

RZ/A2Mグループ

グラフィックスライブラリ RGA

要旨

本アプリケーションノートでは、RZ/A2Mのグラフィックスライブラリ RGA（Renesas Graphics Architecture）について説明します。

RGA の特長を以下に示します。

- ・ ハードウェア アクセラレーションを使って高速に描画
- ・ W3C 標準である HTML Canvas 2D Context をベースに作成した API のため、学習が容易
- ・ アプリケーションが用意したメモリー領域を描画対象、または、入力画像として使用可能
- ・ 半透明な画像の描画
- ・ 画像ファイルをグローバル変数としてアクセスできるようにする変換ツールを付属（ホスト PC 上で動作）

動作確認デバイス

RZ/A2M

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

制限事項

本ライブラリは、ベクターグラフィックスには対応していません。

本ライブラリは、マルチタスクでの使用は非対応です。単一タスクで RGA を使用してください。

本ライブラリは、アフィン変化にのみ対応しています。

目次

1. 仕様	6
2. ファイル構成	11
3. 動作確認条件	12
4. ソフトウェア説明	13
4.1 動作概要	13
4.1.1 バッファ描画モード	13
(1) フローチャート	13
(2) シーケンス図	13
4.1.2 表示描画モード	14
(1) フローチャート	14
(2) シーケンス図	15
4.2 定数／型／クラス／関数	17
4.2.1 RZ/A1用RGAとの比較	17
4.2.2 エラーコード	20
4.2.3 byte_per_pixel_t の図	20
4.3 移植ガイド	21
4.4 補足	22
4.4.1 Canvas 2D との対応表、H/Wアクセラレーション対応表	22
4.4.2 初期化関数の内部の動き	24
4.4.3 画像の形式の識別	25
4.4.4 デフォルト可能フラグ	26
4.4.5 内部変数を定数で初期化する関数について（*_InitConst関数）	28
4.4.6 終了処理について（*_Finalize関数）	29
5. 関数／メソッド	31
5.1 graphics_t クラスのメンバー関数に相当する関数	31
5.1.1 一覧	31
5.1.2 R_GRAPHICS_STATIC_GetVersion	33
5.1.3 R_GRAPHICS_PrintRegisters	33
5.1.4 R_GRAPHICS_Start	33
5.1.5 R_GRAPHICS_FinishPreviousFrame	33
5.1.6 R_GRAPHICS_StartPreviousFrame	33
5.1.7 R_GRAPHICS_GetHasFramePipeline	34
5.1.8 R_GRAPHICS_BeginSoftwareRendering2	34
5.1.9 R_GRAPHICS_BeginSoftwareRenderingA	34
5.1.10 R_GRAPHICS_CONFIG_SetEmpty	34
5.1.11 R_GRAPHICS_InitConst	34
5.1.12 R_GRAPHICS_Initialize	35
5.1.13 R_GRAPHICS_Finalize	35
5.1.14 R_GRAPHICS_SetFrameBuffer	35
5.1.15 R_GRAPHICS_GetFrameBuffer	36
5.1.16 R_GRAPHICS_Finish	36

5.1.17 R_GRAPHICS_Save	36
5.1.18 R_GRAPHICS_Restore	37
5.1.19 R_GRAPHICS_ResetMatrix	37
5.1.20 R_GRAPHICS_SetMatrix_2x3	37
5.1.21 R_GRAPHICS_SetMatrix_3x3	37
5.1.22 R_GRAPHICS_GetMatrix_3x3	38
5.1.23 R_GRAPHICS_TranslateMatrixl	38
5.1.24 R_GRAPHICS_TranslateMatrix	38
5.1.25 R_GRAPHICS_ScaleMatrix	39
5.1.26 R_GRAPHICS_RotateMatrixDegree	39
5.1.27 R_GRAPHICS_ShearMatrix	40
5.1.28 R_GRAPHICS_TransformMatrix	41
5.1.29 R_GRAPHICS_MultiplyMatrix	41
5.1.30 R_GRAPHICS_GetProjectiveMatrix	42
5.1.31 R_GRAPHICS_SetBackgroundColor	42
5.1.32 R_GRAPHICS_GetBackgroundColor	43
5.1.33 R_GRAPHICS_GetClearColor	43
5.1.34 R_GRAPHICS_Clear	43
5.1.35 R_GRAPHICS_DrawImage	44
5.1.36 R_GRAPHICS_DrawImageResized	45
5.1.37 R_GRAPHICS_DrawImageChild	46
5.1.38 R_GRAPHICS_FillRect	47
5.1.39 R_GRAPHICS_SetFillColor	47
5.1.40 R_GRAPHICS_SetFillPattern	47
5.1.41 R_GRAPHICS_BeginPath	47
5.1.42 R_GRAPHICS_Rect	48
5.1.43 R_GRAPHICS_Clip	48
5.1.44 R_GRAPHICS_SetGlobalAlpha	48
5.1.45 R_GRAPHICS_GetGlobalAlpha	49
5.1.46 R_GRAPHICS_SetGlobalCompositeOperation	49
5.1.47 R_GRAPHICS_GetGlobalCompositeOperation	49
5.1.48 R_GRAPHICS_SetQualityFlags	49
5.1.49 R_GRAPHICS_GetQualityFlags	50
5.1.50 R_GRAPHICS_SetStrokeColor	50
5.1.51 R_GRAPHICS_StrokeRect	50
5.1.52 R_GRAPHICS_BeginSoftwareRendering	50
5.1.53 R_GRAPHICS_EndSoftwareRendering	51
5.1.54 R_GRAPHICS_EndRenderingInFin	51
5.2 graphics_image_tクラスのメンバー関数に相当する関数	52
5.2.1 一覧	52
5.2.2 R_GRAPHICS_IMAGE_InitByShareFrameBuffer	52
5.2.3 R_GRAPHICS_IMAGE_InitR8G8B8A8	53
5.2.4 R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8	53
5.2.5 R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8	54
5.2.6 R_GRAPHICS_IMAGE_GetProperties	54
5.2.7 R_GRAPHICS_IMAGE_InitByShareFrameBufferEx	55
5.2.8 R_GRAPHICS_IMAGE_GetImageFormat	55

