[MS-RDPRFX]:
Remote Desktop Protocol:
RemoteFX Codec Extension

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do 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. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/23/2010	0.1	Major	First Release.
06/04/2010	1.0	Major	Updated and revised the technical content.
07/16/2010	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	2.0	Major	Significantly changed the technical content.
10/08/2010	3.0	Major	Significantly changed the technical content.
11/19/2010	4.0	Major	Significantly changed the technical content.
01/07/2011	5.0	Major	Significantly changed the technical content.
02/11/2011	6.0	Major	Significantly changed the technical content.

Contents

1 Introduction

The Remote Desktop Protocol: RemoteFX Codec Extension is an extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]). The RemoteFX Codec Extension specifies a lossy image codec that can be used to encode screen images by using efficient and effective compression.

1.1 Glossary

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

little-endian

The following terms are specific to this document:

blit: Also known as block image transfer. An operation in which a rectangular block of pixels in a source image is copied onto a destination image.

discrete wavelet transform (DWT): A discrete wavelet transform is a mathematical procedure that can be used to derive a discrete representation of a signal.

entropy coding: A lossless data compression scheme used to generate compression codes for input symbols based on their statistical properties.

quantization: A technique used to reduce a range of values to a single representative value.

tile-based transform: A transform technique in which an input image is segmented into a grid of disjoint tiles and the transform is then applied separately to each individual tile.

YCbCr color space: A color space where each color is represented as a triple (Y,Cb,Cr), where Y stands for the Luma (brightness) component and Cb,Cr stand for the two Chroma (color) components.

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.

[ARLGR] Malvar, H.S., "Adaptive Run-Length / Golomb-Rice Encoding of Quantized Generalized Gaussian Sources with Unknown Statistics", Proceedings of the Data Compression Conference, 2006 (DCC 2006) pp. 23 - 32, March 2006, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01607237

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

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

1.2.2 Informative References

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

1.3 Protocol Overview (Synopsis)

The Remote Desktop Protocol: RemoteFX Codec Extension reduces the bandwidth associated with desktop remoting by efficiently compressing images. This is achieved by using the RemoteFX codec. The following sections provide an overview of this codec.

1.3.1 RemoteFX Codec

One of the core requirements of desktop remoting is the ability to efficiently compress server-side screen images so that they can be transported over a network and displayed on a client screen. Any codec used for this purpose needs to be able to deliver effective compression (to reduce network bandwidth requirements) and operate with low-latency (to enable efficient interactions with remoted content). A typical desktop screen contains textual content (synthetic images) along with video and photographic content (natural images). Given the sensitivity of the human eye to the sharp features present in textual content, any applied compression has to be visually lossless; otherwise the text will appear blurred.

The RemoteFX codec has been designed to achieve efficient compression, satisfying the goals of high quality and low latency while using a modest amount of computing resources. It is a **tile-based transform** codec. The transform chosen was a **discrete wavelet transform (DWT)** because it enables superior compression performance when compressing textual bitmap regions at high quality. The **entropy coding** is performed using the Run-Length Golomb-Rice Coder (RLGR) ([ARLGR] section 3), which yields compression gains at relatively low computing requirements. The core functional blocks of the RemoteFX codec are shown in the following diagram.

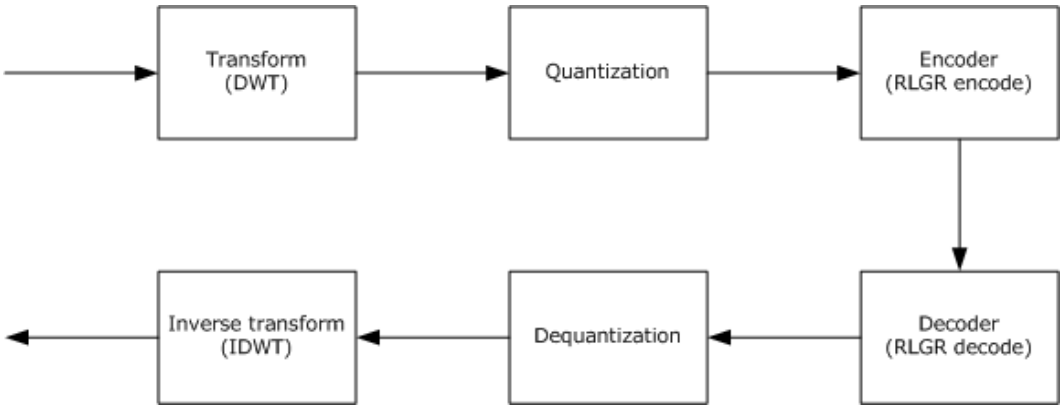


Figure 1: Core functional blocks of the RemoteFX codec

1.3.1.1 Message Flows

RemoteFX codec messages must be transported in order over a lossless transport such as TCP/IP. The message syntax has been designed with this prerequisite.

There are two types of messages: (1) capability messages sent from the client to the server; and (2) encode stream messages sent from the server to the client. The encode stream messages can

be broadly categorized as header or data messages. The syntax of each message is described in detail in section 2. Processing events and sequencing rules are described in section 3. The typical message sequence is depicted in the following diagram.

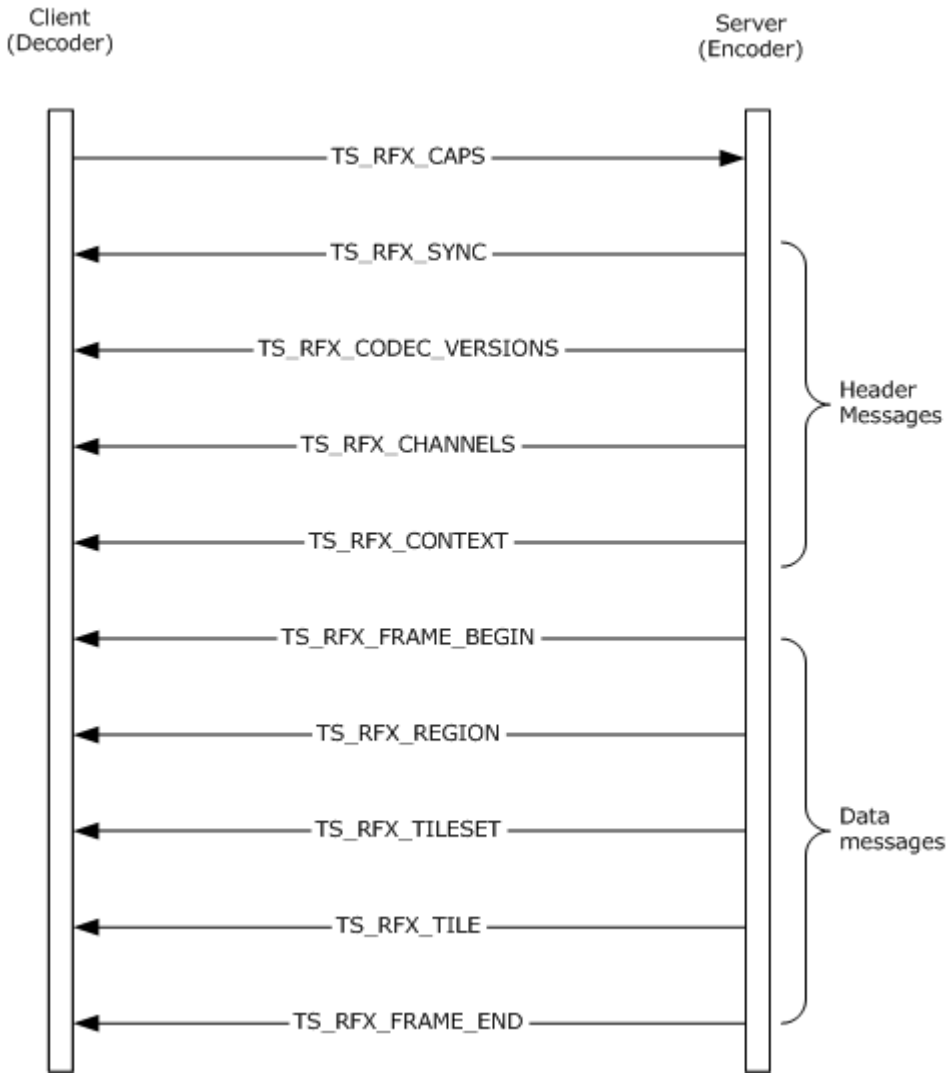


Figure 2: The RemoteFX message sequence

The client initiates the session by sending a Capabilities message. This is the only message sent from the client to the server; it lists the client-side support and preferences for various RemoteFX codec properties.

The server initializes its encoding state based on the client Capabilities message. It starts the encoding stream by sending a sequence of header messages that inform the client of the RemoteFX properties selected by the server:

1. The TS_RFX_SYNC message contains the RemoteFX magic number and the version of the wire format.

2. The TS_RFX_CODEC_VERSIONS message contains the version of the RemoteFX codec.

3. The TS_RFX_CHANNELS message lists the channel or multi-monitor information.

4. The TS_RFX_CONTEXT message contains the encoding properties of the stream.

The header messages are followed by the Data messages, which represent the sequence of encoded frames in the stream:

1. The TS_RFX_FRAME_BEGIN and TS_RFX_FRAME_END messages are used to demarcate an encoded frame.

2. The TS_RFX_REGION message contains the list of rectangles that have been encoded.

3. The TS_RFX_TILESET message encapsulates the list of tiles that have been encoded.

1.4 Relationship to Other Protocols

This protocol extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting [MS-RDPBCGR] by adding advanced compression techniques.

1.5 Prerequisites/Preconditions

1.6 Applicability Statement

This protocol is applicable in situations in which it is necessary to optimize the bandwidth required for graphics remoting. The advanced compression techniques specified in this document enable the efficient transfer of server-side images and video.

1.7 Versioning and Capability Negotiation

This protocol builds on the basic Remote Desktop Protocol [MS-RDPBCGR]. The features provided by this extension are negotiated during the capabilities negotiation phase of the RDP connection sequence (see [MS-RDPBCGR] section 1.3.1.1). In effect, this extension merely expands the set of capabilities used by the base RDP. (RDP versioning and capability negotiation is described in [MS-RDPBCGR] section 1.7.)

Some capabilities, which are specified as optional in [MS-RDPBCGR] section 2.2.7.2, are mandatory when used with RemoteFX. These capabilities are described in detail in section 2.1.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This protocol is an extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification and all packets are tunneled within the RDP transport ([MS-RDPBCGR] section 2.1). Aside from the following use of capabilities of the RDP transport, there are no additional transport-level requirements imposed by this extension.

The following three capabilities, which are specified as optional in [MS-RDPBCGR] section 2.2.7.2, are mandatory when used with RemoteFX.

- The client MUST send the Multifragment Update Capability Set ([MS-RDPBCGR], section 2.2.7.2.6). The **MaxRequestSize** field in the client-to-server Multifragment Update Capability Set MUST be set to a value greater than or equal to the value in the **MaxRequestSize** field of the server-to-client Multifragment Update Capability Set. The client-to-server Multifragment Update Capability Set is transported in the Confirm Active PDU as specified in [MS-RDPBCGR] section 2.2.1.13.2, and the server-to-client Multifragment Update Capability Set is transported in the Demand Active PDU as specified in [MS-RDPBCGR] section 2.2.1.13.1.
- The client MUST send the Large Pointer Capability Set ([MS-RDPBCGR] section 2.2.7.2.7), and the LARGE_POINTER_FLAG_96x96 (0x00000001) MUST be present in the **largePointerSupportFlags** field.
- If the Revision 2 Bitmap Cache Capability Set ([MS-RDPBCGR] section 2.2.7.1.4.2) is sent by the client, then the ALLOW_CACHE_WAITING_LIST_FLAG (0x0002) MUST be present in the **CacheFlags** field.

RemoteFX clients MUST set the **connectionType** field of the Client Core Data ([MS-RDPBCGR] section 2.2.1.3.2) to CONNECTION_TYPE_LAN (0x06). If the **connectionType** field is set to any other value, then the server SHOULD NOT include the TS_RFX_CLNT_CAPS_CONTAINER (section 2.2.1.1) in the Bitmap Codecs Capability Set ([MS-RDPBCGR] section 2.2.7.2.10).

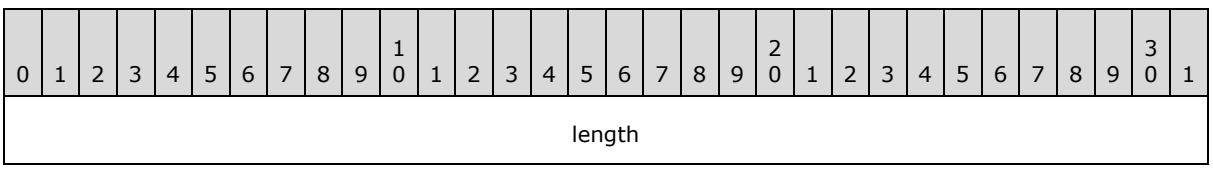
2.2 Message Syntax

All multiple-byte fields within a message MUST be marshaled in **little-endian** byte order, unless otherwise specified.

2.2.1 Capabilities Messages

2.2.1.1 TS_RFX_CLNT_CAPS_CONTAINER

The TS_RFX_CLNT_CAPS_CONTAINER structure is the top-level client capability container that wraps a TS_RFX_CAPS (section 2.2.1.1.1) structure and is sent from the client to the server. It is encapsulated in the **codecProperties** field of the Bitmap Codec ([MS-RDPBCGR] section 2.2.7.2.10.1.1) structure, which is ultimately encapsulated in the Bitmap Codecs Capability Set ([MS-RDPBCGR] section 2.2.7.2.10).



captureFlags
capsLength
capsData (variable)
...

length (4 bytes): A 32-bit, unsigned integer. Specifies the combined size, in bytes, of the **length**, **captureFlags**, **capsLength**, and **capsData** fields.

captureFlags (4 bytes): A 32-bit, unsigned integer. A collection of flags that allow a client to control how data is captured and transmitted by the server.

Flag	Meaning
CARDP_CAPS_CAPTURE_NON_CAC 0x00000001	The client supports mixing RemoteFX data with data compressed by other codecs. The set of other codecs supported by the client will be negotiated using the Bitmap Codecs Capability Set ([MS-RDPBCGR] section 2.2.7.2.10).

capsLength (4 bytes): A 32-bit, unsigned integer. Specifies the size, in bytes, of the **capsData** field.

capsData (variable): A variable-sized field that contains a TS_RFX_CAPS (section 2.2.1.1.1) structure.

2.2.1.1.1 TS_RFX_CAPS

The TS_RFX_CAPS structure contains information about the decoder capabilities.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
blockType																blockLen															
...																numCapsets															
capsetsData (variable)																															
...																															

blockType (2 bytes): A 16-bit, unsigned integer. Specifies the data block type. This field MUST be set to CBY_CAPS (0xCBC0).

