keys are 40, 56 or 128 bits in length.

5.3.5.1 Non-FIPS

The client and server random values are used to create a 384-bit Pre-Master Secret by concatenating the first 192 bits of the Client Random with the first 192 bits of the Server Random.

```
PreMasterSecret = First192Bits(ClientRandom) + First192Bits(ServerRandom)
```

A 384-bit Master Secret is generated using the Pre-Master Secret, the client and server random values and the MD5 and SHA-1 hash functions.

```
SaltedHash(S, I) = MD5(S + SHA(I + S + ClientRandom + ServerRandom))
PreMasterHash(I) = SaltedHash(PremasterSecret, I)

MasterSecret = PreMasterHash('A') + PreMasterHash('BB') + PreMasterHash('CCC')
```

A 384-bit Session Key Blob is generated as follows:

```
MasterHash(I) = SaltedHash(MasterSecret, I)
SessionKeyBlob = MasterHash('X') + MasterHash('YY') + MasterHash('ZZZ')
```

From the Session Key Blob the actual session keys which will be used are derived. Both client and server extract the same key data for generating MAC signatures.

```
MACKey128 = First128Bits(SessionKeyBlob)
```

The initial encryption and decryption keys are generated next (these keys are updated at a later point in the protocol per section 5.3.6.1). The server generates its encryption and decryption keys as follows:

```
FinalHash(K) = MD5(K + ClientRandom + ServerRandom)

InitialServerEncryptKey128 = FinalHash(Second128Bits(SessionKeyBlob))
InitialServerDecryptKey128 = FinalHash(Third128Bits(SessionKeyBlob))
```

The client constructs its initial decryption key with the bytes which the server uses to construct its initial encryption key. Similarly, the bytes which the server uses to form its initial decryption key, the client uses to form its initial encryption key.

```
InitialClientDecryptKey128 = FinalHash(Second128Bits(SessionKeyBlob))
InitialClientEncryptKey128 = FinalHash(Third128Bits(SessionKeyBlob))
```

This means that the client will use its encryption key to encrypt data and the server will use its decryption key to decrypt the same data. Similarly, the server will use its encryption key to encrypt data and the client will use its decryption key to decrypt the same data. In effect, there are two streams of data (client-to-server and server-to-client) encrypted with different session keys which are updated at different intervals.

To reduce the entropy of the keys to either 40 or 56 bits, the 128-bit client and server keys are salted appropriately to produce 64-bit versions with the required strength, as shown in the following table.

Negotiated Key Length	Salt Length	Salt Values	RC4 Key Length
40 bits	3 bytes	0xD1, 0x26, 0x9E	8 bytes
56 bits	1 byte	0xD1	8 bytes
128 bits	0 bytes	N/A	16 bytes

Using the salt values, the 40-bit keys are generated as follows:

```
MACKey40 = 0xD1269E + Last40Bits(First64Bits(MACKey128))
InitialEncryptKey40 = 0xD1269E + Last40Bits(First64Bits(InitialEncryptKey128))
InitialDecryptKey40 = 0xD1269E + Last40Bits(First64Bits(InitialDecryptKey128))
```

The 56-bit keys are generated as follows:

```
MACKey56 = 0xD1269E + Last56Bits(First64Bits(MACKey128))
InitialEncryptKey56 = 0xD1 + Last56Bits(First64Bits(InitialEncryptKey128))
InitialDecryptKey56 = 0xD1 + Last56Bits(First64Bits(InitialDecryptKey128))
```

After any necessary salting has been applied, the generated encryption and decryption keys are used to initialize RC-4 substitution tables which can then be used to encrypt and decrypt data.

At the end of this process the client and server will each possess three symmetric keys to use with the RC4 stream cipher: a MAC key, encryption key and decryption key.

5.3.5.2 FIPS

The client and server random values are used to generate temporary 160-bit initial encryption and decryption keys by using the SHA-1 hash function. The client generates the following:

```
ClientEncryptKeyT = SHA(First128Bits(ClientRandom) + First128Bits(ServerRandom))
ClientDecryptKeyT = SHA(Last128Bits(ClientRandom) + Last128Bits(ServerRandom))
```

The server generates the following:

```
ServerDecryptKeyT = SHA(First128Bits(ClientRandom) + First128Bits(ServerRandom))
ServerEncryptKeyT= SHA(Last128Bits(ClientRandom) + Last128Bits(ServerRandom))
```

To expand the keys to 168 bits in length (which is the standard length for a Triple DES key), the first eight bits of each key are copied to the rear of the key.

```
ClientEncryptKey = ClientEncryptKeyT + First8Bits(ClientEncryptKeyT)
ClientDecryptKey = ClientDecryptKeyT + First8Bits(ClientDecryptKeyT)

ServerDecryptKey = ServerDecryptKeyT + First8Bits(ServerDecryptKeyT)
ServerEncryptKey= ServerEncryptKeyT + First8Bits(ServerEncryptKeyT)
```

The shared secret key to be used with SHA-1 to produce an HMAC (see [RFC2104]) is computed by the client as follows:

```
HMACKey = SHA(ClientEncryptKeyT + ClientDecryptKeyT)
```

The server performs the same computation with the same data:

```
HMACKey = SHA(ServerDecryptKeyT + ServerEncryptKeyT)
```

At the end of this process the client and server will each possess three symmetric keys to use with the Triple DES block cipher: an HMAC key, encryption key and decryption key.