5.3	graphics_pattern_tクラスのメンバー関数に相当する関数	55
5.3.1	一覧	55
5.3.2	R_GRAPHICS_PATTERN_Initialize	55
5.4	window_surfaces_tクラスのメンバー関数に相当する関数	56
5.4.1	一覧	56
5.4.2	R_WINDOW_SURFACES_InitConst	56
5.4.3	R_WINDOW_SURFACES_Initialize	57
5.4.4	R_WINDOW_SURFACES_Finalize	57
5.4.5	R_WINDOW_SURFACES_GetLayerFrameBuffer	57
5.4.6	R_WINDOW_SURFACES_GetLayerCount	57
5.4.7	R_WINDOW_SURFACES_SwapBuffers	58
5.4.8	R_WINDOW_SURFACES_DoMessageLoop	58
5.4.9	R_WINDOW_SURFACES_AccessLayerAttributes	58
5.4.10	R_WINDOW_SURFACES_AllocOffscreenStack	59
5.4.11	R_WINDOW_SURFACES_FreeOffscreenStack	59
5.4.12	R_WINDOW_SURFACES_SetLayerAttributes	59
5.4.13	R_WINDOW_SURFACES_CONFIG_SetEmpty	60
5.4.14	R_LAYER_ATTRIBUTES_SetEmpty	60
5.5	byte_per_pixel_tクラスに関連する関数	61
5.5.1	一覧	61
5.5.2	R_RGA_BitPerPixelType_To_BytePerPixelType	61
5.5.3	R_RGA_BytePerPixelType_To_BitPerPixelType	61
5.5.4	R_BYTE_PER_PIXEL_IsInteger	61
5.5.5	R_BIT_PER_PIXEL_GetBytePerPixel	62
5.5.6	R_BYTE_PER_PIXEL_GetBitPerPixel	62
5.5.7	R_RGA_BYTE_PER_PIXEL_IsInteger	62
5.6	animation_timing_function_tクラスに関連する関数	63
5.6.1	一覧	63
5.6.2	R_Get_AnimationTimingFunction	63
5.6.3	R_ANIMATION_TIMING_FUNCTION_GetValue	64
5.7	JCU ドライバ OSポーティングレイヤ	65
5.7.1	一覧	65
5.7.2	R_GRAPHICS_OnInitialize	65
5.7.3	R_GRAPHICS_OnFinalize	65
5.7.4	R_GRAPHICS_OnInitialized	66
5.7.5	R_GRAPHICS_JCU_ClearCodecEvent	66
5.7.6	R_GRAPHICS_JCU_SetCodecEvent	66
5.7.7	R_GRAPHICS_JCU_WaitForCodecEvent	66
5.7.8	R_GRAPHICS_JCU_ClearTerminateEvent	66
5.7.9	R_GRAPHICS_JCU_SetTerminateEvent	66
5.7.10	R_GRAPHICS_JCU_WaitForTerminateEvent	67
5.8	DRWドライバ OSポーティングレイヤ	68
5.8.1	一覧	68
5.8.2	R_DRW_OnInitialize	68
5.8.3	R_DRW_OnFinalize	68
5.8.4	R_DRW_EnableInterrupt	68
5.8.5	R_DRW_DisableInterrupt	69

5.8.6	R_DRW_WaitForInterruptEvent	69
5.8.7	R_DRW_SetInterruptEvent	70
5.8.8	R_DRW_CheckThread	70
5.8.9	R_DRW_FlushCache	70
5.8.10	R_DRW_ToPhysicalAddress	71
5.8.11	R_DRW_ToCachedAddress	71
5.8.12	R_DRW_ToUncachedAddress	72
5.8.13	R_DRW_OnInterrupting	72
5.9	その他の関数	73
5.9.1	一覧	73
5.9.2	R_RGA_Get_R8G8B8A8	73
5.9.3	R_RGA_CalcWorkBufferB_Size	74
6.	ツール説明	75
6.1	画像フォーマット変換 ImagePackager	75
6.1.1	操作手順	75
6.1.2	ファイル一覧	76
6.1.3	サンプル	76
6.1.4	出力バイナリの種類（言語）	77
6.1.5	出力バイナリ内のファイルフォーマット	77
6.1.6	入力フォーマット	78
6.1.7	BinaryImageConfig.image.xml に記述できるパラメーター	79
6.1.8	XML の基本的な書き方	87
(1)	XML	87
(2)	XPath	88
(3)	# フラグメント	88
6.2	バイナリ変換 ConvertBin	89
6.3	画像ファイル作成 RawToBmp	90
7.	参考ドキュメント	91
7.1	リファレンス シンボル	92
	改訂記録	93