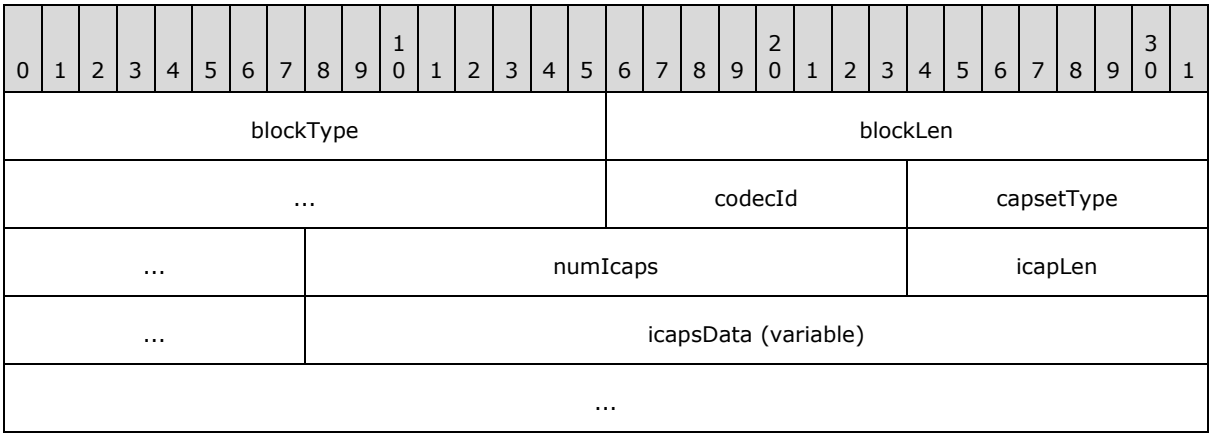
blockLen (4 bytes): A 32-bit, unsigned integer. Specifies the combined size, in bytes, of the **blockType**, **blockLen**, and **numCapsets** fields. This field MUST be set to 0x0008.

numCapsets (2 bytes): A 16-bit, unsigned integer. Specifies the number of TS_RFX_CAPSET (section 2.2.1.1.1.1) structures contained in the **capsetsData** field. This field MUST be set to 0x0001.

capsetsData (variable): A variable-sized array of TS_RFX_CAPSET (section 2.2.1.1.1.1) structures. The structures in this array MUST be packed on byte boundaries. The **blockType** and **blockLen** fields of each TS_RFX_CAPSET structure identify the type and size of the structure.

2.2.1.1.1.1 TS_RFX_CAPSET

The TS_RFX_CAPSET structure contains the capability information specific to the RemoteFX codec. It contains a variable number of TS_RFX_ICAP (section 2.2.1.1.1.1.1) structures that are used to configure the encoder state.



blockType (2 bytes): A 16-bit, unsigned integer. Specifies the data block type. This field MUST be set to CBY_CAPSET (0xCBC1).

blockLen (4 bytes): A 32-bit, unsigned integer. Specifies the combined size, in bytes, of the **blockType**, **blockLen**, **codecId**, **capsetType**, **numIcaps**, **icapLen**, and **icapsData** fields.

codecId (1 byte): An 8-bit, unsigned integer. Specifies the codec ID. This field MUST be set to 0x01.

capsetType (2 bytes): A 16-bit, unsigned integer. This field MUST be set to CLY_CAPSET (0xCFC0).

numIcaps (2 bytes): A 16-bit, unsigned integer. The number of TS_RFX_ICAP structures contained in the **icapsData** field.

icapLen (2 bytes): A 16-bit, unsigned integer. Specifies the size, in bytes, of each TS_RFX_ICAP structure contained in the **icapsData** field.

icapsData (variable): A variable-length array of TS_RFX_ICAP (section 2.2.1.1.1.1.1) structures. Each structure MUST be packed on byte boundaries. The size of each TS_RFX_ICAP structure within the array is specified in the **icapLen** field.

2.2.1.1.1.1.1 TS_RFX_ICAP

The TS_RFX_ICAP structure specifies the set of codec properties that the decoder supports.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
version																tileSize															
flags								colConvBits								transformBits								entropyBits							

version (2 bytes): A 16-bit, unsigned integer. Specifies the codec version. This field MUST be set to 0x0100 CLW_VERSION_1_0, to indicate protocol version 1.0.

tileSize (2 bytes): A 16-bit, signed integer. Specifies the width and height of a tile. This field MUST be set to CT_TILE_64x64 (0x0040), indicating that a tile is 64 x 64 pixels.

flags (1 byte): An 8-bit, unsigned integer. Specifies operational flags.

Flag	Meaning
CODEC_MODE 0x02	The codec will operate in image mode. If this flag is not set, the codec will operate in video mode.

When operating in image mode, the encode headers messages (section 2.2.2.2) MUST always precede an encoded frame. When operating in video mode, the header messages MUST be present at the beginning of the stream and are optional elsewhere.

colConvBits (1 byte): An 8-bit, unsigned integer. Specifies the color conversion transform. This field MUST be set to CLW_COL_CONV_ICT (0x1), and the transformation is by the equations in sections 3.1.8.1.3 and 3.1.8.2.5.

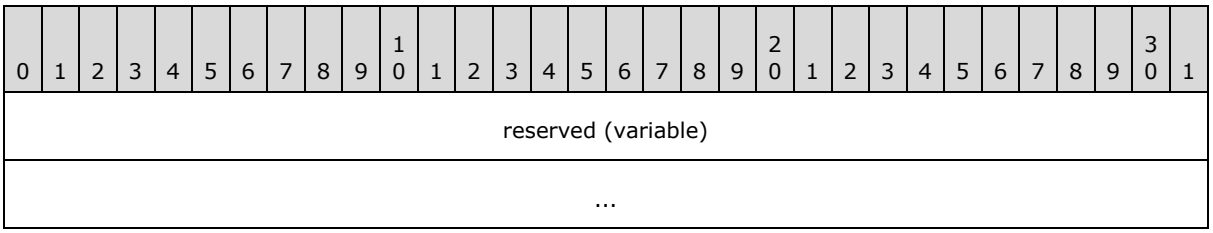
transformBits (1 byte): An 8-bit, unsigned integer. Specifies the DWT. This field MUST be set to CLW_XFORM_DWT_53_A (0x1), the DWT transform given by the lifting equations for the DWT shown in section 3.1.8.1.4 and by the lifting equations for the inverse DWT shown in section 3.1.8.2.4.

entropyBits (1 byte): An 8-bit, unsigned integer. Specifies the entropy algorithm. This field MUST be set to one of the following values.

Value	Meaning
CLW_ENTROPY_RLGR1 0x01	RLGR algorithm as described in 3.1.8.1.7.1.
CLW_ENTROPY_RLGR3 0x04	RLGR algorithm as described in section 3.1.8.1.7.2.

2.2.1.2 TS_RFX_SRVR_CAPS_CONTAINER

The TS_RFX_SRVR_CAPS_CONTAINER structure is the top-level server capability container, which is sent from the server to the client. It is encapsulated in the **codecProperties** field of the Bitmap Codec structure ([MS-RDPBCGR] section 2.2.7.2.10.1.1), which is ultimately encapsulated in the Bitmap Codecs Capability Set ([MS-RDPBCGR] section 2.2.7.2.10).



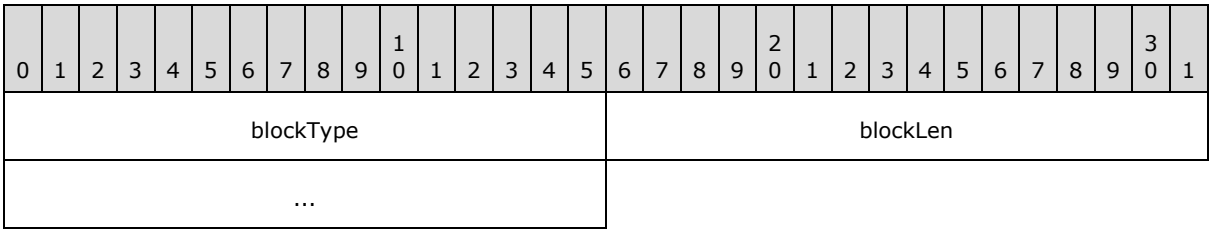
reserved (variable): A variable-sized array of bytes. All the bytes in this field MUST be set to 0. The size of the field is given by the corresponding **codecPropertiesLength** field of the parent TS_BITMAPCODEC, as specified in [MS-RDPBCGR] section 2.2.7.2.10.1.1 Bitmap Codecs Capability Set.

2.2.2 Encode Messages

2.2.2.1 Common Data Types

2.2.2.1.1 TS_RFX_BLOCKT

The TS_RFX_BLOCKT structure identifies the type of an encode message and specifies the size of the message.



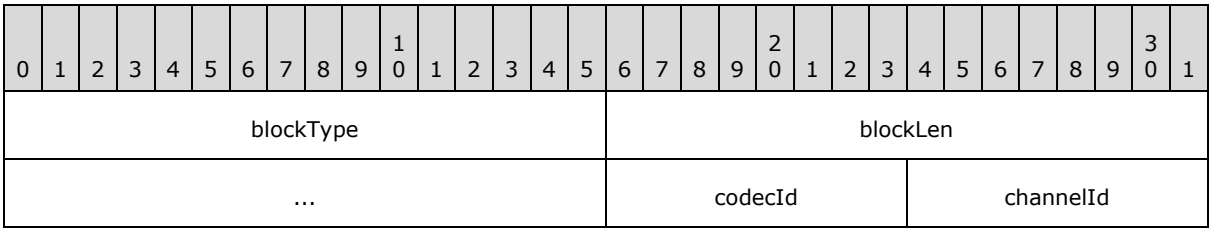
blockType (2 bytes): A 16-bit, unsigned integer. Specifies the data block type. This field MUST be set to one of the following values.

Value	Meaning
WBT_SYNC 0xCCC0	A TS_RFX_SYNC (section 2.2.2.2.1) structure.
WBT_CODEC_VERSIONS 0xCCC1	A TS_RFX_CODEC_VERSIONS (section 2.2.2.2.2) structure.
WBT_CHANNELS 0xCCC2	A TS_RFX_CHANNELS (section 2.2.2.2.3) structure.
CBT_TILE 0xCAC3	A TS_RFX_TILE (section 2.2.2.3.4.1) structure.

blockLen (4 bytes): A 32-bit, unsigned integer. Specifies the size, in bytes, of the data block. This size includes the size of the **blockType** and **blockLen** fields, as well as all trailing data.

2.2.2.1.2 TS_RFX_CODEC_CHANNELT

The TS_RFX_CODEC_CHANNELT structure is an extension of the TS_RFX_BLOCKT structure. It is present as the first field in messages that are targeted for a specific combination of codec and channel.



blockType (2 bytes): A 16-bit, unsigned integer. Specifies the data block type. This field MUST be set to one of the following values.

Value	Meaning
WBT_CONTEXT 0xCCC3	A TS_RFX_CONTEXT (section 2.2.2.2.4) structure.
WBT_FRAME_BEGIN 0xCCC4	A TS_RFX_FRAME_BEGIN (section 2.2.2.3.1) structure.
WBT_FRAME_END 0xCCC5	A TS_RFX_FRAME_END (section 2.2.2.3.2) structure.
WBT_REGION 0xCCC6	A TS_RFX_REGION (section 2.2.2.3.3) structure.
WBT_EXTENSION 0xCCC7	A TS_RFX_TILESET (section 2.2.2.3.4) structure.

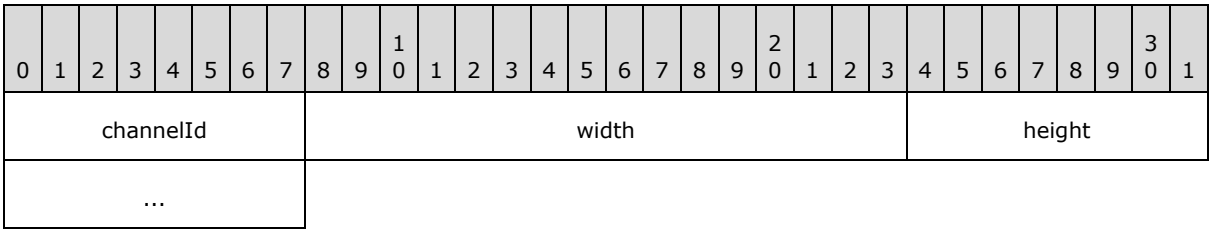
blockLen (4 bytes): A 32-bit, unsigned integer. Specifies the size, in bytes, of the data block. This size includes the size of the **blockType**, **blockLen**, **codecId**, and **channelId** fields, as well as all trailing data.

codecId (1 byte): An 8-bit, unsigned integer. Specifies the codec ID. This field MUST be set to 0x01.

channelId (1 byte): An 8-bit, unsigned integer. Specifies the channel ID. This field MUST be set to 0x00.

2.2.2.1.3 TS_RFX_CHANNELT

The TS_RFX_CHANNELT structure is used to specify the screen resolution of a channel.



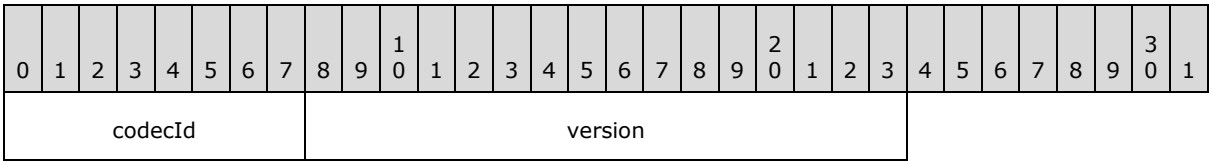
channelId (1 byte): An 8-bit, unsigned integer. Specifies the identifier of the channel. This field MUST be set to 0x00.

width (2 bytes): A 16-bit, signed integer. Specifies the frame width of the channel. This field MUST have value in the range of 1 to 4096.

height (2 bytes): A 16-bit, signed integer. Specifies the frame height of the channel. This field MUST have a value in the range of 1 to 2048.

2.2.2.1.4 TS_RFX_CODEC_VERSIONT

The TS_RFX_CODEC_VERSIONT structure is used to specify support for a specific version of the RemoteFX codec.

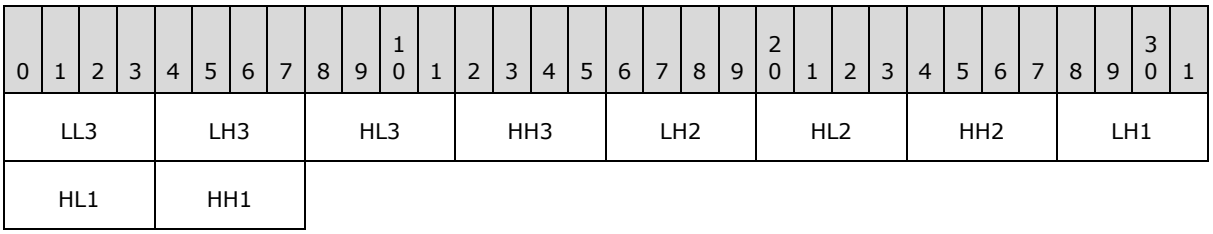


codecId (1 byte): An 8-bit, unsigned integer. Specifies the codec ID. This field MUST be set to 0x01.

version (2 bytes): A 16-bit, signed integer. This field MUST be set to 0x0100.