5.3.6 Encrypting and Decrypting the I/O Data Stream

If the Encryption Level (see section 5.4.1) of the server is greater than zero (all Microsoft RDP servers enforce this), then encryption will always be in effect. At a minimum, all client-to-server traffic (except for licensing PDUs which have optional encryption) will be encrypted and a MAC will be appended to the data to help ensure transmission integrity.

The table which follows summarizes the possible encryption and MAC generation scenarios based on the Encryption Method and Encryption Level selected by the server (the Encryption Method values are described in section 2.2.1.4.3, while the Encryption Levels are described in 5.4.1) as part of the cryptographic negotiation described in section 5.3.2:

Selected Encryption Method	Selected Encryption Level	Data Encryption	MAC Generation
None (0x00)	None (0)	None	None
40-Bit (0x01) 56-Bit (0x08) 128-Bit (0x02)	Low (1)	Client-to-server traffic only using RC4	Client-to-server traffic only using MD5 and SHA-1
40-Bit (0x01) 56-Bit (0x08) 128-Bit (0x02)	Client Compatible (2) High (3)	Client-to-server and server-to-client traffic using RC4	Client-to-server and server-to-client traffic using MD5 and SHA-1
FIPS (0x10)	FIPS (4)	Client-to-server and server-to-client traffic using Triple DES	Client-to-server and server-to-client traffic using SHA-1

5.3.6.1 Non-FIPS

The client and server follow the same series of steps to encrypt a block of data. First, a MAC value is generated over the unencrypted data.

```
Pad1 = 0x36 repeated 40 times to give 320 bits
Pad2 = 0x5C repeated 48 times to give 384 bits

SHAComponent = SHA(MACKeyN + Pad2 + DataLength + Data)
MACSignature = First64Bits(MD5(MACKeyN + Pad1 + SHAComponent))

(MACKeyN is either MACKey40, MACKey56 or MACKey128, depending on the negotiated
   key strength.)
```

DataLength is the size of the data to encrypt in bytes, expressed as a little-endian 32-bit integer. Data is the information to be encrypted. The first 8 bytes of the generated MD5 hash are used as an 8-byte MAC value to send on the wire.

Next, the data block is encrypted with RC4 using the current client or server encryption substitution table. The encrypted data is appended to the 8-byte MAC value in the network packet.

Decryption involves a reverse ordering of the previous steps. First the data is decrypted using the current RC4 decryption substitution table. Then, a 16-byte MAC value is generated over the decrypted data, and the first 8 bytes of this MAC are compared to the 8-byte MAC value that was sent over the wire. If the MAC values do not match, an appropriate error is generated and the connection is dropped.

5.3.6.1.1 Salted MAC Generation

The MAC value may be generated by salting the data to be hashed with the current encryption count. For example, assume that 42 packets have already been encrypted. When the next packet is encrypted the value 42 is added to the SHA component of the MAC signature. The addition of the encryption count can be expressed as follows:

```
SHAComponent = SHA(MACKeyN + Pad2 + DataLength + Data + EncryptionCount)
MACSignature = First64Bits(MD5(MACKeyN + Pad1 + SHAComponent))
```

EncryptionCount is the cumulative encryption count, indicating how many encryptions have been carried out. It is expressed as a little-endian 32-bit integer. The descriptions for DataLength, Data, and MacKeyN are the same as in section 5.3.6.1.

The use of the salted MAC is dictated by capability flags in the General Capability Set (section 2.2.7.1.1), sent by both client and server during the Capability Negotiation phase of the connection sequence (see section 1.3.1.1). In addition, the presence of a salted MAC is indicated by the presence of the TS_ENC_SECURE_CHECKSUM flag in the Security Header flags field (see section 5.3.8).

5.3.6.2 FIPS

Prior to performing an encryption or decryption operation, the cryptographic modules used to implement Triple DES must be configured with the following Initialization Vector:

```
{0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF}
```

The 160-bit MAC signature key is used to key the HMAC function (see [RFC2104]), which uses SHA-1 as the iterative hash function.

```
MACSignature = First64Bits(HMAC(HMACKey, Data + EncryptionCount))
```

EncryptionCount is the cumulative encryption count, indicating how many encryptions have been carried out. It is expressed as a little-endian 32-bit integer. The description for Data is the same as in section 5.3.6.1.