1. 仕様

RGA は、グラフィックスイメージを描画します。表1.1に使用する周辺機能と用途を、
図1-2にブロック図を、表1.2、表1.3に対応しているピクセルフォーマットを示します。

表1.1 使用する周辺機能と用途

周辺機能	用途
2D 描画エンジン (DRW)	グラフィックス描画
JPEG コーデックユニット (JCU)	JPEG 伸張
ビデオディスプレイコントローラ 6 (VDC6)	画面表示

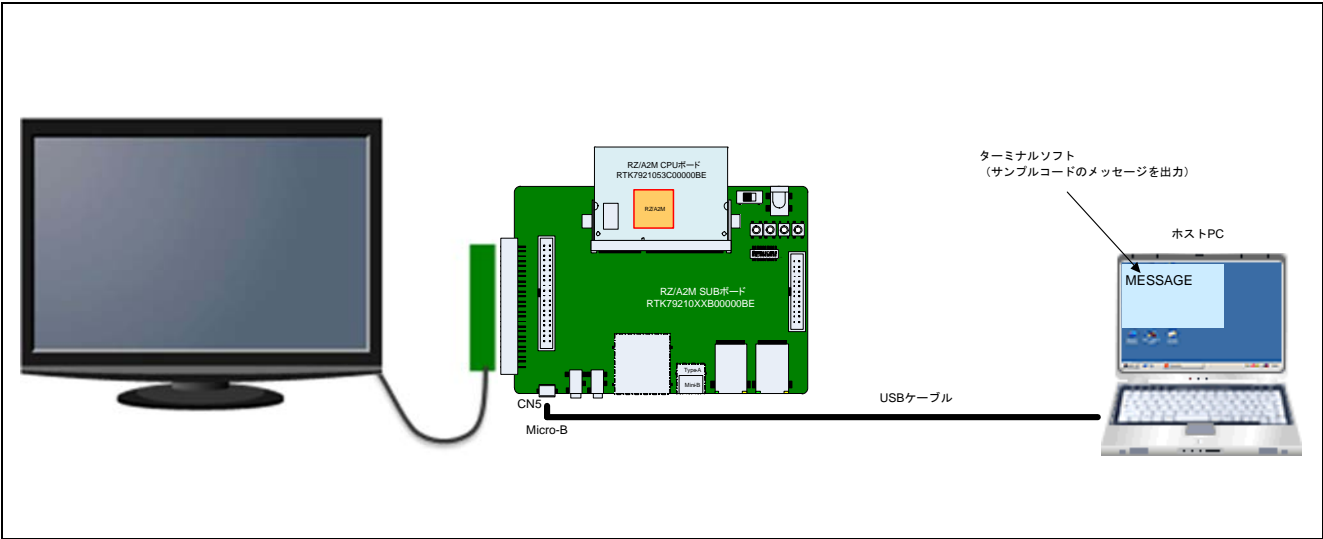


図1.1 動作環境

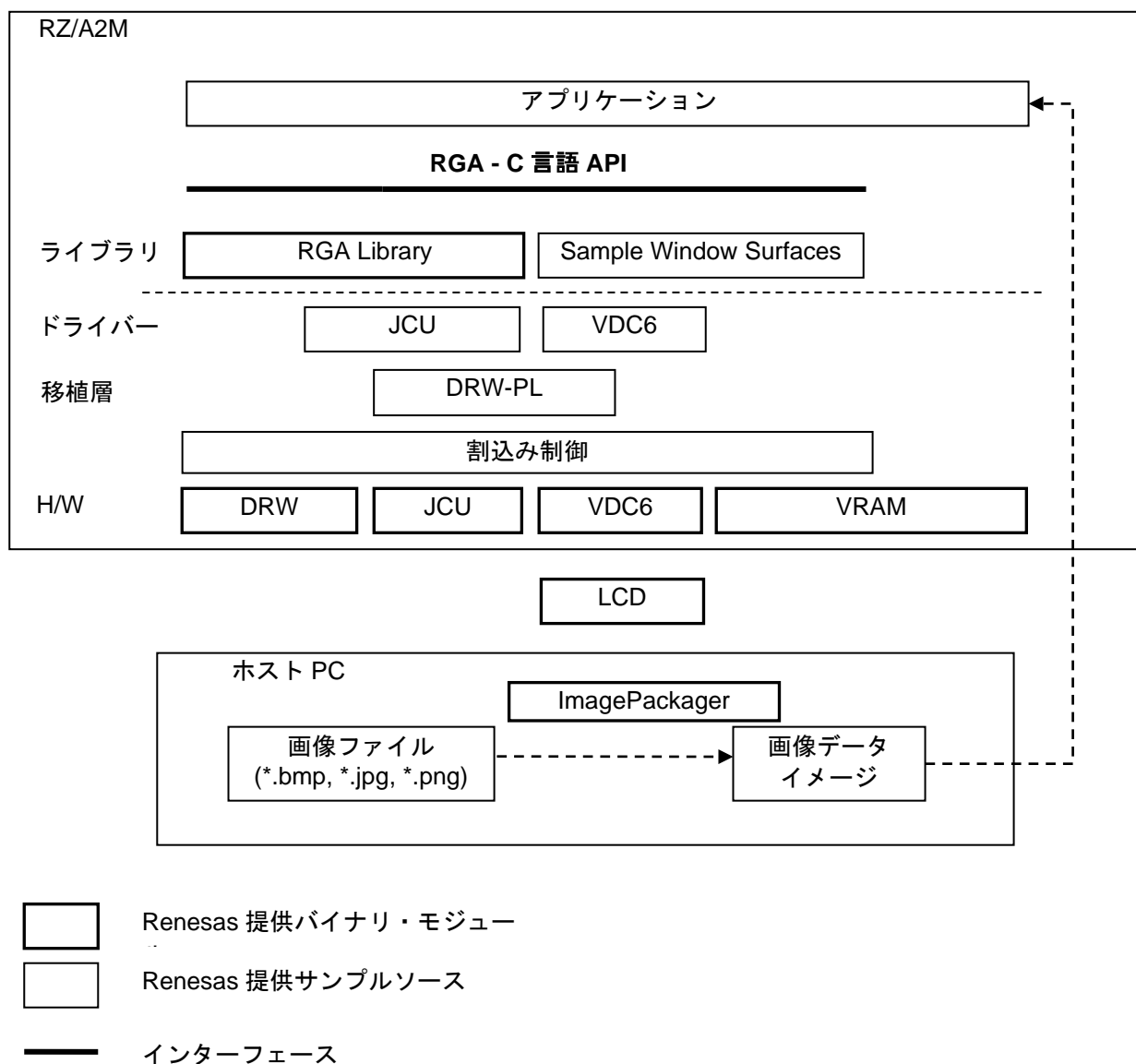


図1-2 ブロック図

本ライブラリの座標系は、左上が原点です。X 軸方向の値が増えると左から右の方向に進みます。Y 軸方向の値が増えると上から下の方向に進みます。描画対象のフレームバッファの大きさの最大は、幅が 2048 ピクセル、高さが 2048 ピクセルです。ソース画像の大きさの最大は、1024×1024 ピクセルです。