RapidFire Programming Guide

Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information.

Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, ATI Radeon™, AMD FirePro™ and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Windows®, Visual Studio® and DirectX® are registered trademarks of Microsoft Corp.

OpenGL® is a registered trademarks of Silicon Graphics, Inc.

OpenCL™ is a registered trademark of Apple Inc. used by permission by Khronos.

Copyright Notice

© 2014-2016 Advanced Micro Devices, Inc. All rights reserved

Notice Regarding Standards. AMD does not provide a license or sublicense to any Intellectual Property Rights relating to any standards, including but not limited to any audio and/or video codec technologies such as MPEG-2, MPEG-4; AVC/H.264; HEVC/H.265; AAC decode/FFMPEG; AAC encode/FFMPEG; VC-1; and MP3 (collectively, the “Media Technologies”). For clarity, you will pay any royalties due for such third party technologies, which may include the Media Technologies that are owed as a result of AMD providing the Software to you.

MIT license

Copyright (c) 2017 Advanced Micro Devices, Inc. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

Preface

About This Document

This document is a programming guide for the RapidFire SDK. The major goal of this document is to provide an introduction to the implementation of virtual desktop and cloud gaming applications using the RapidFire SDK.

The RapidFire SDK provides an interface for virtual desktop and cloud gaming applications to reduce the encoding latency by utilizing AMD Radeon Pro or FirePro GPUs. OpenGL 4.2, DirectX 9 and DirectX 11 textures as well as desktop capturing can be used as input to generate H.264, H.265 or uncompressed stream output. While the encoding is handled entirely by RapidFire, the developer keeps full control over the configuration of the stream with the ability to adjust key encoding parameters such as resolution, frame rate and bit rate control throughout the streaming process.

Audience

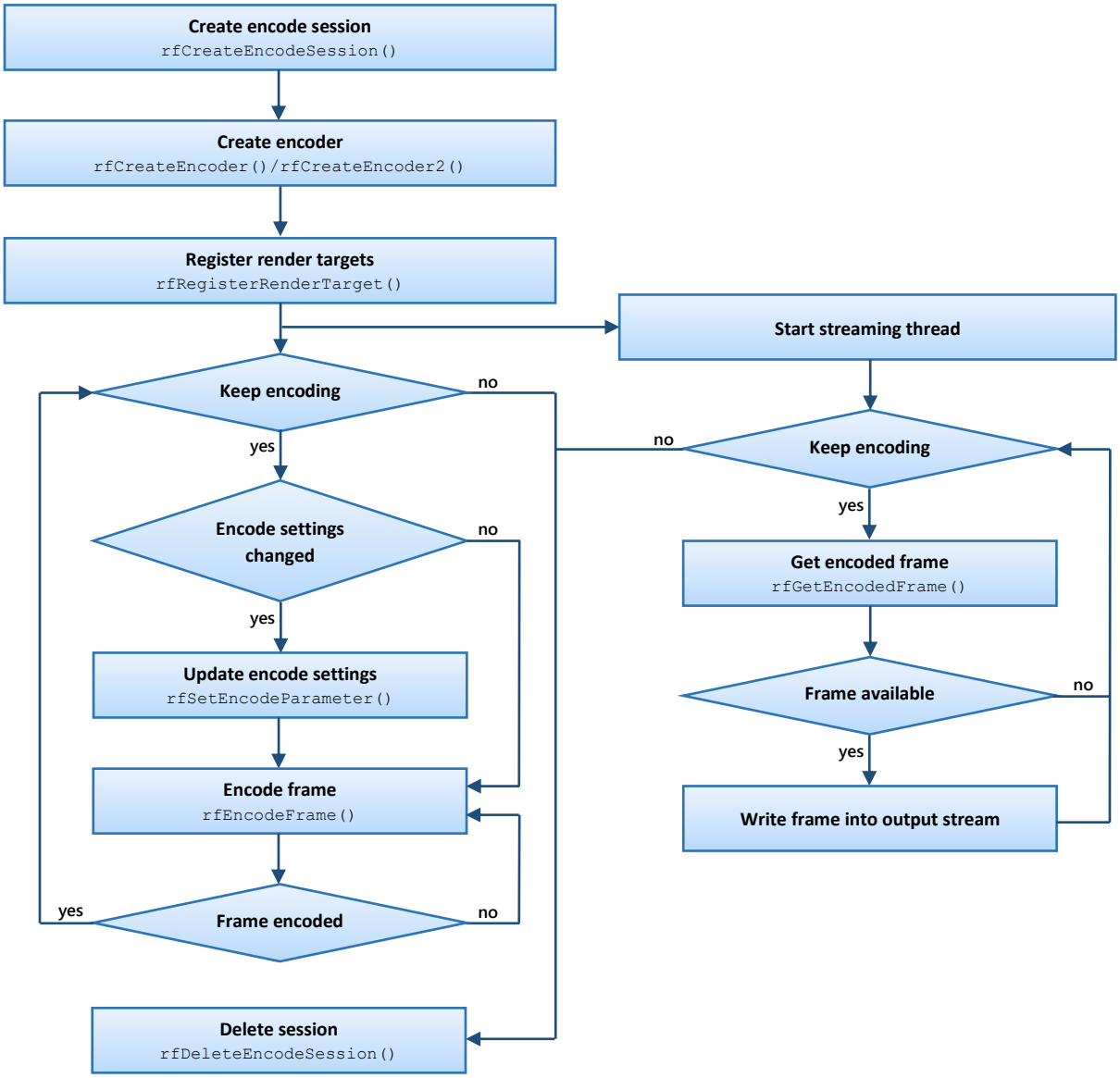
This document is intended for programmers. Prior knowledge in writing code for CPUs and a basic understanding of video encoding, video decoding, OpenGL, DirectX 9 and DirectX 11 is assumed, while a basic understanding of GPU architectures and computer graphics is useful.