2.2.2.1.5 TS_RFX_CODEC_QUANT

The TS_RFX_CODEC_QUANT structure holds the scalar **quantization** values for the ten sub-bands in the 3-level DWT decomposition. Each field in this structure MUST have a value in the range of 6 to 15.



LL3 (4 bits): A 4-bit, unsigned integer. The LL quantization factor for the level-3 DWT sub-band.

LH3 (4 bits): A 4-bit, unsigned integer. The LH quantization factor for the level-3 DWT sub-band.

HL3 (4 bits): A 4-bit, unsigned integer. The HL quantization factors for the level-3 DWT sub-band.

HH3 (4 bits): A 4-bit, unsigned integer. The HH quantization factors for the level-3 DWT sub-band.

LH2 (4 bits): A 4-bit, unsigned integer. The LH quantization factor for the level-2 DWT sub-band.

HL2 (4 bits): A 4-bit, unsigned integer. The HL quantization factor for the level-2 DWT sub-band.

HH2 (4 bits): A 4-bit, unsigned integer. The HH quantization factor for the level-2 DWT sub-band.

LH1 (4 bits): A 4-bit, unsigned integer. The LH quantization factors for the level-1 DWT sub-band.

HL1 (4 bits): A 4-bit, unsigned integer. The HL quantization factors for the level-1 DWT sub-band.

HH1 (4 bits): A 4-bit, unsigned integer. The HH quantization factor for the level-1 DWT sub-band.

2.2.2.1.6 TS_RFX_RECT

The TS_RFX_RECT structure is used to specify a rectangle.

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
x																y															
width																height															

x (2 bytes): A 16-bit, unsigned integer. The X-coordinate of the rectangle.

y (2 bytes): A 16-bit, unsigned integer. The Y-coordinate 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.

2.2.2.2 Encode Header Messages

2.2.2.2.1 TS_RFX_SYNC

The TS_RFX_SYNC message MUST be the first message in any encoded stream. The decoder MUST examine this message to determine whether the protocol version is supported.

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
BlockT																															
...																magic															
...																version															

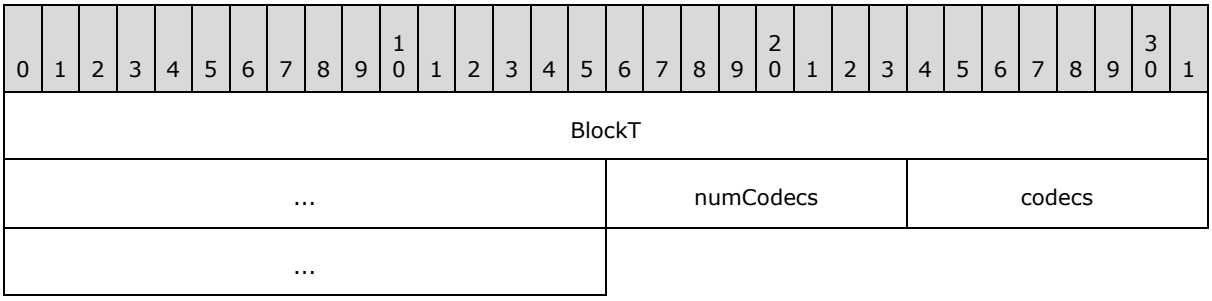
BlockT (6 bytes): A TS_RFX_BLOCKT (section 2.2.2.1.1) structure. The **blockType** field MUST be set to WBT_SYNC (0xCCC0).

magic (4 bytes): A 32-bit, unsigned integer. This field MUST be set to WF_MAGIC (0xCACCACCA).

version (2 bytes): A 16-bit, unsigned integer. Indicates the version number. This field MUST be set to WF_VERSION_1_0 (0x0100).

2.2.2.2.2 TS_RFX_CODEC_VERSIONS

The TS_RFX_CODEC_VERSIONS message indicates the version of the RemoteFX codec that is being used.



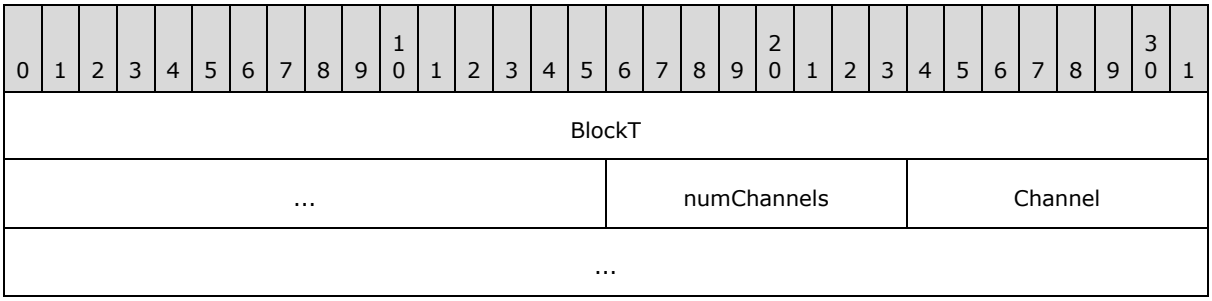
BlockT (6 bytes): A TS_RFX_BLOCKT (section 2.2.2.1.1) structure. The **blockType** field MUST be set to WBT_CODEC_VERSIONS (0xCCC1).

numCodecs (1 byte): An 8-bit, unsigned integer. Specifies the number of codec version data blocks in the **codecs** field. This field MUST be set to 0x01.

codecs (3 bytes): A TS_RFX_CODEC_VERSIONT (section 2.2.2.1.4) structure. The **codecId** field MUST be set to 0x01 and the **version** field MUST be set to WF_VERSION_1_0 (0x0100).

2.2.2.2.3 TS_RFX_CHANNELS

The TS_RFX_CHANNELS message contains the list of open channels, each of which corresponds to a monitor on the server. The decoder endpoint MUST be able to support channels with different frame dimensions.



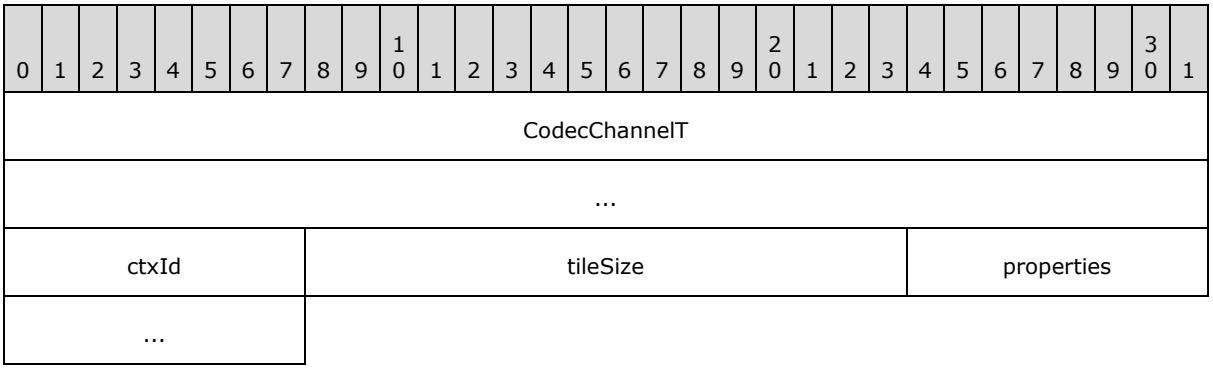
BlockT (6 bytes): A TS_RFX_BLOCKT (section 2.2.2.1.1) structure. The **blockType** field MUST be set to WBT_CHANNELS (0xCCC2).

numChannels (1 byte): An 8-bit, unsigned integer. Specifies the number of channel data blocks in the **channels** field. This field MUST be set to 0x01.

Channel (5 bytes): A TS_RFX_CHANNELT (section 2.2.2.1.3) structure. The **channelId** field MUST be set to 0x00.

2.2.2.2.4 TS_RFX_CONTEXT

The TS_RFX_CONTEXT message contains information regarding the encoding properties being used.

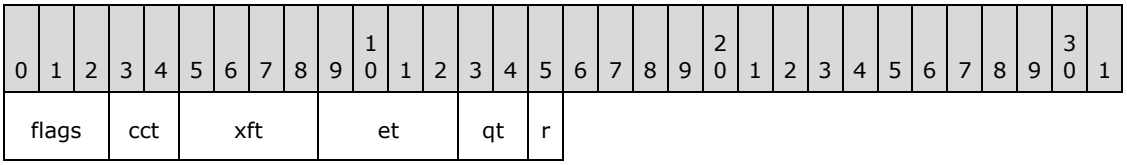


CodecChannelT (8 bytes): A TS_RFX_CODEC_CHANNELT structure. The **blockType** field MUST be set to WBT_CONTEXT (0xCCC3).

ctxId (1 byte): An 8-bit unsigned integer. Specifies an identifier for this context message. This field MUST be set to 0x00.

tileSize (2 bytes): A 16-bit unsigned integer. Specifies the tile size used by the RemoteFX codec. This field MUST be set to CT_TILE_64x64 (0x0040), indicating that a tile is 64 x 64 pixels.

properties (2 bytes): A 16-bit unsigned integer. Contains a collection of bit-packed property fields. The format of this field is described by the following bitfield diagram.



flags (3 bits): A 3-bit unsigned integer. Specifies operational flags.

Flag	Meaning
CODEC_MODE 0x02	The codec is operating in image mode. If this flag is not set, the codec is operating in video mode.

When operating in image mode, the Encode Headers messages (section 2.2.2.2) MUST always precede an encoded frame. When operating in video mode, the header messages MUST be present at the beginning of the stream and MAY be present elsewhere.

cct (2 bits): A 2-bit unsigned integer. Specifies the color conversion transform. This field MUST be set to COL_CONV_ICT (0x1) to specify the transform defined by the equations in sections 3.1.8.1.3 and 3.1.8.2.5.

xft (4 bits): A 4-bit unsigned integer. Specifies the DWT. This field MUST be set to CLW_XFORM_DWT_53_A (0x1), which indicates the DWT given by the equations in sections 3.1.8.1.4 and 3.1.8.2.4.

et (4 bits): A 4-bit unsigned integer. Specifies the entropy algorithm. This field MUST be set to one of the following values.

Value	Meaning
CLW_ENTROPY_RLGR1 0x01	RLGR algorithm as detailed in 3.1.8.1.7.1.
CLW_ENTROPY_RLGR3 0x04	RLGR algorithm as detailed in 3.1.8.1.7.2.

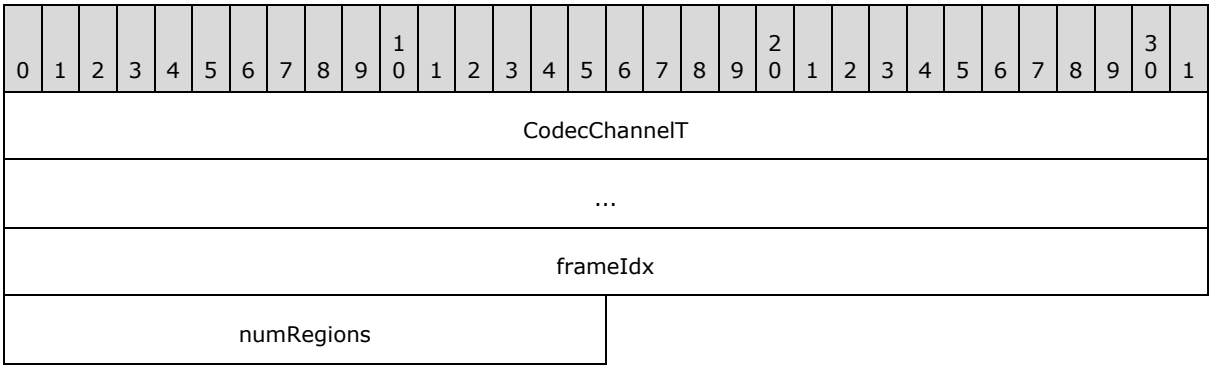
qt (2 bits): A 2-bit unsigned integer. Specifies the quantization type. This field MUST be set to SCALAR_QUANTIZATION (0x1).

r (1 bit): A 1-bit field reserved for future use. This field MUST be ignored when received.

2.2.2.3 Encode Data Messages

2.2.2.3.1 TS_RFX_FRAME_BEGIN

The TS_RFX_FRAME_BEGIN message indicates the start of a new frame for a specific channel in the encoded stream.



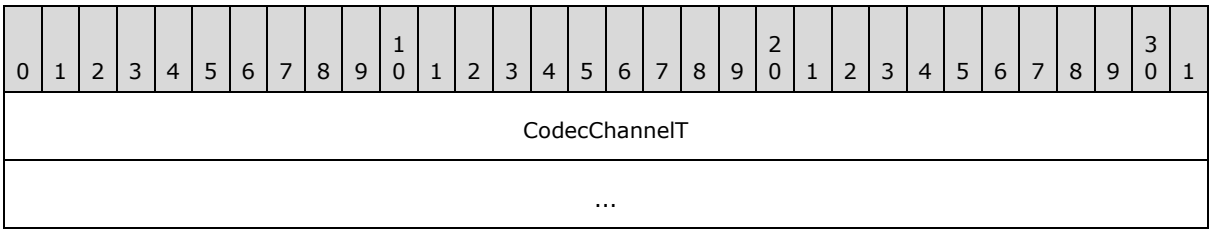
CodecChannelT (8 bytes): A TS_RFX_CODEC_CHANNELT (section 2.2.2.1.2) structure. The **blockType** field MUST be set to WBT_FRAME_BEGIN (0xCCC4).

frameIdx (4 bytes): A 32-bit unsigned integer. Specifies the index of the frame in the current video sequence. This field is used when the codec is operating in video mode, as specified using the **flags** field of the TS_RFX_CONTEXT (section 2.2.2.2.4) message. If the codec is operating in image mode, this field MUST be ignored.

numRegions (2 bytes): A 16-bit signed integer. Specifies the number of TS_RFX_REGION (section 2.2.2.3.3) messages following this TS_RFX_FRAME_BEGIN message. That is, the number of regions in the frame.