1 回の塗りつぶしや画像の拡大ができる範囲（バウンディング ボックス）の最大は、1024×1024 です。バウンディング ボックスの右端・下端が、フレームバッファの外に出ないようにしてください。描画対象とソース画像の先頭アドレスと 1 行のサイズは 32 の倍数のバイト数にする必要があります。

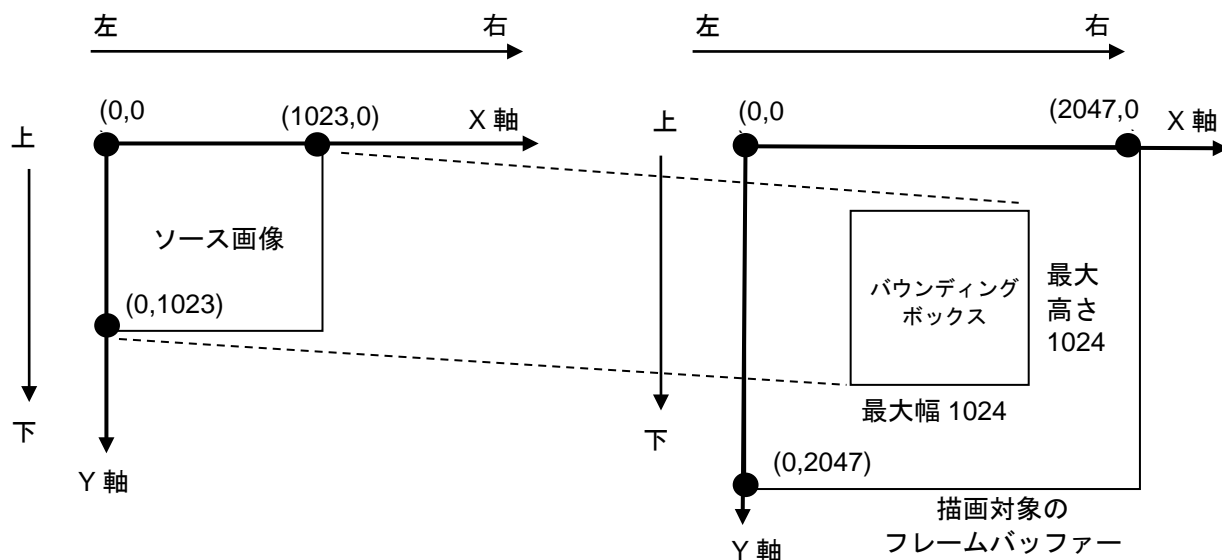


図 1-3 描画対象とソース画像の最大サイズ

表1.2 対応している描画対象のピクセルフォーマット

	XRGB 8888	ARGB 8888	RGB 565	ARGB 4444	A8
H/W レンダリング	○	○	○	○	○
S/W レンダリング	○	○	○	○	×
行列、拡大縮小、ブレンド	○	○	○	○	○
画像描画	○	○	○	○	○
DrawImageChild	○	○	○	○	○
アルファのみ画像	○	○	○	○	○
矩形塗りつぶし	○	○	○	○	×
(参考) ピクセルのバイト数	4	4	2	2	1