RapidFire SDK Overview

The RapidFire SDK enables ISVs to implement remote rendering solutions.

It handles the asynchronous encoding of the frame buffer as a H.264, H.265 or uncompressed stream; it supports the following rendering APIs: OpenGL 4.2 and later, DirectX 9 and DirectX 11. The API also provides functions that allow the capturing and encoding of an entire desktop.

API Flow Diagram:



The RapidFire API

rfCreateEncodeSession

```
RFStatus rfCreateEncodeSession(RFEncodeSession* session,
                               RFProperties* properties)
```

Creates a RapidFire encode session.

Parameters:

session [out]	The created RapidFire encode session.
properties [in]	A list of session property names and their corresponding values.

A RapidFire session encapsulates the encoder to be used, the graphics context that is used by the calling application or the OpenCL context that is used for the color space conversion. For every session, a log file containing further information is generated in the current directory of the application. The location of the log file can be changed with the environment variable **RF_LOG_PATH**.

A session can have different sources for the encoding. It can be a render target or the whole desktop. The following sessions can be created:

- OpenGL session.
- DirectX 9 / DirectX 9 Ex session.
- DirectX 11 session.
- Desktop capturing session.

The parameter *properties* specifies a list of session property names and their corresponding values. Each property name is immediately followed by the corresponding value. The list is terminated with `0`. Depending on the graphics API that is used by the application, this function takes the following properties as input:

Property name	Description	Property type
RF_GL_GRAPHICS_CTX	OpenGL Context	HGLRC
RF_GL_DEVICE_CTX	Device Context	HDC
RF_D3D9_DEVICE	Direct3D 9 Device	IDirect3DDevice9*
RF_D3D9EX_DEVICE	Direct3D 9 Ex Device	IDirect3DDevice9Ex*
RF_D3D11_DEVICE	Direct3D 11 Device	ID3D11Device*
RF_DESKTOP	Select the desktop to capture according to the desktop id.	unsigned int
RF_DESKTOP_DSP_ID	Select the desktop to capture according to the windows display id.	unsigned int

RF_DESKTOP_INTERNAL_ - DSP_ID	Select the desktop to capture according to the display id used internally by RapidFire.	unsigned int
RF_DESKTOP_UPDATE_ - ON_CHANGE	If set to TRUE **rfEncodeFrame** will only enqueue a captured frame if the desktop has changed. (Only available if application is running in a virtual machine)	BOOL
RF_DESKTOP_BLOCK_ - UNTIL_CHANGE	If set to TRUE **rfEncodeFrame** will block until the desktop has changed. (Only available if application is running in a virtual machine)	BOOL
RF_FLIP_SOURCE	If set to TRUE the source image is flipped vertically.	BOOL
RF_ASYNC_SOURCE_COPY	If set to TRUE the captured frame will be copied into system memory asynchronously.	BOOL
RF_ENCODER	Specifies the encoder that will be used.	RFEncoderID
RF_ENCODER_BLOCKING_ - READ	If set to TRUE **rfGetEncodedFrame** will block until AMF has finished encoding a frame.	BOOL
RF_MOUSE_DATA	Enables capturing of the mouse cursor shape.	BOOL

Possible **RFEncoderID**s:

RF_AMF	RF_IDENTITY	RF_DIFFERENCE

AMF stands for Advanced Media Framework API encoder, that encodes an H.264 or H.265 stream. The **IDENTITY** encoder returns a buffer with the uncompressed source image in the specified color format and the **DIFFERENCE** encoder returns a difference map consisting of blocks with 1 being stored in regions where the source image has changed and 0 otherwise.

rfCreateEncodeSession returns **RF_STATUS_OK** if the session was created successfully. Otherwise the session is set to NULL and one of the following error codes is returned:

RF_STATUS_INVALID_SESSION_PROPERTIES if one or more of the properties is invalid.

RF_STATUS_INVALID_ENCODER if **RF_ENCODER** is not one of the possible **RFEncoderID**s.

RF_STATUS_INVALID_D3D_DEVICE if the input Direct3D device is invalid.

RF_STATUS_INVALID_OPENGL_CONTEXT if the input OpenGL context is invalid.

RF_STATUS_INVALID_DESKTOP_ID if the input Desktop ID is invalid.

RF_STATUS_INVALID_OPENCL_ENV if the OpenCL context creation failed.

RF_STATUS_AMF_FAIL if AMF functions failed.

RF_STATUS_DOPP_FAIL if DOPP extension initialization failed.

RF_STATUS_OPENCL_FAIL if OpenCL functions failed.

RF_STATUS_OPENGL_FAIL if the OpenGL context creation failed.