2.2.2.3.2 TS_RFX_FRAME_END

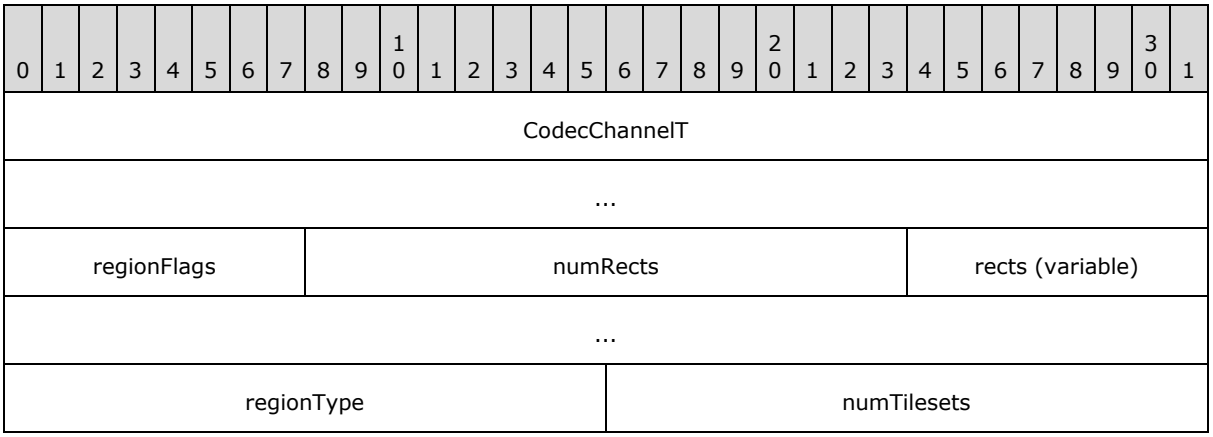
The TS_RFX_FRAME_END message specifies the end of a frame for a specific channel in the encoded stream.



CodecChannelT (8 bytes): A TS_RFX_CODEC_CHANNELT (section 2.2.2.1.2) structure. The **blockType** field MUST be set to WBT_FRAME_END (0xCCC5).

2.2.2.3.3 TS_RFX_REGION

The TS_RFX_REGION message contains information about the list of change rectangles on the screen for a specific channel. It also specifies the number of trailing TS_RFX_TILESET (section 2.2.2.3.4) messages.



CodecChannelT (8 bytes): A TS_RFX_CODEC_CHANNELT (section 2.2.2.1.2) structure. The **blockType** field MUST be set to WBT_REGION (0xCCC6).

regionFlags (1 byte): An 8-bit, unsigned integer. Contains a collection of bit-packed property fields. The format of this field is described by the following bitfield diagram.

0	1	2	3	4	5	6	7
lrf	reserved						

lrf (1 bit): A 1-bit unsigned integer. This field MUST be set to 0x1.

reserved (7 bits: A 7-bit integer reserved for future use. This field MUST be ignored.

numRects (2 bytes): A 16-bit, unsigned integer. Specifies the number of TS_RFX_RECT (section 2.2.2.1.6) structures present in the **rects** field.

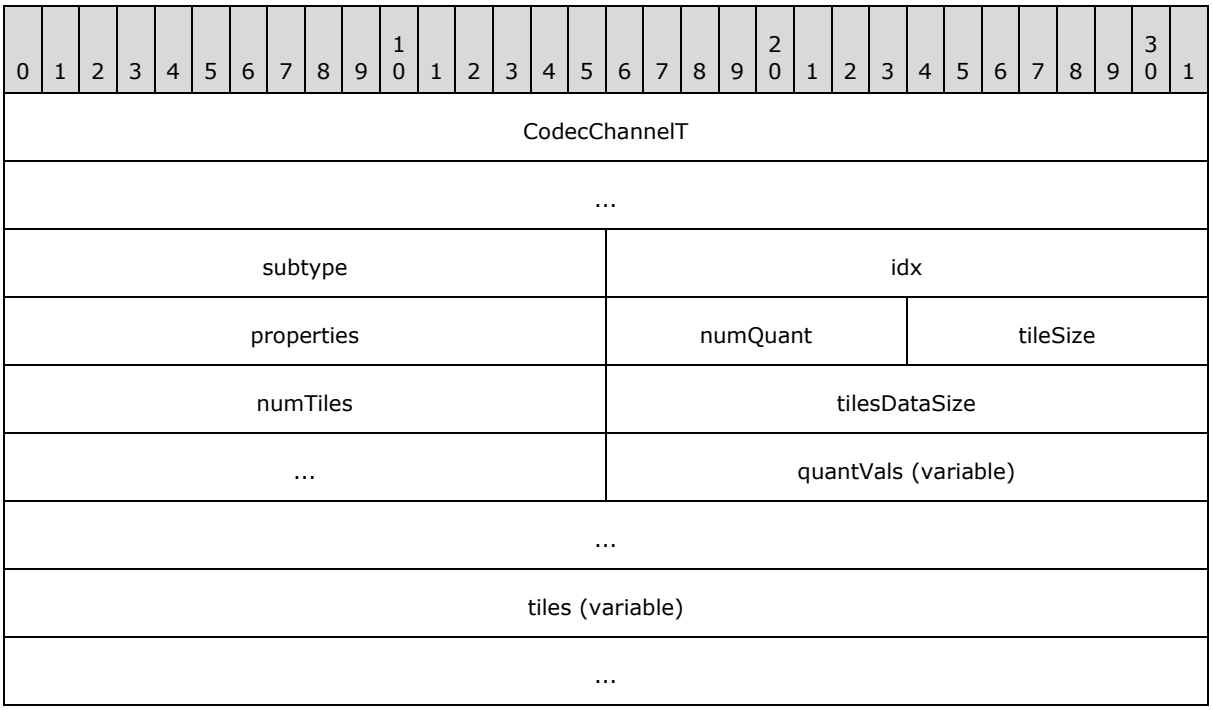
rects (variable): A variable-length array of TS_RFX_RECT (section 2.2.2.1.6)_structures. This array defines the region. The number of rectangles in the array is specified in the **numRects** field.

regionType (2 bytes): A 16-bit, unsigned integer. Specifies the region type. This field MUST be set to CBT_REGION (0xCAC1).

numTilesets (2 bytes): A 16-bit, unsigned integer. Specifies the number of TS_RFX_TILESET (section 2.2.2.3.4) messages following this TS_RFX_REGION message. This field MUST be set to 0x0001.

2.2.2.3.4 TS_RFX_TILESET

The TS_RFX_TILESET message contains encoding parameters and data for an arbitrary number of encoded tiles.

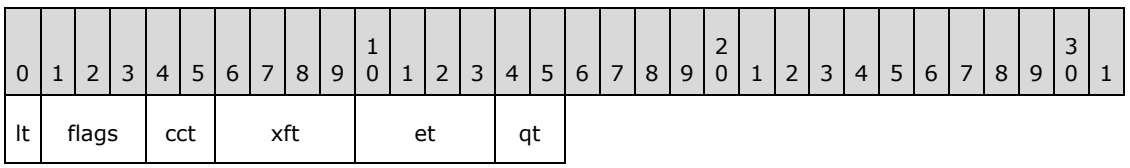


CodecChannelT (8 bytes): A TS_RFX_CODEC_CHANNELT (section 2.2.2.1.2) structure. The **blockType** field MUST be set to WBT_EXTENSION (0xCCC7).

subtype (2 bytes): A 16-bit, unsigned integer. Specifies the message type. This field MUST be set to CBT_TILESET (0xCAC2).

idx (2 bytes): A 16-bit, unsigned integer. Specifies the identifier of the TS_RFX_CONTEXT message referenced by this TileSet message. This field MUST be set to 0x0000.

properties (2 bytes): A 16-bit unsigned integer. Contains a collection of bit-packed property fields. The format of this field is described by the following bitmask diagram.



lt (1 bit): A 1-bit field that specifies whether this is the last TS_RFX_TILESET in the region. This field MUST be set to TRUE (0x1).

flags (3 bits): A 3-bit unsigned integer. Specifies operational flags.

Flag	Meaning
CODEC_MODE 0x02	The codec is operating in image mode. If this flag is not set, the codec is operating in video mode.

When operating in image mode, the Encode Headers messages (section 2.2.2.2) MUST always precede an encoded frame. When operating in Video Mode, the header messages MUST be present at the beginning of the stream and are optional elsewhere.

cct (2 bits): A 2-bit unsigned integer. Specifies the color conversion transform. This field MUST be set to COL_CONV_ICT (0x1) to specify the transform defined by the equations in sections 3.1.8.1.3 and 3.1.8.2.5.

xft (4 bits): A 4-bit unsigned integer. Specifies the DWT. This field MUST be set to CLW_XFORM_DWT_53_A (0x1), which indicates the DWT given by the equations in sections 3.1.8.1.4 and 3.1.8.2.4.

et (4 bits): A 4-bit unsigned integer. Specifies the entropy algorithm. This field MUST be set to one of the following values.

Value	Meaning
CLW_ENTROPY_RLGR1 0x01	RLGR algorithm as detailed in 3.1.8.1.7.1.
CLW_ENTROPY_RLGR3 0x04	RLGR algorithm as detailed in 3.1.8.1.7.2.

qt (2 bits): A 2-bit unsigned integer. Specifies the quantization type. This field MUST be set to SCALAR_QUANTIZATION (0x1).

numQuant (1 byte): An 8-bit, unsigned integer. Specifies the number of TS_RFX_CODEC_QUANT (section 2.2.2.1.5) structures present in the **quantVals** field.

tileSize (1 byte): An 8-bit, unsigned integer. Specifies the width and height of a tile. This field MUST be set to 0x40.

numTiles (2 bytes): A 16-bit, unsigned integer. Specifies the number of TS_RFX_TILE (section 2.2.2.3.4.1) structures present in the **tiles** field.

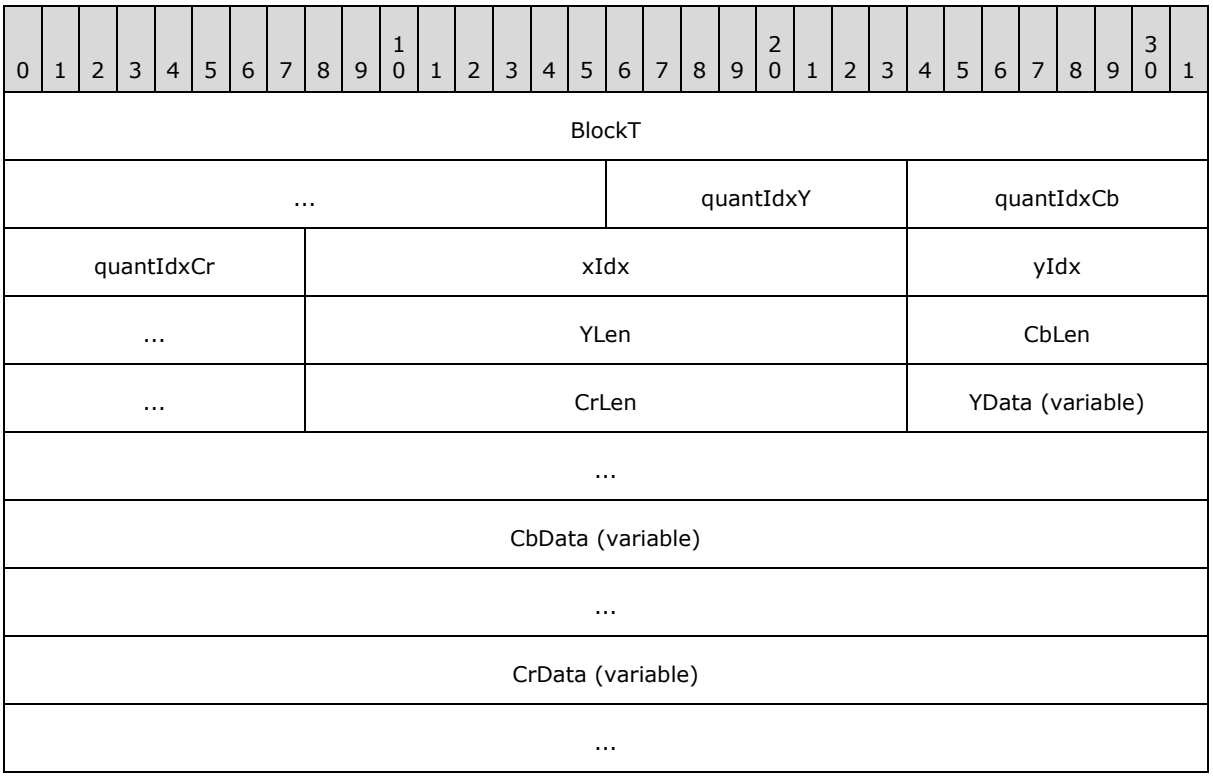
tilesDataSize (4 bytes): A 32-bit, unsigned integer. Specifies the size, in bytes, of the **tiles** field. The **tiles** field contains encoded data for all of the tiles that have changed.

quantVals (variable): A variable-length array of TS_RFX_CODEC_QUANT (section 2.2.2.1.5) structures. The number of structures present in the array is indicated in the **numQuant** field.

tiles (variable): A variable-length array of TS_RFX_TILE (section 2.2.2.3.4.1) structures. The number of structures present in the array is indicated in the **numTiles** field, while the total size, in bytes, of this field is specified by the **tilesDataSize** field.

2.2.2.3.4.1 TS_RFX_TILE

The TS_RFX_TILE structure specifies the position of the tile on the frame and contains the encoded data for the three tile components of Y, Cb, and Cr.



BlockT (6 bytes): A TS_RFX_BLOCKT (section 2.2.2.1.1) structure. The **blockType** field MUST be set to CBT_TILE (0xCAC3).

quantIdxY (1 byte): An 8-bit, unsigned integer. Specifies an index into the TS_RFX_CODEC_QUANT array provided in the TS_RFX_TILESET message. The specified **TS_RFX_CODEC_QUANT** element MUST be used for de-quantization of the sub-bands for the Y-component.

quantIdxCb (1 byte): An 8-bit, unsigned integer. Specifies an index into the TS_RFX_CODEC_QUANT array provided in the TS_RFX_TILESET message. The specified **TS_RFX_CODEC_QUANT** element MUST be used for de-quantization of the sub-bands for the Cb-component.

quantIdxCr (1 byte): An 8-bit, unsigned integer. Specifies an index into the TS_RFX_CODEC_QUANT array provided in the TS_RFX_TILESET message. The specified **TS_RFX_CODEC_QUANT** element MUST be used for de-quantization of the sub-bands for the Cr-component.

xIdx (2 bytes): A 16-bit, unsigned integer. The X-index of the encoded tile in the screen tile grid.

yIdx (2 bytes): A 16-bit, unsigned integer. The Y-index of the encoded tile in the screen tile grid.

YLen (2 bytes): A 16-bit, unsigned integer. Specifies the size, in bytes, of the **YData** field.

CbLen (2 bytes): A 16-bit, unsigned integer. Specifies the size, in bytes, of the **CbData** field.