○=使用可能、×=使用不可

表1.3 対応している画像のピクセルフォーマットと描画対象のピクセルフォーマットの組み合わせ

描画対象 ソース画像	XRGB 8888	ARGB 8888	RGB 565	ARGB 4444	A8
JPEG	○	○	○	○	×
PNG	○	○	○	○	×
XRGB8888	○	○	○	○	×
ARGB8888	○	○	○	○	×
RGB565	○	○	○	○	×
ARGB4444	○	○	○	○	×
CLUT8	(×)	(×)	(×)	(×)	×
CLUT4	(×)	(×)	(×)	(×)	×
CLUT1	(×)	(×)	(×)	(×)	×
A8	○	○	○	○	○

○=使用可能、×=使用不可、(×)=使用不可・ハードウェア的には可能

上記の JPEG は、画像描画関数 (R_GRAPHICS_DrawImage) の引数に JPEG データを指定した場合です。対応している JPEG の形式は、表 1.4 を参照してください。ImagePackager ツールを使って JPEG ファイルや PNG ファイルを Raw 形式 (XRGB8888 など) に変換した場合は、そのピクセルフォーマットの列を見てください。

表 1.4 対応している JPEG の形式

伸張モジュール	内蔵 JPEG コーデックユニット (JCU)
JPEG 対応規格	ベースライン
JPEG 内ピクセルフォーマット	YCbCr420 (H=2:1:1,V=2:1:1) YCbCr422 (H=2:1:1,V=1:1:1) YCbCr444 (H=1:1:1,V=1:1:1) YCbCr411 (H=4:1:1,V=1:1:1)

表 1.5 PNG 形式画像

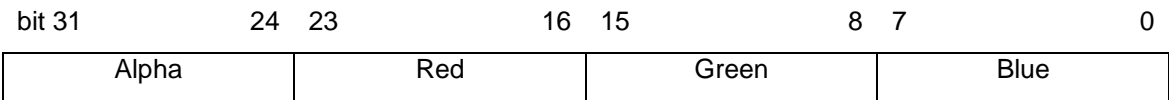
伸張モジュール	LibPNG, zlib
---------	--------------

以下に、ピクセルフォーマットの詳細を示します。RZ/A 版の RGA は、リトルエンディアンです。たとえば、XRGB8888 の Red は、ピクセルの先頭アドレス+2のバイトの位置になります。

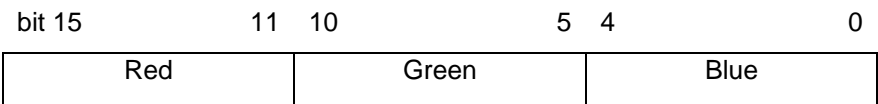
XRGB8888

bit 31	24	23	16	15	8	7	0
0	Red		Green		Blue		

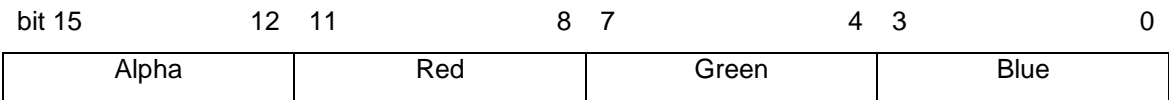
ARGB8888



RGB565



ARGB4444



A8

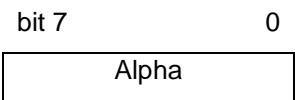
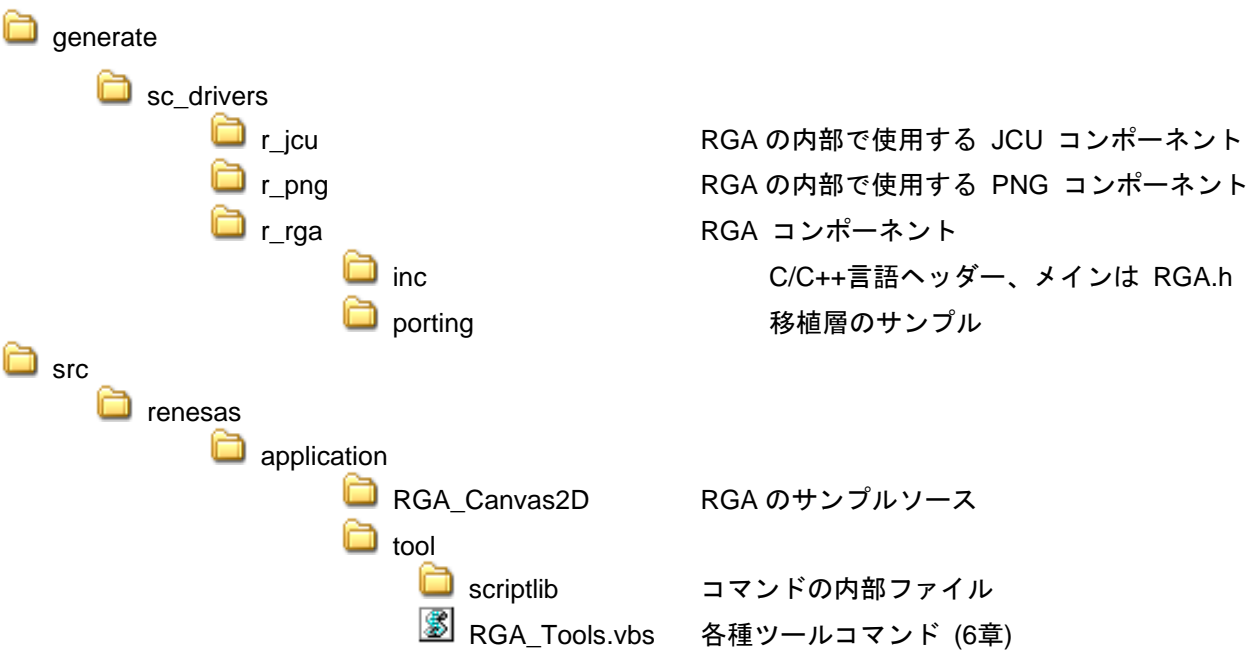