RF_STATUS_MEMORY_FAIL if a memory allocation failed.

RF_STATUS_FAIL if other errors occurred. More information is given in the log file.

rfDeleteEncodeSession

```
RFStatus rfDeleteEncodeSession(RFEncodeSession* session)
```

Deletes the encoding session and frees all associated resources.

Parameters:

session [in]	The encoding session to be deleted.

rfDeleteEncodeSession returns **RF_STATUS_OK** if the session was deleted successfully. Otherwise, it returns the following error value:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

rfCreateEncoder

```
RFStatus rfCreateEncoder(RFEncodeSession session,
                         unsigned int uiWidth,
                         unsigned int uiHeight,
                         const RFEncodePreset preset)
```

Creates the encoder and resources like intermediate buffers. It initializes all properties based on the selected encoding preset.

Parameters:

session [in]	The encoding session.
uiWidth [in]	The width of the encoded stream.
uiHeight [in]	The height of the encoded stream.
preset [in]	The encoding preset used.

Possible **RFEncodePresets**:

RF_PRESET_FAST	RF_PRESET_BALANCED	RF_PRESET_QUALITY
RF_PRESET_HEVC_FAST	RF_PRESET_HEVC_BALANCED	RF_PRESET_HEVC_QUALITY

The presets containing **HEVC** are used to set up the encoder with the H.265 video codec. For the other parameters, the encoder is set up with the H.264 video codec.

FAST stands for fast encoding with low latency. **BALANCED** is balanced between quality and latency. **QUALITY** stands for high video quality. The specific settings for the different presets can be found in the properties table for **`rfCreateEncoder2`**.

Based on the *preset*, *width* and *height*, the encoder is configured. If desktop encoding is selected and the dimensions for the encoding differ from the desktop size, the image is scaled to match the encoding dimensions.

rfCreateEncoder returns **RF_STATUS_OK** if the encoder is created successfully. Otherwise, it returns one of the following error values:

`RF_STATUS_INVALID_SESSION` if *session* parameter is not a valid session.

`RF_STATUS_INVALID_CONFIG` if no preset was specified.

`RF_STATUS_INVALID_OPENCL_CONTEXT` if the OpenCL context of the session is invalid.

`RF_STATUS_INVALID_FORMAT` if an invalid data format was specified for the encoder.

`RF_STATUS_INVALID_DIMENSION` if either *uiWidth* or *uiHeight* is `0` or larger than the maximum supported size of the encoder.

`RF_STATUS_INVALID_ENCODER` if no encoder was specified with the creation of the session.

`RF_STATUS_AMF_FAIL` if the AMF initialization failed.

`RF_STATUS_INVALID_ENCODER_PARAMETER` if an encoder property is not supported or has an invalid value.

`RF_STATUS_OPENCL_FAIL` if OpenCL resources creation failed.

`RF_STATUS_FAIL` if the creation of the encoder failed or the encoder is already created. More information is given in the log file.

rfCreateEncoder2

```
RFStatus rfCreateEncoder2(RFEncodeSession session,
                          unsigned int uiWidth,
                          unsigned int uiHeight,
                          const RFProperties* properties)
```

Creates the encoder with the static and dynamic properties specified.

Parameters:

session [in]	The encoding session.
uiWidth [in]	The width of the encoded stream.

uiHeight [in]	The height of the encoded stream.
Properties [in]	A list of **RFProperties** and their corresponding values. The list is terminated with `0`.

Properties specifies a list of encoder property names and their corresponding values. Each property name is immediately followed by the corresponding value. The list is terminated with `0`. All properties not set by the user are set to the **RF_PRESET_FAST** or **RF_PRESET_HEVC_FAST** preset, depending on the video codec used. Only properties matching the video codec used can be set.

rfCreateEncoder2 takes the following properties as input:

Property name	Description	Property type	Possible values and presets
RF_ENCODER_CODEC	Selects the video codec used for the encoder.	RFVideoCodec	RF_VIDEO_CODEC_NONE RF_VIDEO_CODEC_AVC (default) RF_VIDEO_CODEC_HEVC
RF_ENCODER_FORMAT	Selects the input format of the encoder.	RFFormat	RF_RGBA8 RF_ARGB8 RF_BGRA8 RF_NV12
RF_ENCODER_PROFILE	[H.264] Selects the profile.	unsigned int	66 – Baseline 77 – Main (default) 100 – High
RF_ENCODER_LEVEL	[H.264] Selects the profile level.	unsigned int	10, 11, 12, 13, 20, 21, 22, 30, 31, 32, 40, 41, 42 (default), 50, 51, 52
RF_ENCODER_USAGE	[H.264] Selects the encoder usage. This parameter configures the whole parameter set to a preset according to the usage.	int	-1 – Not set (default) 0 – Transcoding (quality preset) 1 – Ultra low latency (fast preset) 2 – Low latency (balanced preset) 3 – Webcam
RF_ENCODER_BITRATE	[H.264] Sets the target bitrate.	unsigned int	10 KBits/s – 100 MBits/s 20 MBit/s (default)
RF_ENCODER_PEAK_BITRATE	[H.264] Sets the peak bitrate.	unsigned int	10 KBits/s – 100 MBits/s 30 MBit/s (default)
RF_ENCODER_RATE_CONTROL_METHOD	[H.264] Selects the rate control method.	unsigned int	0 – Constant QP 1 – Constant Bitrate 2 – Peak Constrained VBR (balanced and quality preset) 3 – Latency Constrained VBR (fast preset)
RF_ENCODER_MIN_QP	[H.264] Sets the minimum quantizer parameter.	unsigned int	0 – 51 22 (fast and balanced preset) 18 (quality preset)
RF_ENCODER_MAX_QP	[H.264] Sets the maximum quantizer parameter.	unsigned int	0 – 51 48 (fast and balanced preset) 46 (quality preset)
RF_ENCODER_FRAME_RATE	[H.264] Frame rate numerator.	unsigned int	1*FrameRateDen – 120*FrameRateDen 30 (default)