Encryption of the data and construction of the network packet to transmit is similar to section 5.3.6.1. The main difference is that Triple DES (in CBC mode) is used. Because DES is a block cipher the data to be encrypted must be padded to be a multiple of the block size (8 bytes). The FIPS Security Header (see sections 2.2.8.1 and 2.2.9.1) has an extra field to record the number of padding bytes which were appended to the data prior to encryption to ensure that upon decryption these bytes are not included as part of the data.

5.3.7 Session Key Updates

During the course of a session, the symmetric encryption and decryption keys may need to be refreshed.

5.3.7.1 Non-FIPS

The encryption and the decryption keys are updated after 4096 packets have been sent or received. The input and output streams are considered separate for the purposes of counting packets. Generating a new session key involves the use of the initial session key, the current session key, and the RC4 key length from the salt table based on the negotiated key length (Table 1).

The following sequence of steps shows how new client and server encryption keys are generated (the same steps are used to generate new client and server decryption keys). The following padding constants are used:

```
Pad1 = 0x36 repeated 40 times to give 320 bits
Pad2 = 0x5C repeated 48 times to give 384 bits
```

If the negotiated key strength is 40 or 56-bit, then the first 64 bits of the initial session key and the current session key will be used.

```
InitialEncryptKey = First64Bits(InitialEncryptKeyN)
CurrentEncryptKey = First64Bits(CurrentEncryptKeyN)

(InitialEncryptKeyN is either InitialEncryptKey40 or InitialEncryptKey56, depending
 on the negotiated key strength, while CurrentEncryptKeyN is either CurrentEncryptKey40
 or CurrentEncryptKey56, depending on the negotiated key strength.)
```

If the negotiated key strength is 128-bit, then the full 128 bits of the initial and current key will be used.

```
InitialEncryptKey = InitialEncryptKey128
CurrentEncryptKey = CurrentEncryptKey128
```

The initial and current keys are concatenated and hashed together with padding to form a temporary session key as follows:

```
SHAComponent = SHA(InitialEncryptKey + Pad2 + CurrentEncryptKey)
EncryptKey128T = MD5(InitialEncryptKey + Pad1 + SHAComponent)
```

If the key strength is 128 bits then the temporary session key (EncryptKey128T) is used to reinitialize the associated RC4 substitution table and then RC4 is used to encrypt the temporary key to obtain the new encryption key. Finally, the RC4 substitution table is reinitialized with the new encryption key which can then be used to encrypt a further 4096 packets.

If 40 or 56-bit keys are being used, then the first 64 bits of the temporary session key (EncryptKey128T) are used to initialize the associated RC4 substitution table. RC4 is then used to encrypt these 64 bits and the first few bytes are salted according to the key strength to derive a new session key (see section 5.3.7.1 for details on how to perform the salting operation). Finally, the new 40 or 56-bit session key is used to reinitialize the RC4 substitution table.

5.3.7.2 FIPS

No session key updates take place for the duration of a connection if Standard RDP Security is being used with a FIPS Encryption Level.

5.3.8 Packet Layout in the I/O Data Stream

The usage of Standard RDP Security (see section 5.3) results in a security header being present in all packets following the Security Exchange PDU (section 2.2.1.10) (when encryption is in force). Connection sequence PDUs following the RDP Security Commencement phase of the connection sequence (see section 1.3.1.1) and Slow-Path packets have the same general wire format:

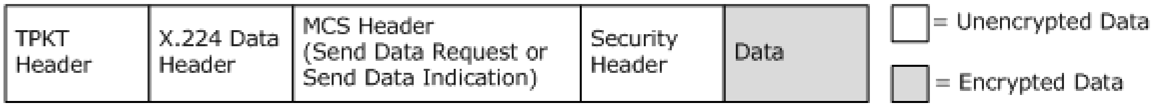


Figure 8: Slow-Path packet layout

The Security Header essentially contains flags and a MAC signature taken over the encrypted data (see section 5.3.6 for details on the MAC generation). In FIPS scenarios, the header also includes the number of padding bytes appended to the data.