図 1-4 ピクセル フォーマットの構造

2. ファイル構成



既存のプロジェクトフォルダーに RGA のライブラリをインポートする方法は、RZ/A2M Graphics RGA Package リリースノート (R01AN4606) の「パッケージ内コンポーネントの使用法」の章を参照してください。

アプリケーションプログラムが RGA を使うときは、RGA.h を#include してください。

メモリーが不足するときは、フレームバッファーを表示用の分だけに減らし、ワークバッファーBをサイズ0にしてください。また、環境によっては、スタック サイズを増やす必要があります。 割込みのスタック サイズもチェックしてください。

r_memory_map.c に設定しているメモリーマップが、お使いの環境の MMU の設定に合っていることをご確認ください。

浮動小数に関するコンパイラーのオプションは、vfp (double レジスタを使う) に設定してください。

表 2.1 主なヘッダーファイル一覧 (\$\{RGA\}\%src\%drivers\%RGA\%inc フォルダー)

ファイル名	内容
RGA.h	RGA メイン
RGA_API.h	RGA サブ : API 関係。RGA.h からインクルードされます
use_config_RGA.h	RGA サブ : 設定関係。RGA.h からインクルードされます

3. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表 3.1 動作確認条件

項目	内容
使用 MCU	RZ/A2M
動作周波数（注）	CPU クロック（I ϕ ）：528MHz 画像処理クロック（G ϕ ）：264MHz 内部バスクロック（B ϕ ）：132MHz 周辺クロック 1（P1 ϕ ）：66MHz 周辺クロック 0（P0 ϕ ）：33MHz QSPI0_SPCLK：66MHz CKIO：132MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（1.8/3.3V 切替 I/O（PVcc_SPI））：3.3V 電源電圧（内部）：1.2V
統合開発環境	e2studio 2020-07
C コンパイラ	GNU Arm Embedded Toolchain 6-2017-q2-update コンパイラオプション（ディレクトリパスの追加は除く） Release: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug: -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0
動作モード	ブートモード 3（シリアルフラッシュブート 3.3V 品）
ターミナルソフトの通信設定	<ul style="list-style-type: none"> 通信速度：115200bps データ長：8 ビット パリティ：なし ストップビット長：1 ビット フロー制御：なし
使用ボード	RZ/A2M CPUボード RTK7921053C00000BE RZ/A2M SUBボード RTK79210XXB00000BE
使用デバイス （ボード上で使用する機能）	<ul style="list-style-type: none"> シリアルフラッシュメモリ（SPI マルチ I/O バス空間に接続） メーカー名：Macronix 社、型名：MX25L51245GXD RL78/G1C（USB 通信とシリアル通信を変換し、ホスト PC との通信に使用） LED1

【注】 クロックモード 1（EXTAL 端子からの 24MHz のクロック入力）で使用時の動作周波数です。

4. ソフトウェア説明

4.1 動作概要

4.1.1 バッファ描画モード

(1) フローチャート

図4-1にアプリケーションが定義したフレームバッファに描画する場合のフローチャートを示します。
実際に動かすときの手順は、別紙「RGA チュートリアル」を参照してください。