RF_ENCODER_FRAME_RATE_DEN	[H.264] Frame rate denominator.	unsigned int	1 – MaxInt 1 (default)
RF_ENCODER_VBV_BUFFER_SIZE	[H.264] Sets the Video Buffering Verifier buffer size in bits.	unsigned int	1 KBit – 100 MBit 735 KBit (fast preset) 4 MBit (balanced preset) 20 MBit (quality preset)
RF_ENCODER_VBV_BUFFER_ - FULLNESS	[H.264] Sets the initial VBV buffer initial fullness.	unsigned int	0 – 64 (0% - 100% of VBV buffer size) 64 (default)
RF_ENCODER_ENFORCE_HRD	[H.264] Disables/enables constraints on QP variation within a picture to meet HRD requirement(s).	bool	true (fast preset) false (balanced and quality preset)
RF_ENCODER_ENABLE_VBAQ	[H.264] Disables/enables Variance Based Adaptive Quantization.	bool	false (default)
RF_ENCODER_IDR_PERIOD	[H.264] Sets IDR period. IDRPeriod = 0 turns IDR off.	unsigned int	0 – 1000 300 (fast and balanced preset) 30 (quality preset)
RF_ENCODER_INTRA_ - REFRESH_NUM_MB	[H.264] Sets the number of intra-refresh macroblocks per slot.	unsigned int	0 - #MBs per frame 225 (fast and balanced preset) 0 (quality preset)
RF_ENCODER_DEBLOCKING_ - FILTER	[H.264] Turns on/off the de-blocking filter.	bool	true (default)
RF_ENCODER_NUM_SLICES_ - PER_FRAME	[H.264] Sets the number of slices per frame.	unsigned int	1 - #MBs per frame 1 (default)
RF_ENCODER_QUALITY_PRESET	[H.264] Selects the quality preset.	unsigned int	0 (Balanced) (quality preset) 1 (Speed) (fast and balanced preset) 2 (Quality)
RF_ENCODER_HALF_PIXEL	[H.264] Turns on/off half-pixel motion estimation.	unsigned int	0, 1 1 (default)
RF_ENCODER_QUARTER_PIXEL	[H.264] Turns on/off quarter-pixel motion estimation.	unsigned int	0, 1 1 (default)
RF_ENCODER_FORCE_INTRA_ - REFRESH	[H.264] Forces the picture type to IDR (pre submission).	bool	false (default)
RF_ENCODER_FORCE_I_FRAME	[H.264] Forces the picture type to I-frames (pre submission).	bool	false (default)
RF_ENCODER_FORCE_P_FRAME	[H.264] Forces the picture type to P-frames (pre submission).	bool	false (default)
RF_ENCODER_INSERT_SPS	[H.264] Inserts Sequence Parameter Set (pre submission).	bool	false (default)
RF_ENCODER_INSERT_PPS	[H.264] Inserts Picture Parameter Set (pre submission).	bool	false (default)
RF_ENCODER_INSERT_AUD	[H.264] Inserts Access Unit Delimiter (pre submission).	bool	false (default)
RF_ENCODER_HEVC_USAGE	[H.265] Selects the encoder usage. This parameter configures the whole parameter set to a preset according to the usage.	int	-1 – Not set (default) 0 – Transcoding (quality preset) 1 – Ultra low latency (fast preset) 2 – Low latency (balanced preset) 3 – Webcam