Fast-Path packets are more compact and formatted differently, but the essential contents of the Security Header are still present. For non-FIPS scenarios the packet layout is:

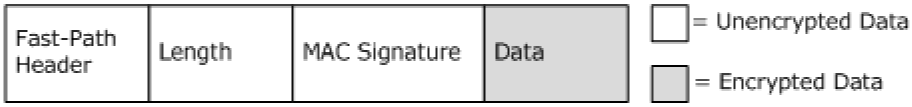


Figure 9: Non-FIPS Fast-Path packet layout

And in FIPS Fast-Path scenarios the packet layout is:

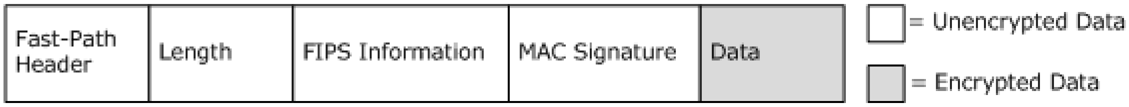


Figure 10: FIPS Fast-Path packet layout

If no encryption is in effect the Selected Encryption Method and Encryption Level (see section 5.3.1) returned to the client is zero) the Security Header will not be included with any data sent on the wire, except for the Client Info (section 2.2.1.11) and licensing PDUs (for an example of a licensing PDU see section 2.2.1.12) which always contain the Security Header.

See sections 2.2.8.1 and 2.2.9.1 for more details on Slow and Fast-Path packet formats and the structure of the Security Header in both of these scenarios.

5.4 Enhanced RDP Security

When Enhanced RDP Security is used, RDP traffic is no longer protected by using the techniques described in section 5.3. Instead, all security operations (such as encryption, data integrity checks and server authentication) are delegated to an External Security Protocol layer. Examples of such layers are TLS (see [RFC2246]) and CredSSP (see [MS-CSSP]). The benefit of this approach is that RDP developers no longer need to manually implement protocol security mechanisms, but can instead rely on well-known and proven security protocol packages (such as the Schannel Security Package which implements SSL, see [MSDN-SCHANNEL]) to provide end-to-end security.

5.4.1 Encryption Levels

Enhanced RDP Security (see section 5.4) supports a subset of the encryption levels used by Standard RDP Security (see section 5.3.1). The required Encryption Level is configured on the server. All Microsoft RDP servers using Enhanced RDP Security use one of these three encryption levels and cannot be configured to run under any other level.

1. Client Compatible: All data sent between the client and the server is protected using encryption techniques negotiated through mechanisms defined by the negotiated security protocol.

2. High: All data sent between the client and the server is protected using encryption techniques which employ at least a 128-bit symmetric key negotiated through mechanisms defined by the negotiated security protocol. The server enforces the key strength and clients that do not support 128-bit symmetric keys cannot connect.

3. FIPS: All data sent between the client and server is protected by the negotiated security protocol using the following Federal Information Processing Standard 140-1 validated methods: RSA for key exchange, Triple DES for bulk encryption and SHA-1 for any hashing operations. Clients that do not support these methods cannot connect.

When a client connects to a server configured for Enhanced RDP Security, the selected encryption level returned to the client is not the configured server encryption level, but rather the value ENCRYPTION_LEVEL_NONE (0). This is due to the fact that the encryption for the session is provided by an External Security Protocol and the Standard RDP Security Protocol mechanisms are disabled.

5.4.2 Security-Enhanced Connection Sequence

When Enhanced RDP Security (see section 5.4) is being used, the connection sequence is changed to incorporate the possible use of an External Security Protocol (see section 5.4.5). A brief overview of the connection sequence changes are described in section 1.3.1.2. The two variations of the Security-Enhanced Connection Sequence are the Negotiation-Based Approach (see section 5.4.2.1) and the Direct Approach (see section 5.4.2.2).

5.4.2.1 Negotiation-Based Approach

The client advertises the security protocols which it supports by appending an RDP Negotiation Request (section 2.2.1.1.1) structure to the X.224 Connection Request PDU (section 2.2.1.1).

Upon receipt of the RDP Negotiation Request, the server examines the client request and selects the protocol to use. The server indicates its response to the client by appending an RDP Negotiation Response (section 2.2.1.2.1) structure to the X.224 Connection Confirm PDU (section 2.2.1.2). If the server does not support any of the protocols requested by the client, or if there was an error setting up the External Cryptographic Protocol Provider, then the server appends an RDP Negotiation Failure (section 2.2.1.2.2) structure to the X.224 Connection Confirm PDU.

If the client accepts the server's choice of security protocol as indicated in the RDP Negotiation Response (and assuming that the security protocol is not using Standard RDP Security (see section 5.3) mechanisms) it instantiates the security protocol by calling into an External Cryptographic Protocol Provider. Once the External Security Protocol handshake has successfully run to completion, the RDP messages resume, continuing with the MCS Connect Initial PDU. From this point all RDP traffic is encrypted using the External Security Protocol.