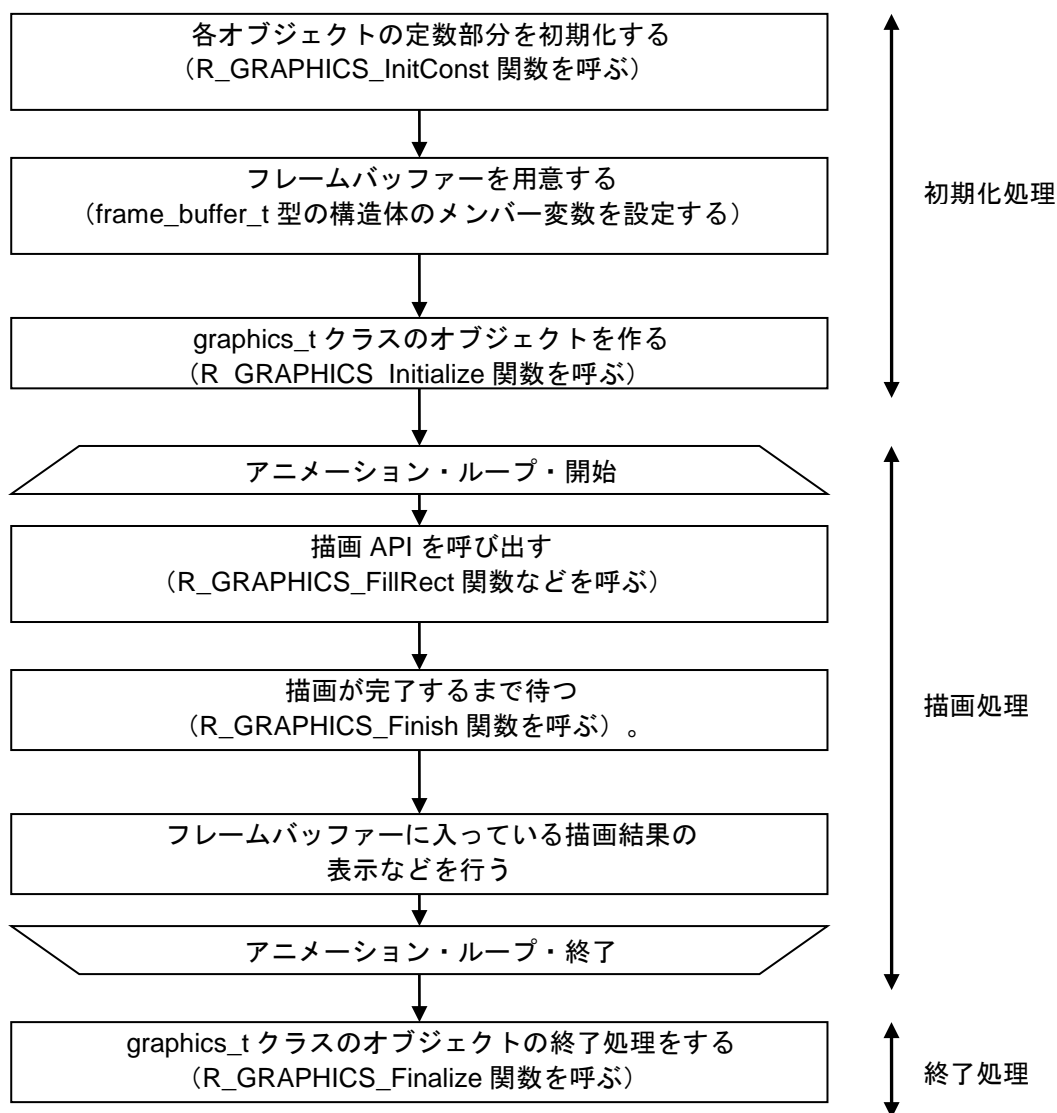


図4-1 アプリケーション定義のバッファに描画する場合 - フローチャート

(2) シーケンス図

アプリケーションが用意したフレームバッファに描画する場合における、ソフトウェア（アプリケーションとライブラリ）とハードウェア（描画ハードウェアと表示ハードウェア）の動作タイミングを下図に示します。ただし、ソフトウェア描画する場合、描画が完了してから描画 API から返ります。