RF_ENCODER_HEVC_PROFILE	[H.265] Selects the profile.	unsigned int	1 – Main (default)
RF_ENCODER_HEVC_LEVEL	[H.265] Selects the profile level.	unsigned int	30 (1.0), 60 (2.0), 63 (2.1), 90 (3.0), 93 (3.1), 120 (4.0), 123 (4.1), 150 (5.0), 153 (5.1), 156 (5.2), 180 (6.0), 183 (6.1), 189 (6.2) (default)
RF_ENCODER_HEVC_TIER	[H.265] Selects the tier.	unsigned int	0 – Main (default) 1 – High
RF_ENCODER_HEVC_RATE_ - CONTROL_METHOD	[H.265] Selects the rate control method.	unsigned int	0 – Constant 1 – Latency Constrained VBR (fast preset) 2 – Peak Constrained VBR (balanced and quality preset) 3 – Constant Bitrate
RF_ENCODER_HEVC_FRAMERATE	[H.265] Frame rate numerator.	unsigned int	1*FrameRateDen - 60*FrameRateDen 30 (default)
RF_ENCODER_HEVC_ - FRAMERATE_DEN	[H.265] Frame rate denominator.	unsigned int	1 – MaxInt 1 (default)
RF_ENCODER_HEVC_VBV_ - BUFFER_SIZE	[H.265] Sets the Video Buffering Verifier buffer size in bits.	unsigned int	1 KBit – 100 MBit 735 KBit (fast preset) 4 MBit (balanced preset) 20 MBit (quality preset)
RF_ENCODER_HEVC_INITIAL_ - VBV_BUFFER_FULLNESS	[H.265] Sets the initial VBV buffer initial fullness.	unsigned int	0 – 64 (0% - 100% of VBV buffer size) 64 (default)
RF_ENCODER_HEVC_RATE_ - CONTROL_PREANALYSIS_ENABLE	[H.265] Disables/enables pre-analysis assisted rate control.	bool	false (default)
RF_ENCODER_HEVC_ENABLE_ - VBAQ	[H.265] Disables/enables Variance Based Adaptive Quantization.	bool	false (default)
RF_ENCODER_HEVC_TARGET_ - BITRATE	[H.265] Sets the target bitrate.	unsigned int	10 KBits/s – 100 MBits/s 20 MBit/s (default)
RF_ENCODER_HEVC_PEAK_ - BITRATE	[H.265] Sets the peak bitrate.	unsigned int	10 KBits/s – 100 MBits/s 30 MBit/s (default)
RF_ENCODER_HEVC_MIN_QP_I	[H.265] Sets the minimum QP for I frame.	unsigned int	0 – 51 22 (fast and balanced preset) 18 (quality preset)
RF_ENCODER_HEVC_MAX_QP_I	[H.265] Sets the maximum QP for I frame.	unsigned int	0 – 51 48 (fast and balanced preset) 46 (quality preset)
RF_ENCODER_HEVC_MIN_QP_P	[H.265] Sets the minimum QP for P frame.	unsigned int	0 – 51 22 (fast and balanced preset) 18 (quality preset)
RF_ENCODER_HEVC_MAX_QP_P	[H.265] Sets the maximum QP for I frame.	unsigned int	0 – 51 48 (fast and balanced preset) 46 (quality preset)
RF_ENCODER_HEVC_QP_I	[H.265] Sets the constant QP for I-pictures. (Only available for CQP rate control method.)	unsigned int	0 – 51 26 (default)

RF_ENCODER_HEVC_QP_P	[H.265] Sets the constant QP for P-pictures. (Only available for CQP rate control method.)	unsigned int	0 – 51 26 (default)
RF_ENCODER_HEVC_ - ENFORCE_HRD	[H.265] Disables/enables constraints on QP variation within a picture to meet HRD requirement(s).	bool	true (fast preset) false (balanced and quality preset)
RF_ENCODER_HEVC_MAX_AU_ - SIZE	[H.265] Maximum AU size in bits.	unsigned int	0 – 100 Mbits 0 (default)
RF_ENCODER_HEVC_FILLER_ - DATA_ENABLE	[H.265] Enable filler data for CBR usage.	bool	false (default)
RF_ENCODER_HEVC_RATE_ - CONTROL_SKIP_FRAME_ENABLE	[H.265] Enables skip frame for rate control.	bool	true (fast and balanced preset) false (quality preset)
RF_ENCODER_HEVC_HEADER_ - INSERTION_MODE	[H.265] Sets the headers insertion mode.	unsigned int	0 – None (default) 1 – GOP aligned 2 – IDR aligned
RF_ENCODER_HEVC_GOP_SIZE	[H.265] The period to insert IDR/CRA in fixed size mode. 0 means only insert the first IDR/CRA (infinite GOP size).	unsigned int	0 – 1000 300 (fast and balanced preset) 30 (quality preset)
RF_ENCODER_HEVC_NUM_ - GOPS_PER_IDR	[H.265] Determines the frequency to insert IDR as start of a GOP. 0 means no IDR will be inserted except for the first picture in the sequence.	unsigned int	1 – 65535
RF_ENCODER_HEVC_DE_ - BLOCKING_FILTER_ENABLE	[H.265] Disables/enables the deblocking filter.	bool	false (default)
RF_ENCODER_HEVC_SLICES_ - PER_FRAME	[H.265] Sets the number of slices per frame.	unsigned int	1 - #CTBs per frame 1 (default)
RF_ENCODER_HEVC_QUALITY_ - PRESET	[H.265] Selects the quality preset.	unsigned int	0 – Balanced 5 – Quality (quality preset) 10 – Speed (fast and balanced preset)
RF_ENCODER_HEVC_MOTION_ - HALF_PIXEL	[H.265] Turns on/off half-pixel motion estimation.	bool	true (default)
RF_ENCODER_HEVC_MOTION_ - QUARTERPIXEL	[H.265] Turns on/off quarter-pixel motion estimation.	bool	true (default)
RF_ENCODER_HEVC_FORCE_ - INTRA_REFRESH	[H.265] Forces the picture type to IDR (pre submission).	bool	false (default)
RF_ENCODER_HEVC_FORCE_I_ - FRAME	[H.265] Forces the picture type to I-frames (pre submission).	bool	false (default)
RF_ENCODER_HEVC_FORCE_P_ - FRAME	[H.265] Forces the picture type to P-frames (pre submission).	bool	false (default)
RF_ENCODER_HEVC_INSERT_ - HEADER	[H.265] Inserts SPS, PPS and VPS.	bool	false (default)
RF_ENCODER_HEVC_INSERT_ - AUD	[H.265] Inserts Access Unit Delimiter (pre submission).	bool	false (default)