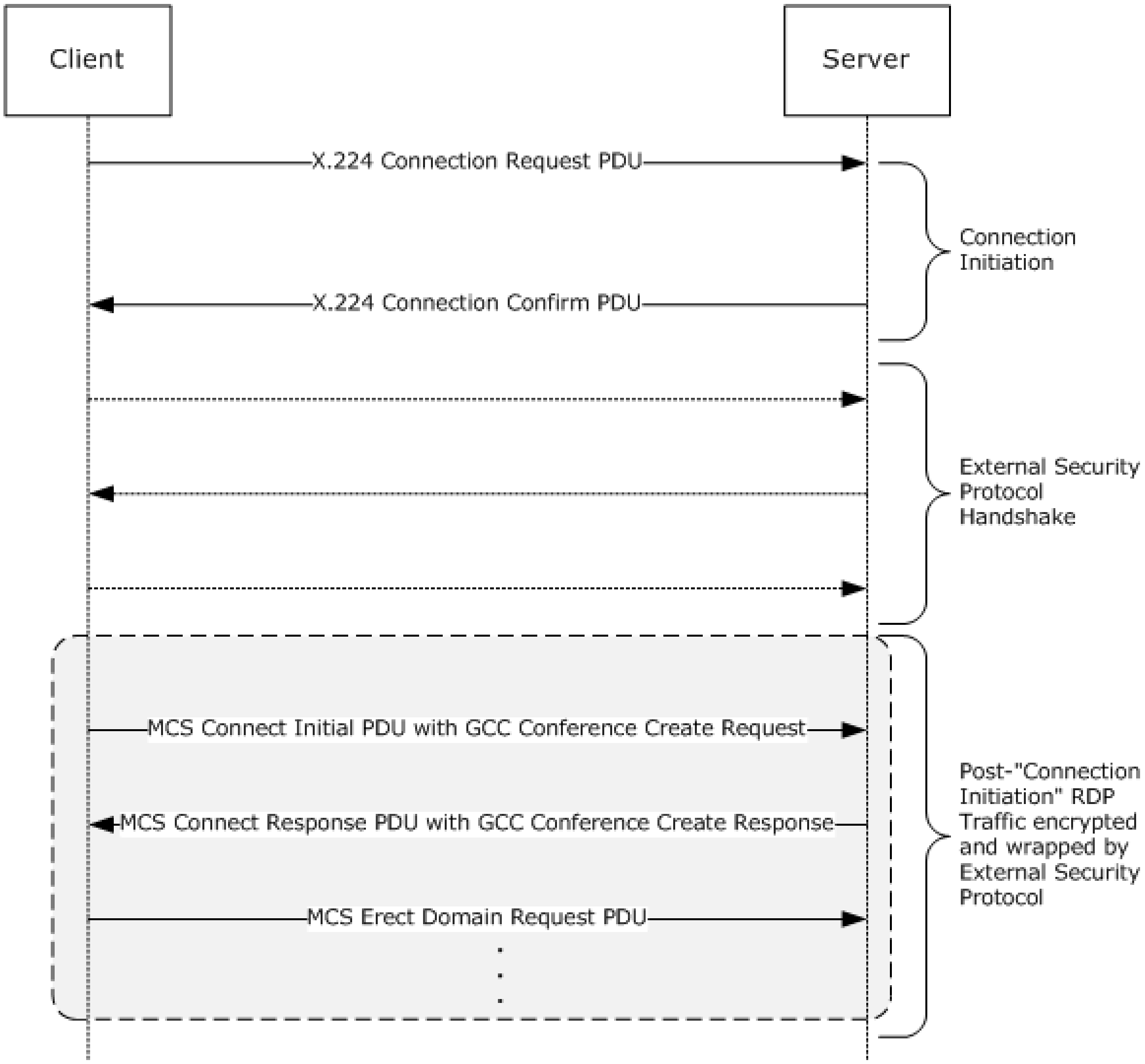

Client
Server
X.224 Connection Request PDU
X.224 Connection Confirm PDU
Connection Initiation
External Security Protocol Handshake
MCS Connect Initial PDU with GCC Conference Create Request
MCS Connect Response PDU with GCC Conference Create Response
MCS Erect Domain Request PDU
Post-"Connection Initiation" RDP Traffic encrypted and wrapped by External Security Protocol

Figure 11: Negotiation-based security-enhanced connection sequence

Since both the RDP Negotiation Request and RDP Negotiation Response are initially exchanged in the clear, they are re-exchanged in the reverse direction after the External Security Protocol handshake as part of the Basic Settings Exchange phase of the RDP connection sequence (see section 1.3.1.1). This step ensures that no tampering has taken place. The client replays the server's protocol choice in the Client Core Data (section 2.2.1.3.2), while the server replays the client's requested protocols in the Server Core Data (section 2.2.1.4.2).

5.4.2.2 Direct Approach

The Negotiation-Based Approach (specified in section 5.4.2.1) aims to have the client and server agree on a security protocol to use for the connection. The fact that the X.224 messages are unencrypted helps to ensure backward compatibility with prior versions of RDP servers, as the packets can always be read. However, the fact that the X.224 PDUs are unencrypted is also a threat, as an attacker can seek to compromise or take down the server by sending malformed X.224 PDUs. Hence the goal of the Direct Approach is to ensure that all RDP traffic is protected.