CrLen (2 bytes): A 16-bit, unsigned integer. Specifies the size, in bytes, of the **CrData** field.

YData (variable): A variable-length array. Contains the encoded data for the Y-component of the tile. The size, in bytes, of this field is specified by the **YLen** field.

CbData (variable): A variable-length array. Contains the encoded data for the Cb-component of the tile. The size, in bytes, of this field is specified by the **CbLen** field.

CrData (variable): A variable-length array. Contains the encoded data for the Cr-component of the tile. The size, in bytes, of this field is specified by the **CrLen** field.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

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

OperationalMode: Stores the operational mode currently in use. Operational modes include video mode, and image mode.

EntropyAlgorithm: Stores the entropy algorithm currently in use. Entropy algorithms include RLGR1 and RLGR3.

FrameIndex: A 32-bit integer variable. Used by the server to keep track of the current index of the encoded frame within an encoding session. This variable is used only when the **OperationalMode** is video mode. If the **OperationalMode** is bitmap mode, the server does not need to maintain this variable.

In video mode, this variable should be initialized to 0 at the start of the session and then incremented by 1 after every encoded frame. The current value of this variable is stored in the **frameIdx** field of the TS_RFX_FRAME_BEGIN message (section 2.2.2.3.1).

3.1.1.1 State Machine

The following figure and table describe the state machine of the codec at the server end.

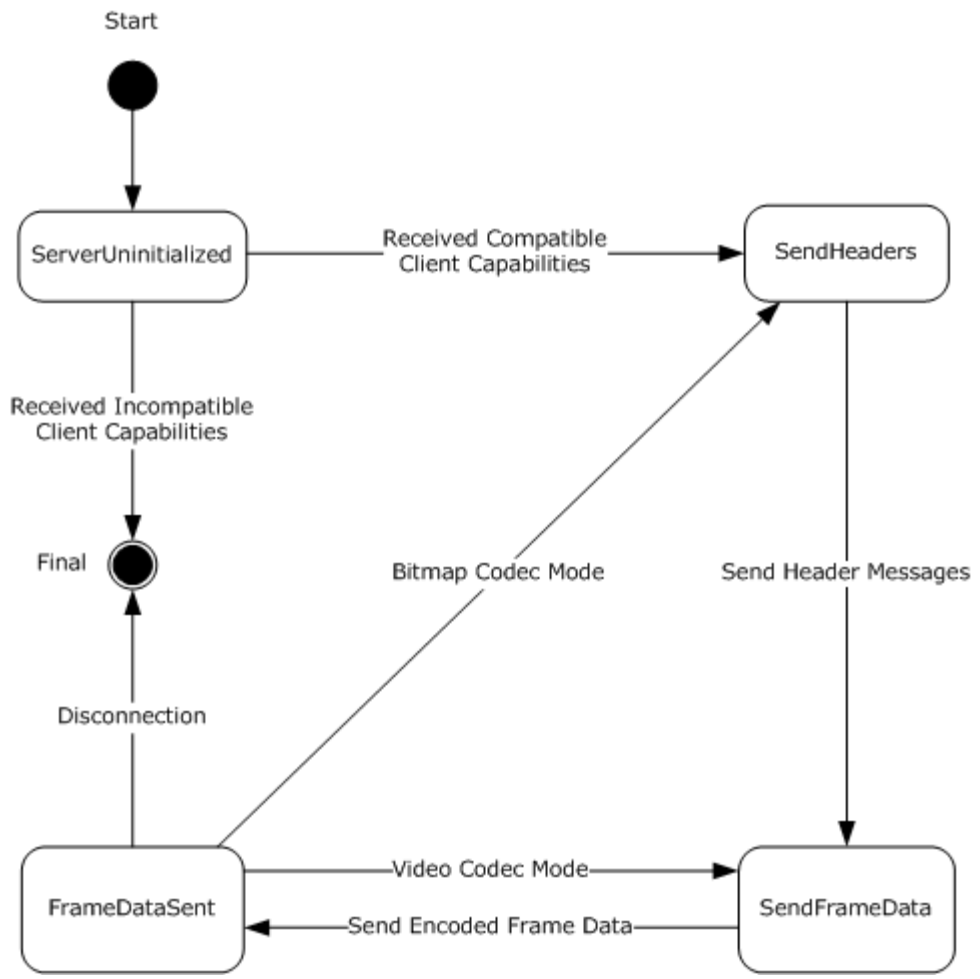


Figure 3: Server state diagram

State Name	Description
ServerUninitialized	This is the initial state of the server. In this state, the server waits for the TS_RFX_CLNT_CAPS_CONTAINER message from the client. On receiving this message, the server processes it as described in section 3.1.5.1. If it finds a compatible TS_RFX_ICAP, it initializes itself and gets into the SendHeaders state. Otherwise, the connection is terminated (section 3.1.5.1).
SendHeaders	In this state, the server sends the Header message sequence as described in section 3.1.8.3.1 and shown in Figure 17. The server then transitions to the SendFrameData state.
SendFrameData	In this state, the server sends the encoded frame data messages as described in section 3.1.8.3.1 and shown in Figure 18. The server then transitions to the FrameDataSent state.
FrameDataSent	If the **OperationalMode** of the server is bitmap mode, the server transitions to the SendHeaders state. If the **OperationalMode** is video mode, the server transitions to either the SendFrameData state or the SendHeaders state.

3.1.2 Timers

None.

3.1.3 Initialization

The Bitmap Codecs Capability Set message ([MS-RDPBCGR] section 2.2.7.2.10) MUST be processed by the server, as specified in section 3.1.5.1, before RemoteFX encoding begins. This establishes the encoding properties that will be used by the server when sending the encoded data stream.

The Bitmap Codecs Capability Set is sent by the client, encapsulating the TS_RFX_CLNT_CAPS_CONTAINER (section 2.2.1.1). The server ultimately processes the encapsulated TS_RFX_CLNT_CAPS_CONTAINER (section 2.2.1.1) message as specified in section 3.1.5.1, picking a TS_RFX_ICAP (section 2.2.1.1.1.1) element. From that point on, the server uses the capability properties listed in that element to encode RemoteFX data streams.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

Establishing the connection: RemoteFX capabilities messages are exchanged to establish the encoding properties used by the server, as specified in sections 3.1.3 and 3.1.5.1. A compliant server MUST process the TS_RFX_CLNT_CAPS_CONTAINER message as specified in section 3.1.5.1 before RemoteFX encoding can begin.

Sending RemoteFX encoded data: RemoteFX encoded data is sent to the client as a sequence of the RemoteFX messages defined in section 2.2.2. A compliant server MUST always send the encoded messages in the correct order, as specified in section 3.1.8.3.1.

Header messages: The encoded message sequence MUST include header messages, as specified in section 3.1.8.3.1.

Header messages contain information about the encoding properties used for encoding data messages, and are used by the client to decode the message stream. Header messages MUST always specify the encoding properties initialized and stored in **OperationalMode** and **EntropyAlgorithm**. The encode header and data message sequences are shown in the figures Generation of RemoteFX encode header messages and Generation of RemoteFX encode data messages (section 3.1.8.3.1).

Header messages can appear multiple times within the message stream, depending on the **OperationalMode** property:

- If the encoder is initialized with **OperationalMode** set to video mode, then it MUST send the encode header message sequence at the start of the stream. The encode header message sequence is then followed by an arbitrary number of encode data message sequences.
- If the encoder is initialized with **OperationalMode** set to image mode, then it MUST send the encode header message sequence before every encode data message sequence.

Entropy Algorithm: The server MUST use the entropy algorithm, initialized and stored in **EntropyAlgorithm**, to encode every data message in the encoded data stream.

Error conditions: If the client receives an out-of-sequence, unspecified, or malformed message, then the client MUST treat this as an error and terminate the RDP connection.

3.1.5.1 Processing the TS_RFX_CLNT_CAPS_CONTAINER Message

The structure and fields of the TS_RFX_CLNT_CAPS_CONTAINER, and its constituent members, are specified in section 2.2.1.1.

TS_RFX_CLNT_CAPS_CONTAINER has a TS_RFX_CAPS field. The TS_RFX_CAPS field contains a TS_RFX_CAPSET sub-field, which is composed of a variable number of TS_RFX_ICAP structures.

The encoder parses the TS_RFX_CLNT_CAPS_CONTAINER message to get to the array of TS_RFX_ICAP structures. It processes each element of this array to check whether it can support all the properties listed in that TS_RFX_ICAP element. From this set of supported elements, it will arbitrarily pick one element and use only the properties listed in that specific TS_RFX_ICAP to encode the data stream.

For example, if the decoder supports both RLGR1 and RLGR3, the client can specify support for both of them. This support is specified using two TS_RFX_ICAP elements in the TS_RFX_CAPSET message that the client sends to the server. If the encoder also supports both RLGR1 and RLGR3, it then arbitrarily picks one of the TS_RFX_CAPSET elements to use for encoding.

Once a TS_RFX_ICAP element has been picked, the **OperationalMode** and **EntropyAlgorithm** ADM elements are set as follows.

- If the TS_RFX_ICAP element's **flags** field is set to include the 0x02 flag, the **OperationalMode** is set to image mode; otherwise, the **OperationalMode** is set to video mode.
- When the TS_RFX_ICAP element's **entropyBits** field is set to 0x01, the **EntropyAlgorithm** is set to RLGR1; when **entropyBits** is set to 0x04, **EntropyAlgorithm** is set to RLGR3.

Error conditions: If the server cannot support any of the TS_RFX_ICAP elements, it MUST stop sending messages and consider the RDP connection terminated.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.1.8 RemoteFX Algorithm

RemoteFX is a tile-based transform codec. It has the same functional stages as those found in most structured compression systems (section 1.3.1). At the encode endpoint, these stages are transform, quantization, and entropy encoding. The inverse of these operations (in the reverse order) takes place at the decode endpoint. RemoteFX uses DWTs and Run-Length Golomb-Rice Coding (RLGR) ([ARLGR] section 3) for transformation and entropy encoding respectively.

3.1.8.1 Encoding

The functional stages involved in the encoding path are illustrated in the following figure. Each of these stages is described in the subsections that follow.

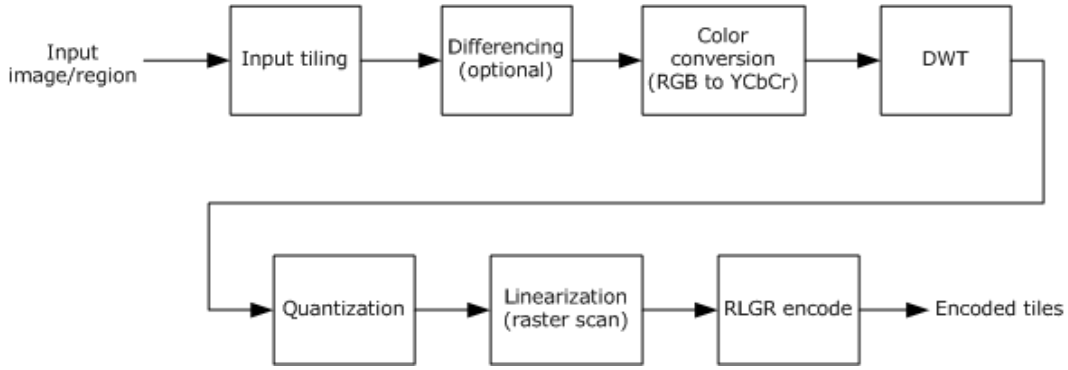


Figure 4: RemoteFX encoding stages

3.1.8.1.1 Input Tiling

The input to the encoder is an arbitrary region contained within an image to be encoded. The input image is overlaid with a tile grid anchored to the top left corner of the screen (0, 0) and aligned to the tile size (the tile size MUST be 64 x 64 pixels). The set of tiles that map to the input region are computed. These tiles are candidates for further processing, and each tile is processed independently of the others.

3.1.8.1.2 Differencing (Optional)

If the RemoteFX codec is to be used as a video codec, the encoder can, optionally, do differencing between current and previous frames to compute the smallest set of tiles that have changed. The differencing is done by comparing the set of input tiles in the current frame with collocated tiles from the previous frame to determine whether any pixels have changed. Only the input tiles with changed pixels in the current frame are processed for compression. Note that the input tile is compressed as is; this algorithm does not compress the tile of difference values formed by subtracting the input tile pixels of the current frame from the collocated tile pixels in the previous frame. This means that the decoder does not need to determine whether the encoder is doing differencing. The encoder can use differencing to reduce the number of tiles that it needs to encode, thereby reducing the bandwidth required to send the compressed tiles to the decoder. If the codec is to be used as an image codec, this stage MUST be skipped.

3.1.8.1.3 Color Conversion (RGB to YCbCr)

Each input tile is converted from the RGB color space to the **YCbCr color space**. The transform used takes an RGB input value with each component in the range [0-255] and transforms it into Y, Cb, and Cr, in the ranges [0.0, 255.0], [-128.0, 127.0], and [-128.0, 127.0], respectively. The Y-component is level-shifted down by 128, so that it also falls into the [-128.0, 127.0] range. The input tile in this level-shifted symmetric YCbCr color space is used as the input for the next stage of DWT. The following figure shows the matrix equation for this conversion.

$$\begin{bmatrix} Y & Cb & Cr \end{bmatrix} = \begin{bmatrix} R & G & B \end{bmatrix} \begin{bmatrix} 0.299 & -0.168935 & 0.499813 \\ 0.587 & -0.331665 & -0.418531 \\ 0.114 & 0.50059 & -0.081282 \end{bmatrix}$$

Figure 5: The RGB to YCbCr conversion matrix

3.1.8.1.4 DWT

Each tile component (Y, Cb, Cr) is individually transformed by a 2-D DWT using a 5/3 wavelet basis. Filtering at the boundary is done by mirroring the input coefficients. The filter coefficients used for both of the lifting-based implementations are presented in the following figure.

$$H[n] = \left\lfloor \frac{X[2n+1] - \left\lfloor \frac{X[2n] + X[2n+2]}{2} \right\rfloor}{2} \right\rfloor$$

a. High-pass coefficients

$$L[n] = X[2n] + \left\lfloor \frac{H[n-1] + H[n]}{2} \right\rfloor$$

b. Low-pass coefficients

Figure 6: Lifting equations for the DWT

For each level of decomposition, we first perform the DWT in the vertical direction, followed by the DWT in the horizontal direction. After the first level of decomposition, there are 4 sub-bands: LL1, LH1, HL1, HH1. For each successive level of decomposition, the LL sub-band of the previous level is used as the input. Each tile component undergoes three levels of decomposition. This results in 10 sub-bands per component. LH1, HL1, and HH1 contain the highest frequency bands present in the image tile, while LL3 contains the lowest frequency band.

The three-level DWT decomposition is illustrated in the following figure.