RF_DIFF_ENCODER_BLOCK_S	Block width for difference encoder	unsigned int	BLOCK_S == multiple of 8 BLOCK_S * BLOCK_T == multiple of 64 16 (default)
RF_DIFF_ENCODER_BLOCK_T	Block height for difference encoder	unsigned int	BLOCK_T == multiple of 8 BLOCK_S * BLOCK_T == multiple of 64 16 (default)
RF_DIFF_ENCODER_LOCK_ - BUFFER	Should be set to true if the **RF_DIFFERENCE** encoder is used and **rfGetEncodedFrame** is called in a different thread than **rfEncodeFrame**.	bool	false (default)

RFFormat is the data format of the input for the encoder:

RF_ARGB8	RF_BGRA8	RF_RGBA8	RF_NV12

ARGB8, BGRA8 and **RGBA8** define 32-bit RGB values with an alpha channel, each pixel is represented by one byte each for the red, green, blue, and alpha channels. **NV12** is an 8-bit Y plane followed by an interleaved U/V plane with 2x2 subsampling.

rfCreateEncoder2 returns **RF_STATUS_OK** if the encoder is created successfully. Otherwise, it returns an error value listed in function **rfCreateEncoder** as well as the following error code:

RF_STATUS_INVALID_PROPERTIES if one of the properties is invalid.

rfRegisterRenderTarget

```
RFStatus rfRegisterRenderTarget(RFEncodeSession session,
                                RFRenderTarget renderTarget,
                                unsigned int uiRTWidth,
                                unsigned int uiRTHeight,
                                unsigned int* idx)
```

Registers a render target that is created by the user and returns the index used for this render target in *idx*. The render target must have the same dimensions as the encoder. Otherwise the call fails. Up to 3 render targets can be registered and used for triple buffering.

Parameters:

session [in]	The encoding session.
renderTarget [in]	The handle of the render target.
uiRTWidth [in]	The width of the render target.
uiRTHeight [in]	The height of the render target.
idx [out]	The index used for this render target.

Possible **RFRenderTarget** formats are:

GLuint	OpenGL texture
ID3D11Texture2D*	DirectX 11 texture
IDirect3DSurface9*	DirectX 9 texture

rfRegisterRenderTarget returns **RF_STATUS_OK** if it registers a render target successfully. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *idx* is a null-pointer.

RF_STATUS_INVALID_DIMENSION if *uiRTWidth* or *uiRTHeight* is `0` or if the render target dimension does not match the encoder dimension.

RF_STATUS_INVALID_RENDER_TARGET if *renderTarget* is an invalid handle.

RF_STATUS_INVALID_OPENCL_CONTEXT if the OpenCL context of the session is invalid.

RF_STATUS_RENDER_TARGET_FAIL if the application has already registered the maximum number of 3 render targets.

RF_STATUS_OPENCL_FAIL if the creation of an OpenCL buffer for the render target failed.

RF_STATUS_FAIL if other errors occurred. More information is given in the log file.

rfRemoveRenderTarget

```
RFStatus rfRemoveRenderTarget(RFEncodeSession session,
                              unsigned int idx)
```

Removes a registered render target with the index *idx* that is no longer used in the session.

Parameters:

session [in]	The encoding session.
idx [in]	The index of the render target.

rfRemoveRenderTarget returns **RF_STATUS_OK** if it removes a registered render target successfully. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_INDEX if the *idx* exceeds the maximum number of 3 render targets.

RF_STATUS_INVALID_RENDER_TARGET if the render target is not known or not registered.

rfGetRenderTargetState

```
RFStatus rfGetRenderTargetState(RFEncodeSession session,
                                RFRenderTargetState* state,
                                unsigned int idx)
```

Gets the state of the render target with the index *idx*.

Parameters:

session [in]	The encoding session.
state [out]	The state of the render target.
idx [in]	The index of the render target.

RFRenderTargetState describes the state of a render target:

RF_STATE_INVALID	RF_STATE_FREE	RF_STATE_BLOCKED

INVALID: The render target is not known or not registered yet.

FREE: The render target was successfully registered and is currently not used by the API. It can be used by the application.

BLOCKED: The render target was submitted for encoding and is currently in use by the API. It should not be used by the application.

rfGetRenderTargetState returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *state* is a null-pointer.

RF_STATUS_INVALID_OPENCL_CONTEXT if the OpenCL context of the session is invalid.

RF_STATUS_INVALID_INDEX if *idx* is out of range or the render target was not registered.

rfResizeSession

```
RFStatus rfResizeSession(RFEncodeSession session,
                         unsigned int uiWidth,
                         unsigned int uiHeight)
```

Resizes the session and the encoder if the encoder supports resizing. Before calling **rfResizeSession** the encoding queue must be empty. Otherwise submitting additional frames to the encoding queue might fail. Render targets that are registered by the application will be removed and have to be registered again with the new size.