When using the Direct Approach, no negotiation of the security protocol takes place. The client and server are hard-coded to use a specific security protocol when a connection is initiated. Once the security protocol handshake has completed successfully, the RDP Connection Sequence begins, starting with the X.224 messages which form the Connection Initiation phase (see section 1.3.1.1). From this point all RDP traffic is encrypted using the security protocol.

The RDP Negotiation Request (section 2.2.1.1.1) must still be appended to the X.224 Connection Request PDU and the requested protocol list must contain the identifier of the hard-coded security protocol which is being used. If this is not the case, the server will append an RDP Negotiation Failure to the X.224 Connection Confirm PDU with a failure code of INCONSISTENT_FLAGS (0x04). Similarly, the server must indicate that the hard-coded security protocol is the selected protocol in the RDP Negotiation Response which is appended to the X.224 Connection Confirm PDU.

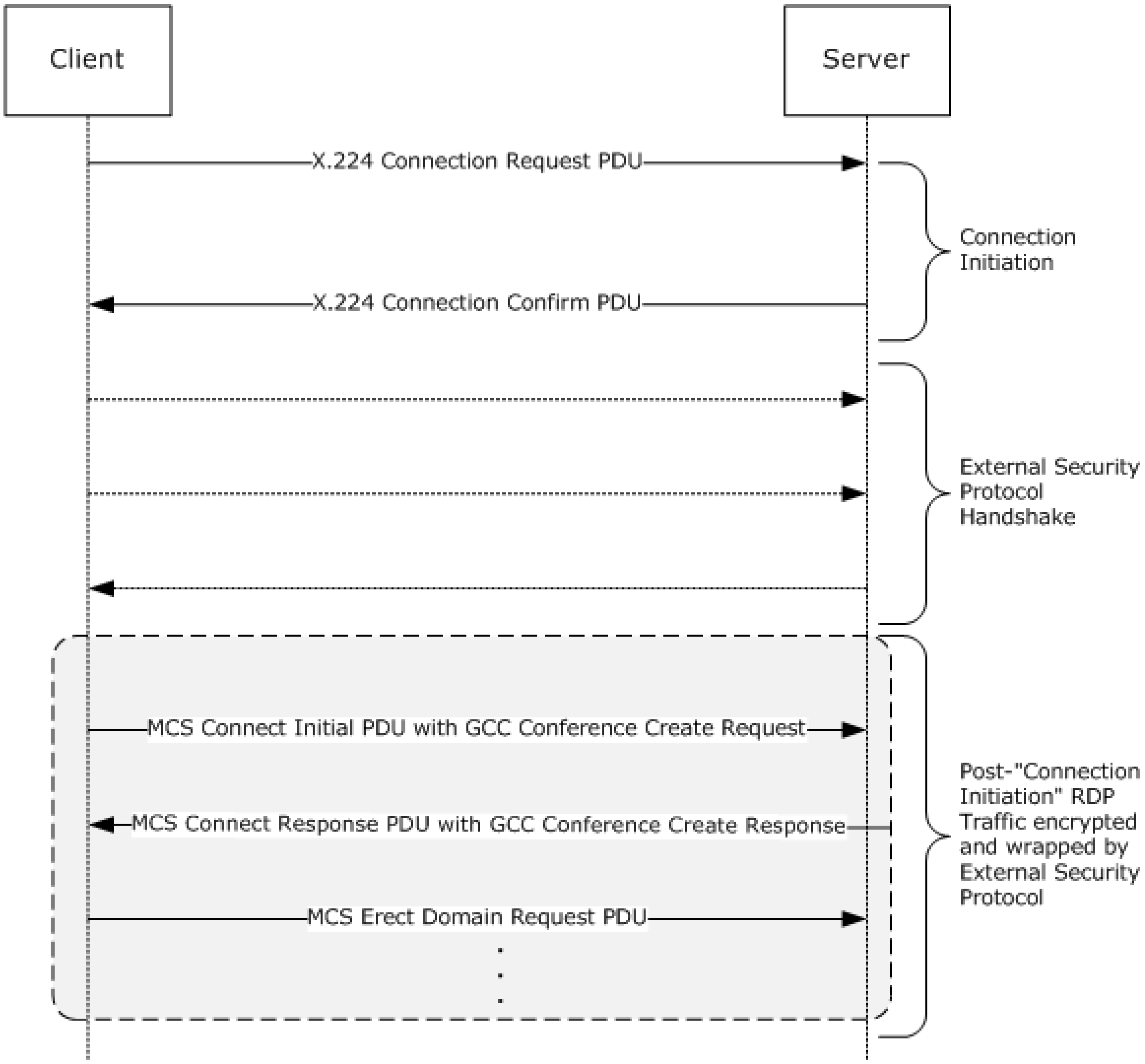

Client
Server
X.224 Connection Request PDU
X.224 Connection Confirm PDU
Connection Initiation
External Security Protocol Handshake
MCS Connect Initial PDU with GCC Conference Create Request
MCS Connect Response PDU with GCC Conference Create Response
MCS Erect Domain Request PDU
Post-"Connection Initiation" RDP Traffic encrypted and wrapped by External Security Protocol