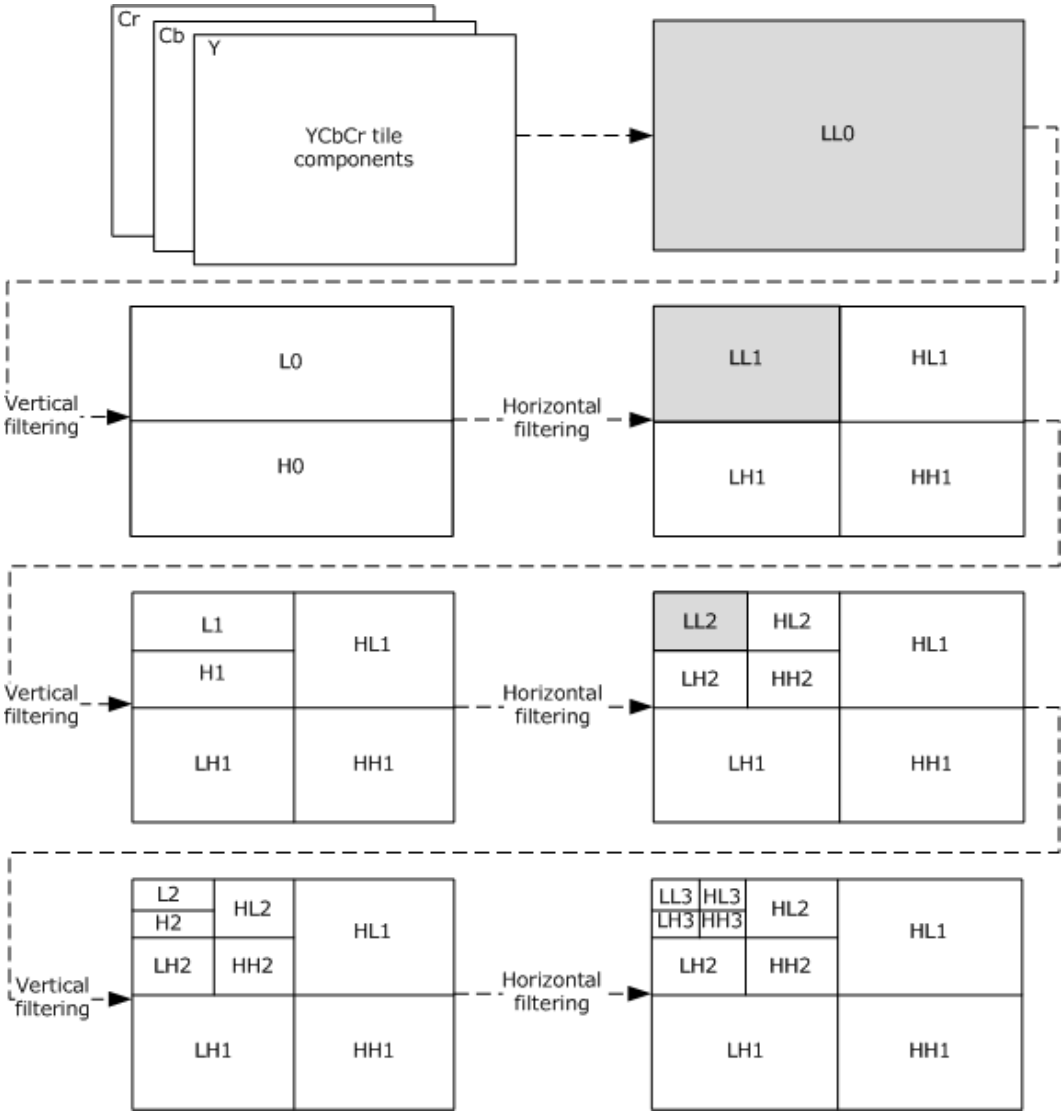


Figure 7: Three-level DWT decomposition

3.1.8.1.5 Quantization

The encoder determines a scale value for each sub-band and uses it to quantize all the coefficients in that sub-band, which is done by dividing each coefficient by the scale value and rounding it. These scale values are represented as quantization factors in a TS_RFX_CODEC_QUANT structure, which is embedded in a TS_RFX_TILESET message. The conversion between a scale value and a quantization factor is given by the following figure.

Scale _ value = (1<< (quantization _ factor — 6))

Figure 8: Quantization factor to scale value conversion

3.1.8.1.6 Linearization

The quantized tile components are linearized by raster scanning each of the sub-bands. The sub-band coefficient scan and traversal order is illustrated in the following figure.

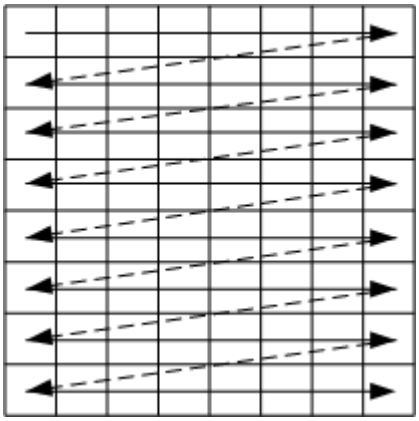


Figure 9: Raster scan of sub-band coefficients

Linearization of the sub-bands is conducted in the following sequence: HL1, LH1, HH1, HL2, LH2, HH2, HL3, LH3, HH3, and LL3, as shown in the following figure.

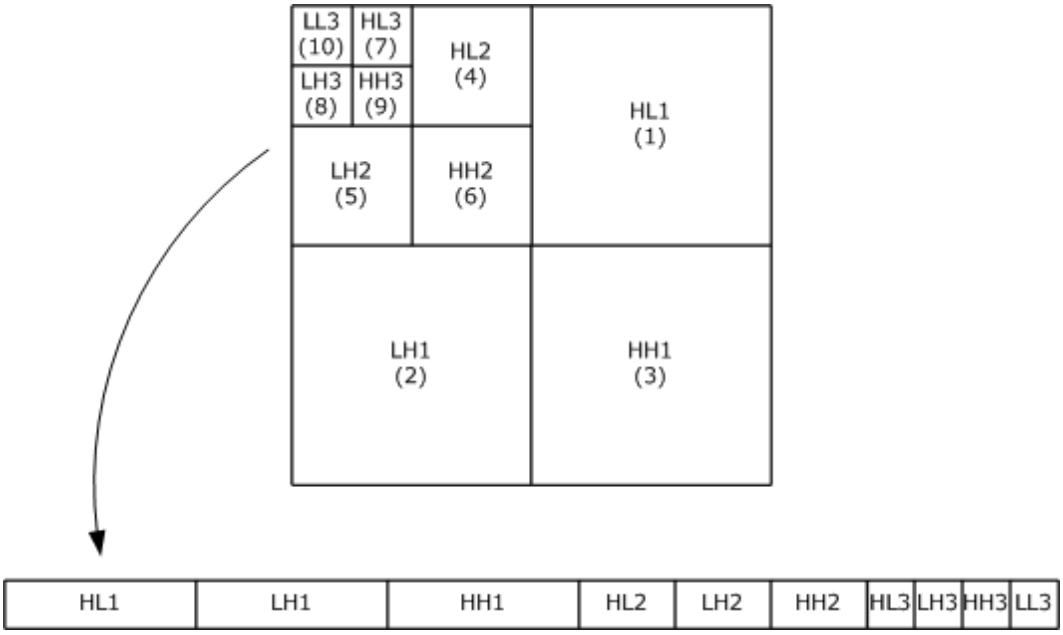


Figure 10: Sub-band traversal order

The coefficients from LL3 also undergo differential encoding. Except for the first coefficient, every raster-scanned LL3 coefficient is subtracted from its previous neighbor.

3.1.8.1.7 RLGR Entropy Encoding

Each 64 x 64-pixel tile contains an array of 4096 coefficients. This coefficient array is losslessly entropy-encoded using the RLGR algorithm ([ARLGR] section 3). The three tile components are assembled into an encoded tile packet (section 2.2.2.3.4.1). RLGR is an algorithm that adaptively switches between Run-Length encoding of zeroes and Golomb-Rice coding of nonzero coefficients. There are two variants of the RLGR algorithm that can be used, RLGR1 and RLGR3. The decoder endpoint specifies its preference through the capabilities negotiation (see the **entropyBits** field in section 2.2.1.1.1.1.1).

3.1.8.1.7.1 RLGR1

The RLGR algorithm is described in [ARLGR]. In this specification, the parameter adaptation rules, which are central to the algorithm, are detailed in section [ARLGR] section 3. The specific adaptation values used by RLGR1 are given in the following two tables.

$p = 0$	decrease k_R by setting $k_{RP} = k_{RP} - 2$
$p = 1$	no change in k_R
$p > 1$	increase k_R by setting $k_{RP} = k_{RP} + p$

Figure 11: Adaptation rule for RLGR1/RLGR3 parameter kR

$k = 0$	$u = 0 : k_P = k_P + 3$	
	$u > 0 : k_P = k_P - 3$	
$k > 0$	complete run	$k_P = k_P + 4$
	partial run	$k_P = k_P - 6$

Figure 12: Adaptation rule for RLGR1/RLGR3 main parameter k

The initial value for the parameters k_{RP} and k_P is 8, and the value of L is 8. Both the parameters k_{RP} and k_P are clipped to the range [0, 80] after every update.

3.1.8.1.7.2 RLGR3

The core RLGR algorithm [ARLGR] switches between the Run-Length and the Golomb-Rice mode as a stream of data is encoded, based on the Run-Length parameter, k, as shown in figure "Adaptation rule for RLGR1/RLGR3 main parameter k" in section 3.1.8.1.7.2. The Golomb-Rice mode in the RLGR algorithm [ARLGR], operates by encoding one coefficient at a time and updating the Golomb-Rice parameter, kR, as shown in figure "Adaptation rule for RLGR1/RLGR3 parameter kR" in section

3.1.8.1.7.2. In RLGR3, the Run-Length mode parameter, k, and the Golomb-Rice parameter, kR, are updated exactly as in RLGR1, but the Golomb-Rice mode is processed differently. Once the algorithm enters into the Golomb-Rice mode, RLGR3 takes the sum of the next two coefficients and Golomb-Rice encodes the sum and updates the parameters k and kR as with RLGR1. After the sum is encoded, the value of the first coefficient is emitted as binary code in the exact number of bits it takes to represent the sum. Pseudocode for Golomb-Rice mode in RLGR3 encode follows.

```
ENCODER GOLOMB-RICE MODE:

SUM = COEFF[n] + COEFF[n+1]  // sum of the next two coefficients in the input stream
GOLOMB_RICE_ENCODE(SUM)      // Golomb-Rice encode the value of SUM
BINARY_ENCODE(COEFF[n])      // using LOG2(SUM) bits
UPDATE_RLGR_PARAMETERS(SUM)  // update RLGR parameters (k,kR) based on SUM
```

The decoder follows the same sequence of steps as the encoder, as it decodes the encoded data. The pseudocode for the decoder in Golomb-Rice mode is given below.

```
DECODER GOLOMB-RICE MODE:

SUM = GOLOMB_RICE_DECODE()     // decode the next value from the encoded stream
NBS = LOG2(SUM)                // number of bits in binary representation of SUM
COEFF[n] = BINARY_DECODE(NBS)  // read next NBS bits from input stream as COEFF[n]
COEFF[n+1] = SUM - COEFF[n]    // compute COEFF[n+1]
UPDATE_RLGR_PARAMETERS(SUM)    // update RLGR parameters (k,kR) based on SUM
```

In general, RLGR3 encodes faster than RLGR1 but is marginally worse in terms of compression ratio.

3.1.8.2 Decoding

The functional stages involved in the decoding path are illustrated in the following figure. Compared to the encoding stages, the decoding stage operations are the operations of the encoding stage in reverse order.

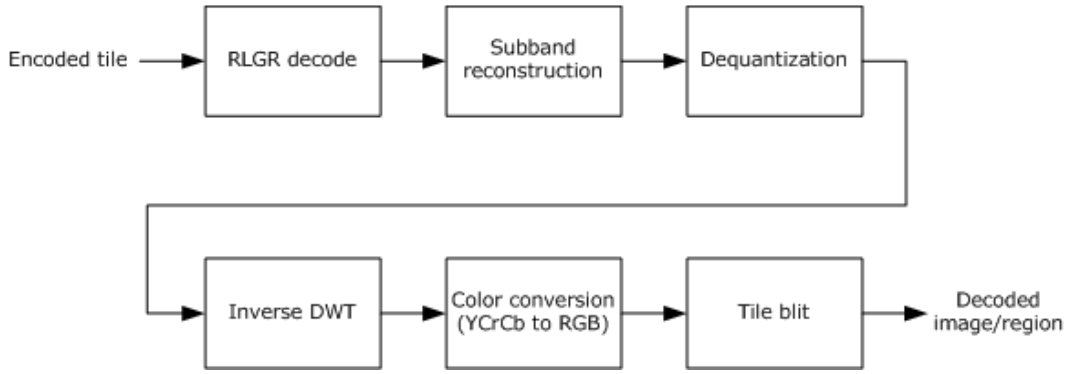


Figure 13: RemoteFX decoding stages

3.1.8.2.1 RLGR Entropy Decoding

The three encoded tile components (Y, Cb, and Cr) are entropy decoded independently. The algorithm used is either inverse RLGR1 or RLGR3, depending on which one was used for encoding. For details of the decoding process, refer to [ARLGR] (the encoding process is described in section 3.1.8.1.7).

3.1.8.2.2 Sub-Band Reconstruction

The RLGR entropy decoding stage results in the generation of an array of 4096 coefficients per tile component. The last 64 coefficients (which correspond to the LL3 sub-band) are differentially decoded. After this step, the coefficients are rearranged to form the sub-band structure shown in the figure in section 3.1.8.1.6 that describes the sub-band traversal order.

3.1.8.2.3 Dequantization

The quantization factors used for each sub-band are specified in the TS_RFX_TILE (section 2.2.2.3.4.1) structure. A scale value is computed from the quantization factor using the formula shown in the figure in section 3.1.8.1.5. Each coefficient in the appropriate sub-band is dequantized by multiplying it with this scale value and rounded.

3.1.8.2.4 Inverse DWT

Each tile component undergoes three levels of inverse discrete wavelet transformation (IDWT). The 5/3 lifting equations used for the IDWT are presented in the following figure.

$$X[2n] = L[n] - \left\lfloor \frac{H[n-1] + H[n] + 1}{2} \right\rfloor$$

a. Even coefficients

$$X[2n+1] = 2 * H[n] + \left\lfloor \frac{X[2n] + X[2n+2]}{2} \right\rfloor$$

b. Odd coefficients

Figure 14: Lifting equations for inverse DWT

This stage results in the decoded Y, Cb, and Cr components for the tile.