Parameters:

session [in]	The encoding session.
uiWidth [in]	The new width of the stream.
uiHeight [in]	The new height of the stream.

rfResizeSession returns **RF_STATUS_OK** if the resources of the session were resized successfully. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_FAIL if other errors occurred. More information is given in the log file.

rfEncodeFrame

```
RFStatus rfEncodeFrame(RFEncodeSession session,
                       unsigned int idx)
```

Is called once the application has finished rendering into the render target with the index idx. The render target will then be captured and encoded.

Parameters:

session [in]	The encoding session.
idx [in]	The index of the render target which will be encoded. (ignored for encoding sessions capturing the desktop)

rfEncodeFrame returns **RF_STATUS_OK** if capturing and encoding are successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_CONTEXT if the AMF context of the encoder is invalid.

RF_STATUS_INVALID_OPENCL_CONTEXT if the OpenCL context of the session is invalid.

RF_STATUS_INVALID_ENCODER if the encoder is invalid or not created.

RF_STATUS_INVALID_FORMAT if the format of the encoder is invalid.

RF_STATUS_INVALID_INDEX if *idx* is out of range or the render target was not registered.

RF_STATUS_INVALID_OPENCL_MEMOBJ if the encoder couldn't load the OpenCL result buffer for the encoding.

RF_STATUS_DOPP_NO_UPDATE if the session parameter **RF_DESKTOP_UPDATE_ON_CHANGE** was set and the desktop texture didn't change since the last call.

RF_STATUS_QUEUE_FULL if there aren't any more free result buffers in the queue left. Call **rfGetEncodedFrame** to free the buffers.

RF_STATUS_OPENCL_FAIL if an OpenCL functions failed.

RF_STATUS_AMF_FAIL if AMF encoding failed.

rfGetEncodedFrame

```
RFStatus rfGetEncodedFrame(RFEncodeSession session,
                           unsigned int* uiSize,
                           void** pBitStream)
```

Returns the encoded frame in *pBitStream* and the size in bytes of the encoded frame in *uiSize*.

For the **RF_DIFFERENCE** encoder the returned buffer is a 2D array of byte-blocks for the captured image, where a 1 stored in a block indicates that the image has changed. The captured image can be obtained with the **rfGetSourceFrame** function.

Parameters:

session [in]	The encoding session.
uiSize [out]	The size of the bit stream.
pBitStream [out]	The bit stream of the encoded frame.

rfGetEncodedFrame returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *uiSize* or *pBitStream* are a null-pointers.

RF_STATUS_INVALID_ENCODER if the encoder is invalid.

RF_STATUS_NO_ENCODED_FRAME if there is no encoded frame in the result buffer queue.

RF_STATUS_AMF_FAIL if an AMF function failed.

rfGetSourceFrame

```
RFStatus rfGetSourceFrame(RFEncodeSession session,
                          unsigned int* uisSize,
                          void** pBitStream)
```

Returns the image that was used as the input for the encoder. If a color format conversion is required for the encoder, the returned image contains the result of the color format conversion. The function needs to be called prior to **rfGetEncodedFrame** to guarantee that the returned image is the source image of the encoded image returned by **rfGetEncodedFrame**.

Parameters:

session [in]	The encoding session.
uiSize [out]	The size of the bit stream.

pBitStream [out]	The bit stream of the encoded frame.

rfGetSourceFrame returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *uiSize* or *pBitStream* are a null-pointers.

RF_STATUS_INVALID_ENCODER if the encoder is invalid.

RF_STATUS_NO_ENCODED_FRAME if there is no encoded frame in the result buffer queue.

rfSetEncodeParameter

```
RFStatus rfSetEncodeParameter(RFEncodeSession session,
                              const int property,
                              RFProperties value)
```

Sets encoding properties that do not require a re-creation of the encoder, e.g. adapting the encoding quality to the available network bandwidth. The available properties depend on the selected encoder. This function should be called after **rfCreateEncoder**/ **rfCreateEncoder2**, otherwise the function does not have any effect.

Parameters:

session [in]	The encoding session.
property [in]	The encoding property to change.
value [in]	The new value of the encoding property.

For a list of valid properties and descriptions refer to the table of properties for **rfCreateEncoder2.**

rfSetEncodeParameter returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_ENCODER if the encoder is invalid.

RF_STATUS_INVALID_CONFIG if the encoder configuration is invalid.

RF_STATUS_INVALID_ENCODER_PARAMETER if the property is invalid.

RF_STATUS_PARAM_ACCESS_DENIED if the encoder property can't be changed by the user.

RF_STATUS_FAIL if the encoder of the session is invalid or the encoder property can't be changed without recreating the encoder.

rfGetEncodeParameter

```
RFStatus rfGetEncodeParameter(RFEncodeSession session,
                              const int property,
                              RFProperties* value)
```

Queries the encoder property of the parameter *property*. The available properties depend on the selected encoder. This function should be called after **rfCreateEncoder** / **rfCreateEncoder2**, otherwise the function does not have any effect.

Parameters:

session [in]	The encoding session.
property [in]	The requested encoding property.
value [out]	The value of the encoding property.