Figure 12: Direct security-enhanced connection sequence

As specified in the Negotiation-Based Approach (see section 5.4.2.1), the client and server must also confirm the selected protocol and the requested protocols in the Client Core Data (section 2.2.1.3.2) and Server Core Data (section 2.2.1.4.2) respectively.

5.4.2.3 Changes to the Security Commencement Phase

Because Standard RDP Security is not in effect, the Security Commencement phase of the connection sequence (see section 1.3.1.1) is not executed, with the result that the client does not send the Security Exchange PDU (section 2.2.1.10). This PDU can be dropped as the Client Random is redundant in this case because encryption for the connection is provided by the External Security Protocol and not by the Standard RDP Security mechanisms specified in section 5.3.

5.4.2.4 Disabling Forced Encryption of Licensing Packets

Encryption of licensing PDUs is optional when Standard RDP Security is being used. However, if an External Security Protocol is being employed, then the server and client do not need to ever encrypt any licensing packets as the External Security Protocol will encrypt them. For this reason, the SEC_LICENSE_ENCRYPT_CS (0x0200) and SEC_LICENSE_ENCRYPT_SC (0x0200) flags (see section 2.2.8.1.1.2.1) do not need to be set in the Security Header which is always attached to licensing packets.

5.4.3 Encrypting and Decrypting the I/O Data Stream

Encryption and decryption of RDP traffic is only carried out by the External Security Protocol layer. There is no double-encryption of data which takes place.

5.4.4 Packet Layout in the I/O Data Stream

Because RDP encryption is not used in the presence of an External Security Protocol layer, the security header data (see section 5.4.4) is not present in any RDP traffic (except for the Client Info and licensing PDUs). All of the RDP traffic which is encrypted by the External Security Protocol is wrapped by headers determined by the protocol specification.

For example, if SSL is used as the External Security Protocol, an encrypted RDP Slow-Path packet would appear as follows:

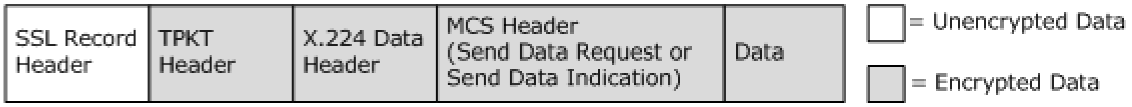


Figure 13: Encrypted slow-path packet

A Fast-Path packet would appear as follows if SSL is the External Security Protocol:

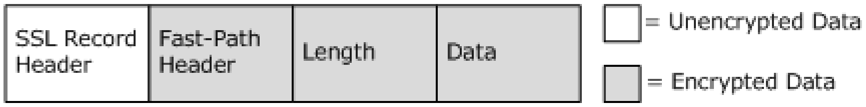


Figure 14: Encrypted fast-path packet

Notice that in both of these cases, the security header data is missing. See sections 2.2.8.1 and 2.2.9.1 for more details on Slow and Fast-Path packet formats.

5.4.5 External Security Protocols used by RDP

RDP supports two External Security Protocols: TLS 1.0 (see [RFC2246]) and the Credential Security Support Provider (CredSSP) Protocol (see [MS-CSSP]). Both TLS and CredSSP protocols require external infrastructure, such as server authentication certificates (TLS and CredSSP) or Key Distribution Centers (CredSSP), to run successfully. These resources are opaque to RDP and left to implementers to provide, set up and maintain.

5.4.5.1 Transport Layer Security (TLS) 1.0

TLS 1.0 is represented by the TS_NEG_PROTOCOL_SSL flag in the RDP Negotiation Request (section 2.2.1.1.1) and RDP Negotiation Response (section 2.2.1.2.1) structures. TLS 1.0 is derived from SSL 3.0 (see [SSL3]) and was added to RDP primarily to enable server authentication so as to mitigate man-in-the-middle attacks on RDP traffic.

5.4.5.2 CredSSP

CredSSP is represented by the TS_NEG_PROTOCOL_HYBRID flag in the RDP Negotiation Request (section 2.2.1.1.1) and RDP Negotiation Response (section 2.2.1.2.1) structures. The Credential Security Support Provider (CredSSP) Protocol is essentially the amalgamation of TLS with Kerberos and NTLM. Besides enabling server authentication, the Credential Security Support Provider (CredSSP) Protocol also facilitates user authentication and the transfer of user credentials from client to server, hence enabling single-sign-on scenarios.