3.1.8.2.5 Color Conversion (YCbCr to RGB)

The Y-component is first level shifted up by 128 so that it falls in the [0.0, 255.0] range. The three YCbCr components are then transformed to the RGB color space by using the following transform matrix.

$$\begin{bmatrix} R & G & B \end{bmatrix} = \begin{bmatrix} Y & Cb & Cr \end{bmatrix} \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.0 & -0.344 & 1.77 \\ 1.403 & -1.77 & 0.0 \end{bmatrix}$$

Figure 15: The YCbCr to RGB conversion matrix

3.1.8.2.6 Tile Blit

Each decoded RGB tile is **blitted** to the appropriate location on the screen based on its tile grid index. This results in a completely decoded and reconstructed frame.

3.1.8.3 RemoteFX Stream

A RemoteFX stream is defined to be the set of all of the codec messages, sent sequentially from an encoder endpoint to a decoder endpoint. There is no message to signify the end of a stream. If the RemoteFX codec is to be used as an image codec, the stream only contains messages pertaining to a single image. If the RemoteFX codec is to be used as a video codec, the stream can contain an arbitrary number of images.

3.1.8.3.1 Message Sequence

An encoded RemoteFX stream is composed of a sequence of encode messages that are described in section 2.2.2. All encode messages start with a TS_RFX_BLOCKT (section 2.2.2.1.1) structure. When parsing the message blocks, the **blockLen** field of a TS_RFX_BLOCKT MUST be used to obtain the length of the data block. This length MUST NOT be less than the length based on the **blockType** field of the TS_RFX_BLOCKT.

The RemoteFX stream is structured as a set of header messages followed by encoded data messages. The header messages contain global information necessary to decompress the data messages. The header messages are described in section 2.2.2.2, and the data messages are described in section 2.2.2.3. The stream MUST start with the header messages and any of these headers can appear in the stream at a later stage. The header messages can be repeated.

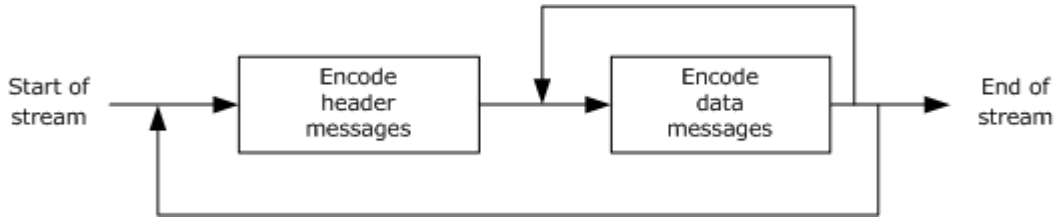


Figure 16: Message sequencing in the RemoteFX stream

The first message in a RemoteFX stream MUST be the TS_RFX_SYNC (section 2.2.2.2.1) message. This message MUST be followed by the TS_RFX_CHANNELS (section 2.2.2.2.3), TS_RFX_CODEC_VERSIONS (section 2.2.2.2.2), and TS_RFX_CONTEXT (section 2.2.2.2.4) messages, as shown in the figure that follows. It is permissible for these three messages to occur in any order. These three messages contain all of the information needed to initialize the decoder.

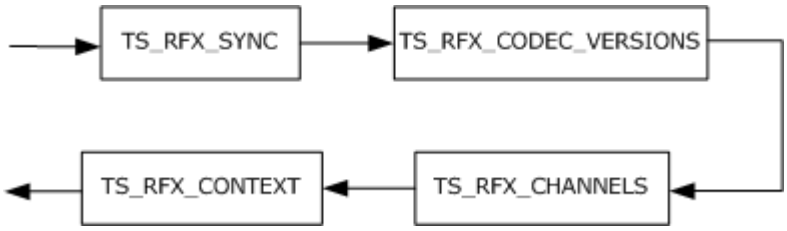


Figure 17: Generation of RemoteFX encode header messages

The TS_RFX_CHANNELS message MUST contain a single channel, and the frame dimensions of this channel are given by the **width** and **height** fields of the corresponding TS_RFX_CHANNELT structure (section 2.2.2.1.3). The decoder MUST check the TS_RFX_CODEC_VERSIONS and TS_RFX_CONTEXT messages to determine whether it is compatible with the RemoteFX codec version and the encoding properties listed in these messages. If the decoder cannot support the codec version, the channel frame dimensions, or any of the listed encoding properties, it MUST reject the encoded stream.

The data associated with each encoded frame or image is always bracketed by the TS_RFX_FRAME_BEGIN (section 2.2.2.3.1) and TS_RFX_FRAME_END (section 2.2.2.3.2) messages. The sequence of blocks that comprise a frame are described in the figure that follows. There MUST only be one TS_RFX_REGION (section 2.2.2.3.3) message per frame and one TS_RFX_TILESET (section 2.2.2.3.4) message per TS_RFX_REGION. All of the messages corresponding to a frame associated with a given channel MUST occur consecutively within the codec byte-stream. The messages corresponding to frames from two different channels MUST NOT be interleaved.

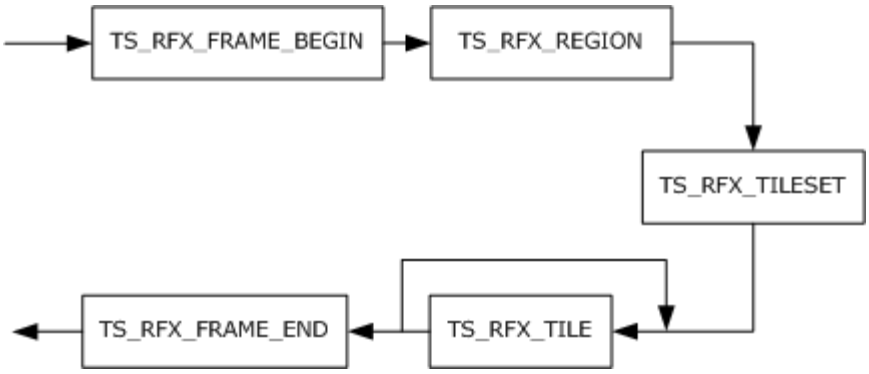


Figure 18: Generation of RemoteFX encode data messages

4 Protocol Examples

4.1 Sample Use Case

Consider the case of a remote endpoint system with one monitor configured to use a display resolution of 1280 x 1024 pixels. In this scenario, there is one instance of the RemoteFX encoder running on the remote system. The encoder is configured to use one channel with frame buffer dimensions of 1280 x 1024. As the contents of the screen are updated over time, the changes are captured, and the affected regions in the frame buffer corresponding to the bounding rectangles of the updated areas are fed as input to the encoder to compress.

The encoder examines the update regions and determines the set of tiles that correspond to those regions. The tile grid is anchored to the frame at (0, 0) and aligned to the tile size. This means that as an update region (for example, a window being dragged) moves around on the screen, the number of tiles corresponding to that update region can vary. In the figure that follows, the regions A and B are the same size (3 x 3) but they correspond to 9 and 16 tiles respectively due to their location on the screen. In the case of border tiles where the update region is not aligned to the tile grid, the area of the tile outside of the actual update region can contain arbitrary data and hence cannot be relied upon to contain valid image data. In the figure that follows, region B is not aligned to the tile grid and hence the perimeter tiles only contain a partial image.

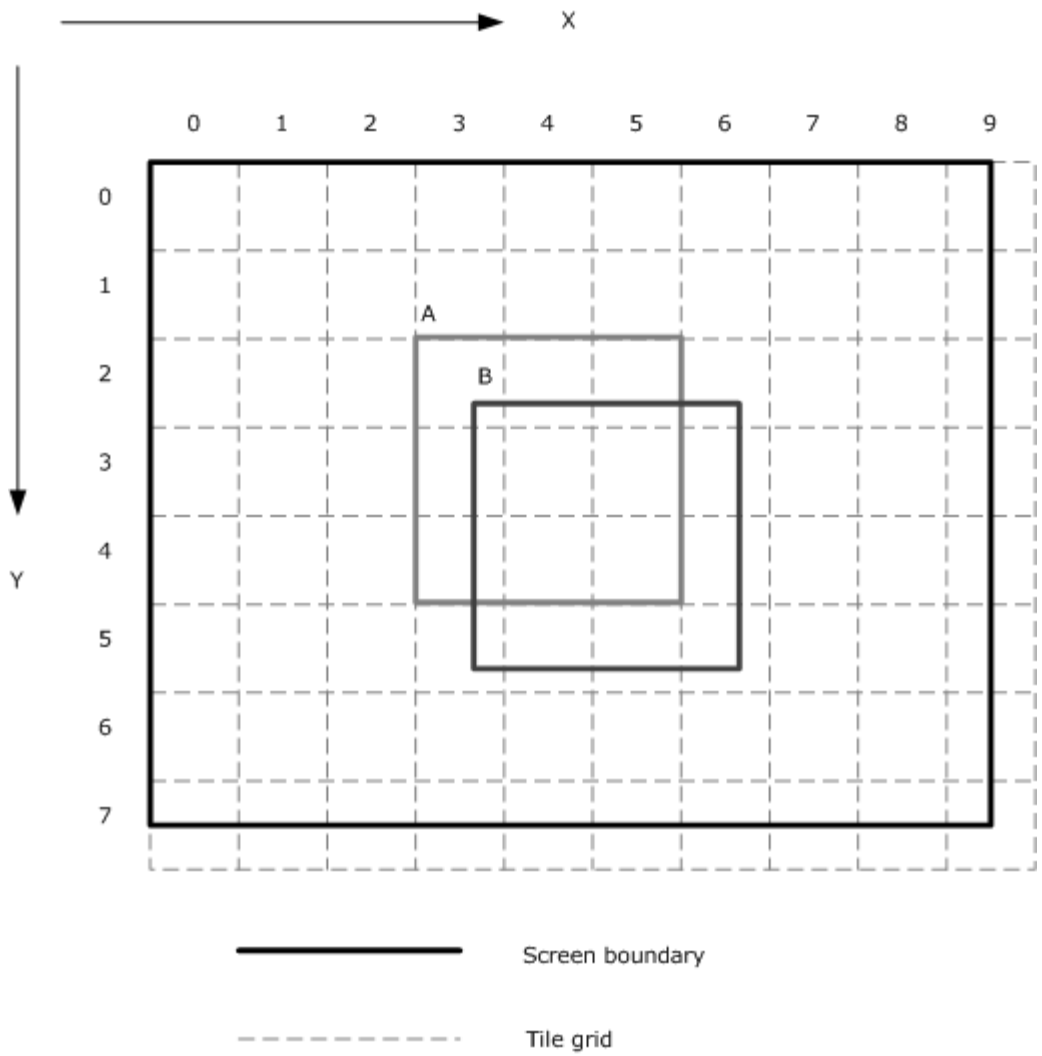


Figure 19: Update region to tile grid mapping

The single monitor configuration results in a TS_RFX_CHANNELS (section 2.2.2.2.3) header message that specifies one channel set to 1280 x 1024. For every frame that is encoded due to region updates, the message sequence that results is described in the encode messages diagram in section 3.1.8.3.1. The TS_RFX_REGION (section 2.2.2.3.3) message contains the list of updated rectangles, and the accompanying TS_RFX_TILESET (section 2.2.2.3.4) message contains the corresponding set of tiles. The TS_RFX_TILE (section 2.2.2.3.4.1) structure contains the location of the tile in the frame. Conceptually, the decoder can decode each tile, blit it to the proper location in a temporary frame buffer, and then blit all of the updated rectangles to an output frame buffer.

4.2 Annotated RemoteFX Messages

4.2.1 Capabilities Messages

The following is an annotated network capture of the TS_RFX_CLNT_CAPS_CONTAINER message (section 2.2.1.1).

```
00000000 31 00 00 00 01 00 00 00 25 00 00 00 c0 cb 08 00
00000010 00 00 01 00 c1 cb 1d 00 00 00 01 c0 cf 02 00 08
00000020 00 00 01 40 00 00 01 01 01 00 01 40 00 00 01 01
00000030 04
```

TS_RFX_CAPS message (section 2.2.1.1.1):

```
c0 cb -> TS_RFX_CAPS::blockType = CBY_CAPS
08 00 00 00 -> TS_RFX_CAPS::blockLen = 8
01 00 -> TS_RFX_CAPS::numCapsets = 1
```

TS_RFX_CAPSET message (section 2.2.1.1.1.1):

```
c1 cb -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::blockType = CBY_CAPSET
1d 00 00 00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::blockLen = 29
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::codecId = 1
C0 cf -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::capsetType = CLY_CAPSET
02 00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::numIcaps = 2
08 00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::icapLen = 8
```

TS_RFX_ICAP message (section 2.2.1.1.1.1.1):

```
00 01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::version = CLW_VERSION_1_0
40 00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::tileSize = 64
00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::flags = VIDEO_MODE (0)
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::colConvBits = CLW_COL_CONV_ICT
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::transformBits = CLW_XFORM_DWT_53_A
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[0]::entropyBits = CLW_ENTROPY_RLGR1
00 01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::version = CLW_VERSION_1_0
40 00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::tileSize = 64
00 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::flags = VIDEO_MODE (0)
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::colConvBits = CLW_COL_CONV_ICT
01 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::transformBits = CLW_XFORM_DWT_53_A
04 -> TS_RFX_CAPS::TS_RFX_CAPSET[0]::TS_RFX_ICAP[1]::entropyBits = CLW_ENTROPY_RLGR3
```

The client has specified support for both RLGR1 and RLGR3, by including two TS_RFX_ICAP elements. The server can pick either of the two and use the corresponding encoding properties, as described in section 3.1.5.1.

The following is an annotated dump of the TS_RFX_SRVR_CAPS_CONTAINER message (section 2.2.1.2).

```
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The server has sent an array of bytes set to zero as described in section 2.2.1.2.

4.2.2 Encode Header Messages

The following is an annotated network capture of the Encode Header Messages (section 2.2.2.2).