Possible encoder properties that can be queried are:

Property name	Description	Property type
RF_ENCODER_FORMAT	The color format of the encoder output.	RFFormat
RF_ENCODER_WIDTH	The width of the render target.	unsigned int
RF_ENCODER_HEIGHT	The height of the render target.	unsigned int
RF_ENCODER_OUTPUT_WIDTH	The width of the encoder output.	unsigned int
RF_ENCODER_OUTPUT_HEIGHT	The height of the encoder output.	unsigned int

For a list of other valid properties and descriptions refer to the table of properties for **rfCreateEncoder2.**

rfGetEncodeParameter returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *value* is a null-pointer.

RF_STATUS_INVALID_ENCODER if the encoder is invalid.

RF_STATUS_PARAM_ACCESS_DENIED if the property is blocked by AMF.

RF_STATUS_INVALID_ENCODER_PARAMETER if the property queried is invalid.

rfGetMouseData

```
RFStatus rfGetMouseData(RFEncodeSession session,
                        int iWaitForShapeChange,
```

RFMouseData* mouseData)

Returns the mouse cursor shape data. To use it the session needs to be created with the **RF_MOUSE_DATA** set to true.

Parameters:

session [in]	The encoding session.
iWaitForShapeChange [in]	If set to 1 the call blocks until the mouse shape changed.
mouseData [out]	The returned mouse shape data.

The **RFBitmapBuffer** structure stores the bitmap data:

uiWidth	The width of the bitmap.
uiHeight	The height of the bitmap.
uiPitch	The Pitch of the bitmap.
uiBitsPerPixel	Bits per pixel of the bitmap.
pPixels	The data of the bitmap.

The **RFMouseData** structure stores the cursor shape data:

iVisible	Is 1 if the cursor is visible and 0 otherwise.
uiXHot	The horizontal position of the cursor hot spot.
uiYHot	The vertical position of the cursor hot spot.
mask	The cursor bitmask bitmap. If the cursor is monochrome, this bitmask is formatted so that the upper half is the cursor AND bitmask and the lower half is the XOR bitmask. If the cursor is colored, this mask defines the AND bitmask of the cursor.
color	The cursor color bitmap containing the color data. This member is optional. If the cursor is monochrome, color.pPixels is NULL. For pixels with false in the mask bitmap the color is directly blended with the destination pixel. For pixels with true in the mask bitmap the color is XORed with the color of the destination pixel.

rfGetMouseData returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *mouseData* is a null-pointer.

RF_STATUS_MOUSEGRAB_NO_CHANGE if the cursor shape did not change.

RF_STATUS_FAIL if the session was not created with RF_MOUSE_DATA set to true.

rfGetMouseData2

```
RFStatus rfGetMouseData2(RFEncodeSession session,
                         int iWaitForShapeChange,
                         RFMouseData2* mouseData)
```

Returns the mouse cursor shape data. To use it the session needs to be created with the **RF_MOUSE_DATA** set to true.

Parameters:

session [in]	The encoding session.
iWaitForShapeChange [in]	If set to 1 the call blocks until the mouse shape changed.
mouseData [out]	The returned mouse shape data.

The **RFBitmapBuffer** structure stores the bitmap data:

uiWidth	The width of the bitmap.
uiHeight	The height of the bitmap.
uiPitch	The Pitch of the bitmap.
uiBitsPerPixel	Bits per pixel of the bitmap.
pPixels	The data of the bitmap.

The **RFMouseData2** structure stores the cursor shape data:

iVisible	Is 1 if the cursor is visible and 0 otherwise.
uiXHot	The horizontal position of the cursor hot spot.
uiYHot	The vertical position of the cursor hot spot.
uiFlags	The type of the cursor. 1 for monochrome cursors, 2 for color cursors and 4 for masked color cursors.
pShape	A RFBitmapbuffer containing the cursor shape data. The pPixels buffer contains the cursor shape data in a format that is compatible with the DXGKARG_SETPOINTERSHAPE struct used in the kernel mode driver function DxgkddiSetpointershape.

rfGetMouseData2 returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_INVALID_PARAMETER if *mouseData* is a null-pointer.

RF_STATUS_MOUSEGRAB_NO_CHANGE if the cursor shape did not change.

RF_STATUS_FAIL if the session was not created with RF_MOUSE_DATA set to true.

rfReleaseEvent

```
RFStatus rfReleaseEvent(RFEncodeSession session,
                        RFNotification const rfNotification)
```

Signals a notification event. This can be used to unblock a thread that is waiting for the event to be signaled.

Parameters:

session [in]	The encoding session.
rfNotification [in]	Specifies which event to signal.

RapidFire uses **RFNotification** events to get signaled on mouse cursor shape changes and desktop changes:

RFDesktopNotification	RFMouseShapeNotification

The **RFDesktopNotification** event is used to unblock a thread that called **rfEncodeFrame** for a RapidFire session with the property **RF_DESKTOP_BLOCK_UNTIL_CHANGE** set to 1.

The **RFMouseShapeNotification** event is used to unblock a thread that called **rfGetMouseData** with the *iWaitForShapeChange* parameter set to 1.

rfReleaseEvent returns **RF_STATUS_OK** if successful. Otherwise, it returns one of the following error values:

RF_STATUS_INVALID_SESSION if *session* is not a valid session.

RF_STATUS_FAIL if **RFNotification** is invalid or the event was not created.