When the Credential Security Support Provider (CredSSP) Protocol begins execution, the TLS handshake will always be executed. Once a TLS channel has been successfully established (the identity of the server may have been authenticated in the process), Kerberos or NTLM will be used within the TLS channel to authenticate the user (and possibly the server as well if Kerberos is being used). Once Kerberos or NTLM has completed successfully, the user's credentials are sent to the server. Traffic on the wire remains encrypted with TLS and is wrapped by TLS headers. There is no double-encryption of traffic as the Kerberos (or NTLM) session is securely bound to the TLS session.

5.4.5.2.1 User Authorization Failures

In Microsoft RDP server implementations, user authorization needs to happen on the server prior to the establishment of a remote session. If an authorization error happens during the process of logging a user into a session with credentials obtained from CredSSP (for example, the user is not part of the "Remote Desktop Users" group), then a Set Error Info PDU (section 2.2.5.1) is sent to the client with the error code ERRINFO_SERVER_INSUFFICIENT_PRIVILEGES (0x00000009).

5.5 Automatic Reconnection

The automatic reconnection feature allows a client to reconnect to an existing session (after a short-term network failure has occurred) without having to resend the user's credentials to the server. A connection which employs automatic reconnection proceeds as follows:

1. The user logs in to a new or existing Terminal Server session. As soon as they have been authenticated, a Server Auto-Reconnect Packet (section 2.2.4.2) is generated by the server and sent to the client in the Save Session Info PDU (section 2.2.10.1). The Auto-Reconnect Packet (also called the auto-reconnect cookie) contains a 16-byte cryptographically secure random number (called the auto-reconnect random) and the ID of the session to which the user has connected.

2. The client receives the cookie and stores it in memory, never allowing programmatic access to it.

   In the case of a disconnection due to a network error:

3. The client attempts to reconnect to the server by trying to reconnect continuously or for a predetermined number of times. Once it has connected, the client and server may exchange large random numbers (the client and server random specified in section 5.3.4) —if Enhanced RDP Security is in effect, no client random is sent to the server (see section 5.3.2).

4. The client derives a 16-byte security verifier from the random number contained in the auto-reconnect cookie received in Step 2. This security verifier is wrapped in a Client Auto-Reconnect Packet (section 2.2.4.3) and sent to the server as part of the extended information (see section 2.2.1.11.1.1) of the Client Info PDU (section 2.2.1.11.1).

   The auto reconnect random is used to key the HMAC function (see [RFC2104]), which uses MD5 as the iterative hash function. The security verifier is derived by applying the HMAC to the client random received in Step 3:

```
SecurityVerifier = HMAC(AutoReconnectRandom, ClientRandom)
```

The one-way HMAC transformation prevents an unauthenticated server from obtaining the original auto-reconnect random and replaying it for the purpose of connecting to the user's disconnected session.

When Enhanced RDP Security is in effect the client random value is not generated (see section 5.3.2). In this case, for the purpose of generating the security verifier, the client random is assumed to be an array of sixteen zero bytes, that is, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }. This effectively means that the derived security verifier will always have the same value when carrying out auto-reconnect under the Enhanced RDP Security. Hence, care must be taken to authenticate the identity of the server to which the client is reconnecting, ensuring that the identity has not changed in the period between connections.

5. When the server receives the Client Auto-Reconnect Packet, it looks up the auto-reconnect random for the session and computes the security verifier using the client random (the same calculation executed by the client). If the security verifier value which the client transmitted matches the one computed by the server, the client is granted access. At this point, the server has confirmed that the client requesting auto-reconnection was the last one connected to the session in question.

6. If the check in Step 5 passes, then the client is automatically reconnected to the desired session; otherwise the client must obtain the user's credentials to regain access to the remote session.

The auto-reconnect cookie associated with a given session is flushed and regenerated whenever a client connects to the session or the session is reset. This ensures that if a different client connects to the session, then any previous clients which were connected can no longer use the auto-reconnect mechanism to connect. Furthermore, the server invalidates and updates the cookie at hourly intervals, sending the new cookie to the client in the Save Session Info PDU.

6 Appendix A: Windows Behavior

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

- Windows Vista
- Windows Server 2003
- Windows XP

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.5: By default, the server listens on port 3389. The client, by extension, attempts to connect on the same port.

<2> Section 3.3.5.9.5.1: RDP 5.0, 5.1 and 5.2 send the non-plain notification if the INFO_LOGONNOTIFY and INFO_AUTOLOGON flag was set by the client in the Client Info PDU or if the username or domain used to log on to the session is different from what was sent in the Client Info PDU.

7 Index