```
00000000  c0 cc 0c 00 00 00 ca ac cc ca 00 01 c3 cc 0d 00
00000010  00 00 01 ff 00 40 00 28 a8 c1 cc 0a 00 00 00 01
00000020  01 00 01 c2 cc 0c 00 00 00 01 00 40 00 40 00
```

TS_RFX_SYNC message (section 2.2.2.2.1):

```
c0 cc -> TS_RFX_SYNC::BlockT::blockType = WBT_SYNC
0c 00 00 00 -> TS_RFX_SYNC::BlockT::blockLen = 12
ca ac cc ca -> TS_RFX_SYNC::magic = WF_MAGIC
00 01 -> TS_RFX_SYNC::version = 0x0100
```

TS_RFX_CONTEXT message (section 2.2.2.2.4):

```
c3 cc -> TS_RFX_CONTEXT::CodecChannelT::BlockT::blockType = WBT_CONTEXT
0d 00 00 00 -> TS_RFX_CONTEXT::CodecChannelT::BlockT::blockLen = 13
01 -> TS_RFX_CONTEXT::CodecChannelT::codecId = 1
ff -> TS_RFX_CONTEXT::CodecChannelT::channelId = 255
00 -> TS_RFX_CONTEXT::ctxId = 0
40 00 -> TS_RFX_CONTEXT::tileSize = 64
28 a8 -> TS_RFX_CONTEXT::properties
TS_RFX_CONTEXT::properties::flags = VIDEO_MODE (0)
TS_RFX_CONTEXT::properties::cct = COL_CONV_ICT (1)
TS_RFX_CONTEXT::properties::xft = CLW_XFORM_DWT_53_A (1)
TS_RFX_CONTEXT::properties::et = CLW_ENTROPY_RLGR3 (4)
TS_RFX_CONTEXT::properties::qt = SCALAR_QUANTIZATION (1)
TS_RFX_CONTEXT::properties::r = RESERVED
```

TS_RFX_CODEC_VERSIONS message (section 2.2.2.2.2):

```
c1 cc -> TS_RFX_CODEC_VERSIONS::BlockT::blockType = WBT_CODEC_VERSION
0a 00 00 00 -> TS_RFX_CODEC_VERSIONS::BlockT::blockLen = 10
01 -> TS_RFX_CODEC_VERSIONS::numCodecs = 1
01 -> TS_RFX_CODEC_VERSIONS::TS_RFX_CODEC_VERSIONT::codecId = 1
00 01 -> TS_RFX_CODEC_VERSIONS::TS_RFX_CODEC_VERSIONT::version 0x0100
```

TS_RFX_CHANNELS message (section 2.2.2.2.3):

```
c2 cc -> TS_RFX_CHANNELS::BLockT::blockType = WBT_CHANNELS
0c 00 00 00 -> TS_RFX_CHANNELS::BlockT::blockLen = 12
01 -> TS_RFX_CHANNELS::numChannels = 1
00 -> TS_RFX_CHANNELS::TS_RFX_CHANNELT::channelId = 0
40 00 -> TS_RFX_CHANNELS::TS_RFX_CHANNELT::width = 64
40 00 -> TS_RFX_CHANNELS::TS_RFX_CHANNELT::height = 64
```

The server has chosen to encode using RLGR3 (TS_RFX_CONTEXT) and specified one channel with dimensions of 64x64 (TS_RFX_CHANNELS).

4.2.3 Encode Data Messages

The following is an annotated network capture of the Encode Data Messages (section 2.2.2.3).

```
00000000 c4 cc 0e 00 00 00 01 00 00 00 00 00 01 00 c6 cc
```

```
00000010 17 00 00 00 01 00 cd 01 00 00 00 00 00 40 00 40
00000020 00 c1 ca 01 00 c7 cc d9 03 00 00 01 00 c2 ca 00
00000030 00 51 50 01 40 01 00 be 03 00 00 66 66 77 88 98
00000040 c3 ca be 03 00 00 00 00 00 00 00 00 00 26 01 3d
00000050 01 48 01 19 82 1d 10 62 9d 28 85 2c a2 14 b2 88
00000060 52 ca 21 4b 28 85 2c a2 14 b2 88 52 ca 21 4b 28
00000070 85 2c a2 14 b2 88 52 ca 21 4b 28 85 2c a2 14 b2
00000080 88 52 ca 21 4b 28 85 2c a2 14 b2 88 52 ca 21 4b
00000090 28 85 2c a2 14 b2 88 52 ca 21 4b 28 85 2c a2 14
000000A0 b2 88 52 ca 21 4b 28 85 2c a2 14 b0 00 20 f4 40
000000B0 0c c1 1e 20 26 22 20 33 23 c4 23 88 86 50 f1 22
000000C0 68 4c 91 85 10 34 4c 84 78 a2 0d 13 21 1e 29 06
000000D0 89 90 8f 14 83 44 f4 23 c5 20 d1 3d 08 f1 48 34
000000E0 4f 42 3c 52 0d 13 d0 8f 14 83 44 f4 23 c5 20 d1
000000F0 3d 08 f1 48 34 4f 42 3c 52 0d 13 d0 8f 14 83 44
00000100 f4 23 c5 20 00 08 47 70 15 02 e0 7f e4 9d c2 51
00000110 71 f4 99 c9 57 ff 32 87 9d 17 d6 50 6e 06 2f ac
00000120 a0 9c 0c 5f 59 41 38 18 be b2 82 70 31 7d 65 00
00000130 00 10 ff 9c 33 41 f1 c4 b0 3c ff a2 15 bd 7b ea
00000140 86 9b 5f fc 78 8c f5 ed a8 68 da fd 3c 45 7a f4
00000150 d4 34 6d 7e 9e 22 bd 7a 6a 1a 36 bf 4f 11 5e bd
00000160 35 0d 1b 5f a7 88 af 5e 9a 86 8d af d3 c4 57 af
00000170 4d 43 46 d7 e9 e2 20 30 00 1b 04 7f 04 31 5f c2
00000180 94 af 05 29 5e 0a 52 bc 14 a5 78 29 25 78 29 25
00000190 78 29 25 68 52 4a f0 52 4a f0 52 4a d0 a4 95 e0
000001A0 a4 95 e0 a4 95 a1 49 2b c1 49 2b c1 49 2b 42 92
000001B0 57 82 92 57 82 92 56 85 24 af 05 24 af 05 24 ad
000001C0 0a 49 5e 0a 49 5e 0a 49 5a 14 92 bc 14 92 bc 14
000001D0 92 b4 29 25 78 29 25 78 00 02 0f 02 00 ac 13 fc
000001E0 c0 0a 20 10 2b 27 f9 80 b0 08 aa 3d 60 8c 0b 24
000001F0 ff 30 80 c0 aa 13 fc c2 03 05 90 9f e6 10 18 2c
00000200 84 ff 30 81 82 c8 4f f3 08 18 2c 84 ff 31 03 05
00000210 90 9f ff d8 40 60 59 09 fe 61 01 81 64 27 f9 84
00000220 06 0b 21 3f cc 20 30 59 09 fe 61 03 05 90 9f e6
00000230 10 30 59 09 fe 62 00 00 42 15 00 10 15 01 fe 20
00000240 84 d5 01 0a 8f f1 40 33 78 17 f9 c2 03 83 01 78
00000250 e1 01 c1 00 bc 70 80 e0 80 5e 38 40 70 40 2f 1c
00000260 20 38 20 17 8e 10 00 00 87 d5 08 70 ef 81 a2 d8
00000270 ff ff ff fb d1 2d 4e a6 ce 20 a4 ef 05 78 35 3a
00000280 9b 38 82 93 bc 15 e0 d4 ea 66 71 05 27 78 2b c1
00000290 29 d4 cc e2 0a 4e f0 57 82 53 a9 99 c4 14 9d e0
000002A0 af 04 a7 53 33 88 29 3b c1 5e 09 4e a6 67 10 52
000002B0 77 82 bc 00 18 00 1b fc 11 c1 0f 4a c1 4f 4a c1
000002C0 4f 4a a1 4d 95 42 9e 95 42 9e 95 42 9b 2a 85 3d
000002D0 2a 85 3d 2a 85 36 55 0a 7a 55 0a 7a 55 0a 6c aa
000002E0 14 f4 aa 14 f4 aa 14 d9 54 29 e9 54 29 e9 54 29
000002F0 b2 a8 53 d2 a8 53 d2 a8 53 65 50 a7 a5 50 a7 a5
00000300 50 a6 ca a1 4f 4a a1 4f 4a a1 4d 95 42 9e 95 42
00000310 9e 95 42 9b 2a 80 00 41 e3 80 3f e2 09 9c 00 22
00000320 07 03 e1 26 70 06 07 1f 04 67 00 61 df 02 67 00
00000330 0c 3b fe 01 33 80 06 1d ff 00 99 c0 03 0e ff 80
00000340 4c e0 01 87 7f c0 26 70 00 c3 bf e0 13 38 00 61
00000350 df f0 09 9c 00 30 ef f8 04 ce 00 18 77 fc 02 67
00000360 00 0c 3b fe 01 33 80 06 1d ff 00 99 c0 03 0e ff
00000370 80 4c e0 01 87 7f c0 26 70 00 00 08 3c 20 1f f1
00000380 00 f0 05 02 93 84 3d 20 f0 52 81 c7 ff ff ea 54
00000390 01 80 05 f5 4a 80 30 00 b6 a5 40 18 00 5f 54 a8
000003A0 03 00 0b ea 95 00 60 01 6d 4a 80 30 00 00 22 3f
000003B0 ba 08 10 2b 1f f2 20 3e 49 9c 1f 6e 0f 5a 0f fb
```

```
000003C0 18 46 ae 27 9b 83 cb 41 f3 18 46 ae 27 9b 83 c5
000003D0 a0 f9 8c 22 d7 13 8d c1 e2 d0 7c c6 11 6b 89 c6
000003E0 e0 f1 68 3e 63 08 b5 c4 e3 70 78 b4 1f 31 84 5a
000003F0 e2 71 b8 3c 5a 0f 98 c2 2d 71 30 83 c0 00 c5 cc
00000400 08 00 00 00 01 00
```

TS_RFX_FRAME_BEGIN message (section 2.2.2.3.1):

```
c4 cc -> TS_RFX_FRAME_BEGIN::CodecChannelT::blockType = WBT_FRAME_BEGIN
0e 00 00 00 -> TS_RFX_FRAME_BEGIN::CodecChannelT::blockLen = 14
01 -> TS_RFX_FRAME_BEGIN::CodecChannelT::codecId = 1
00 -> TS_RFX_FRAME_BEGIN::CodecChannelT::channelId = 0
00 00 00 00 -> TS_RFX_FRAME_BEGIN::frameIdx = 0
01 00 -> TS_RFX_FRAME_BEGIN::numRegions  = 1
```

TS_RFX_REGION message (section 2.2.2.3.3):

```
c6 cc -> TS_RFX_REGION::CodecChannelT::blockType = WBT_REGION
17 00 00 00 -> TS_RFX_REGION::CodecChannelT::blockLen = 23
01 -> TS_RFX_REGION::CodecChannelT::codecId = 1
00 -> TS_RFX_REGION::CodecChannelT::channelId = 0
0d -> TS_RFX_REGION::regionFlags
TS_RFX_REGION::regionFlags::lrf = 1
01 00 -> TS_RFX_REGION::numRects = 1
00 00 -> TS_RFX_REGION::TS_RFX_RECT::x = 0
00 00 -> TS_RFX_REGION::TS_RFX_RECT::y = 0
40 00 -> TS_RFX_REGION::TS_RFX_RECT::width = 64
40 00 -> TS_RFX_REGION::TS_RFX_RECT::height = 64
c1 ca -> TS_RFX_REGION::regionType = CBT_REGION
01 00 -> TS_RFX_REGION::numTilesets = 1
```

TS_RFX_TILESET message (section 2.2.2.3.4):

```
c7 cc -> TS_RFX_TILESET::CodecChannelT::blockType = WBT_EXTENSION
d9 03 00 00 -> TS_RFX_TILESET::CodecChannelT::blockLen = 985
01 -> TS_RFX_TILESET::codecId = 1
00 -> TS_RFX_TILESET::channelId = 0
c2 ca -> TS_RFX_TILESET::subtype = CBT_TILESET
00 00 -> TS_RFX_TILESET::idx = 0x00
51 50 -> TS_RFX_TILESET::properties
TS_RFX_TILESET::properties::lt = TRUE (1)
TS_RFX_TILESET::properties::flags =  VIDEO_MODE (0)
TS_RFX_TILESET::properties::cct = COL_CONV_ICT (1)
TS_RFX_TILESET::properties::xft = CLW_XFORM_DWT_53_A (1)
TS_RFX_TILESET::properties::et = CLW_ENTROPY_RLGR3 (4)
TS_RFX_TILESET::properties::qt = SCALAR_QUANTIZATION (1)
01 -> TS_RFX_TILESET::numQuant = 1
40 -> TS_RFX_TILESET::tileSize = 64
01 00 -> TS_RFX_TILESET::numTiles = 1
df 03 00 00 -> TS_RFX_TILESET::tilesDataSize = 991
66 66 77 88 98 -> TS_RFX_TILESET::quantVals
TS_RFX_TILESET::quantVals::LL3 = 6
TS_RFX_TILESET::quantVals::LH3 = 6
TS_RFX_TILESET::quantVals::HL3 = 6
TS_RFX_TILESET::quantVals::HH3 = 6
```

```
TS_RFX_TILESET::quantVals::LH2 = 7
TS_RFX_TILESET::quantVals::HL2 = 7
TS_RFX_TILESET::quantVals::HH2 = 8
TS_RFX_TILESET::quantVals::LH1 = 8
TS_RFX_TILESET::quantVals::HL1 = 8
TS_RFX_TILESET::quantVals::HH1 = 9
```

TS_RFX_TILE message (section 2.2.2.3.4.1):

```
c3 ca -> TS_RFX_TILE::BlockT::blockType = CBT_TILE
be 03 -> TS_RFX_TILE::BlockT::blockLen = 958
00 -> TS_RFX_TILE::quantIdxY = 0
00 -> TS_RFX_TILE::quantIdxCb = 0
00 -> TS_RFX_TILE::quantIdxCr = 0
00 00 -> TS_RFX_TILE::xIdx = 0
00 00 -> TS_RFX_TILE::yIdx = 0
26 01 -> TS_RFX_TILE::YLen = 294
3d 01 -> TS_RFX_TILE::CbLen = 317
48 01 -> TS_RFX_TILE::CrLen = 328
00000053:00000178 -> TS_RFX_TILE::YData
00000179:000002b5 -> TS_RFX_TILE::CbData
000002b6:000003fd -> TS_RFX_TILE::CrData
```

TS_RFX_FRAME_END message (section 2.2.2.3.2):

```
c5 cc -> TS_RFX_FRAME_END::CodecChannelT::blockType = WBT_FRAME_END
08 00 00 00 -> TS_FRAME_END::CodecChannelT::blockLen = 14
01 -> TS_FRAME_END::CodecChannelT::codecId = 1
00 -> TS_FRAME_END::CodecChannelT::channelId = 0
```

The server has sent a frame that is delineated by TS_RFX_FRAME_BEGIN and TS_RFX_FRAME_END messages. The frame contains a single region, which has a single tileset. This tileset, in turn, consists of one 64x64 tile. The compressed sizes, in bytes, for the Y,Cb,Cr components of this tile are (294, 317, 328).

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

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

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

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

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

7 Change Tracking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.1 Transport	59726 Specified that RemoteFX clients MUST set the connectionType field of the Client Core Data to CONNECTION_TYPE_LAN.	Y	Content updated.
4.2 Annotated RemoteFX Messages	57203 Added new section.	Y	New content added.
4.2.1 Capabilities Messages	57203 Added new section.	Y	New content added.
4.2.2 Encode Header Messages	57203 Added new section.	Y	New content added.
4.2.3 Encode Data Messages	57203 Added new section.	Y	New content added.

8 Index

U

V