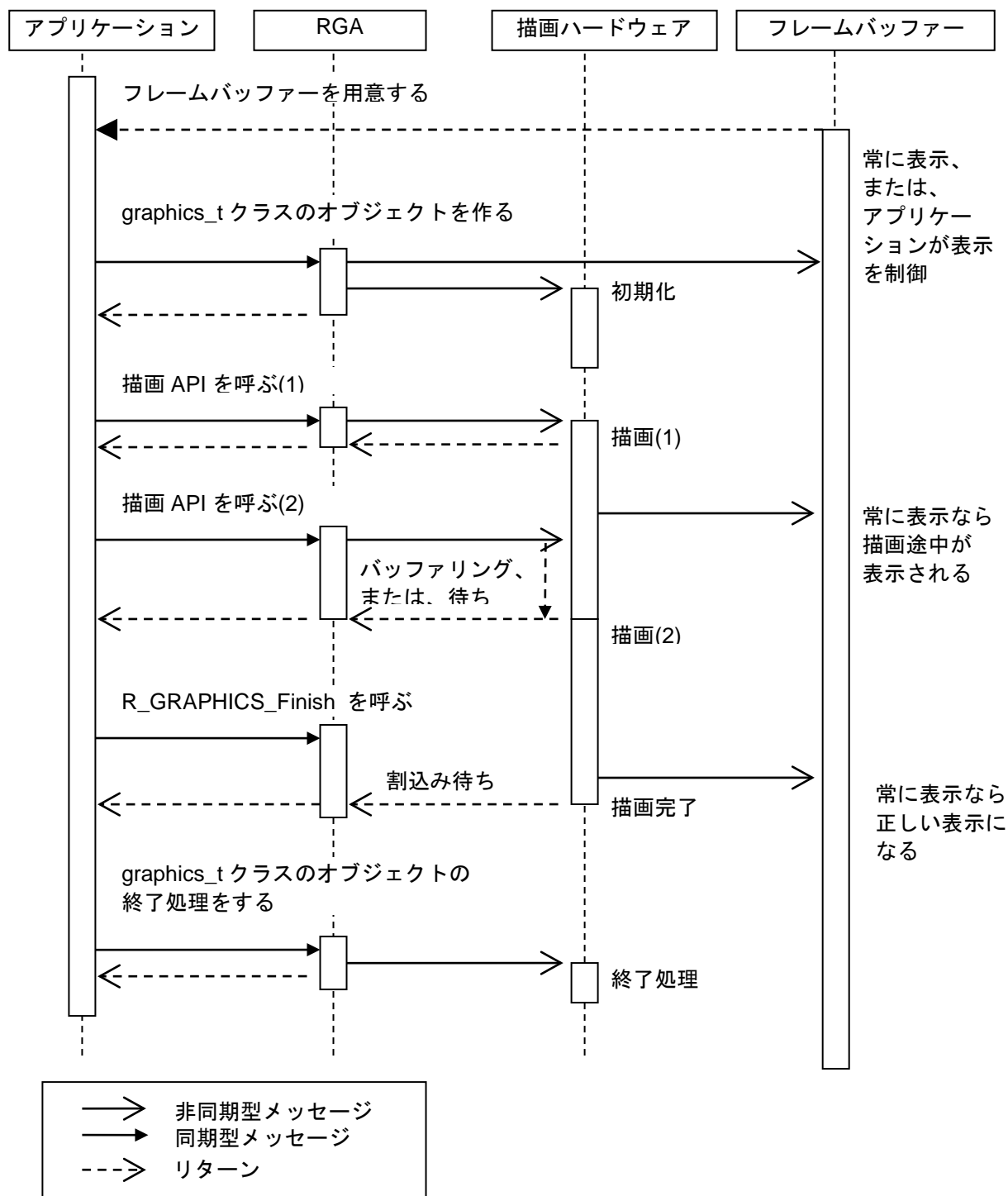


図4-2 動作タイミング

4.1.2 表示描画モード

(1) フローチャート

図4-3に表示画面に描画する場合のフローチャートを示します。

実際に動かすときの手順は、別紙「RGA チュートリアル」を参照してください。

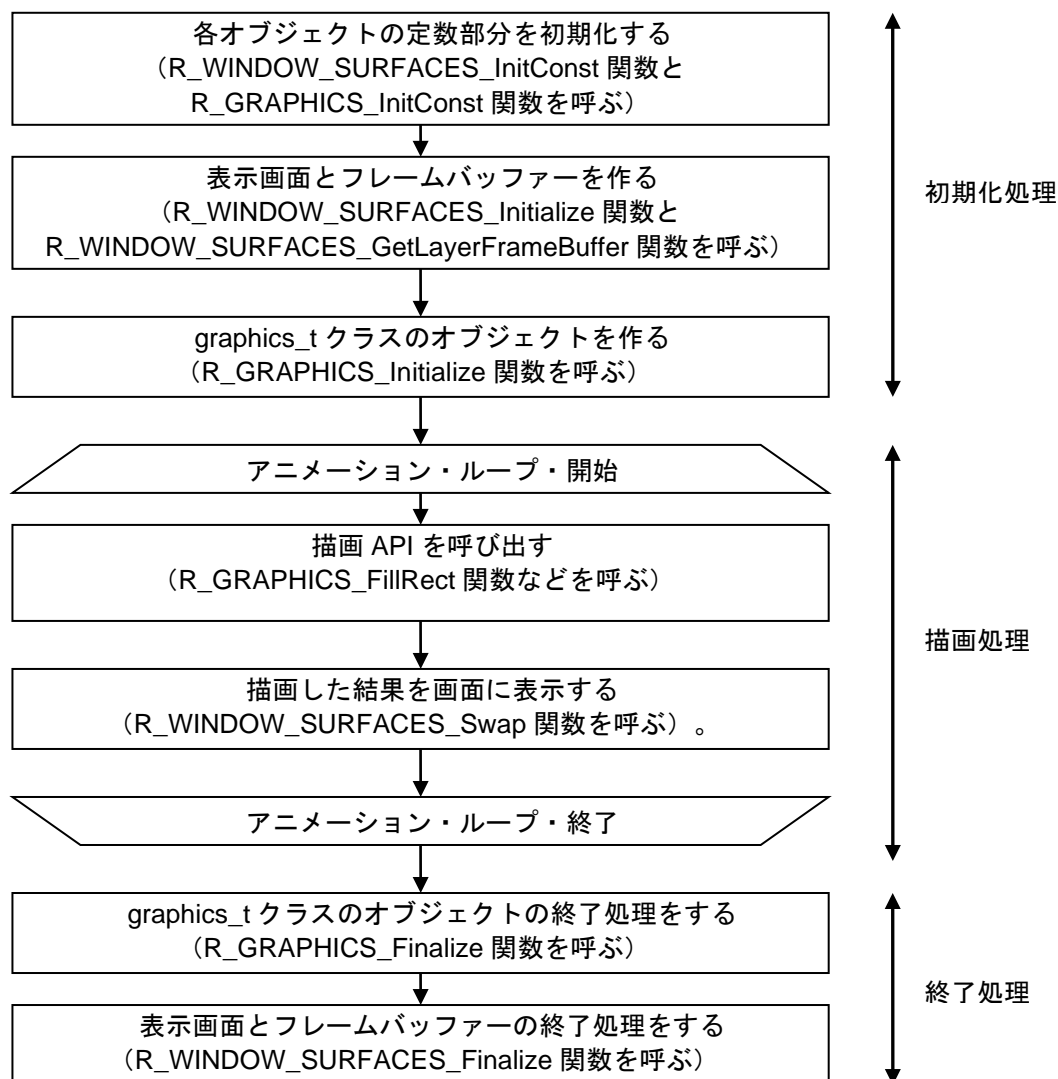


図4-3 表示画面に描画する場合 - フローチャート処理

(2) シーケンス図

RGA の WindowSurfaces ライブラリが用意したフレームバッファに描画するケースにおける、ソフトウェア（アプリケーションとライブラリ）とハードウェア（描画ハードウェアと表示ハードウェア）の動作タイミングを図4-4に示します。ただし、ソフトウェア描画する条件の場合は、描画が完了してから描画 API から返ります。

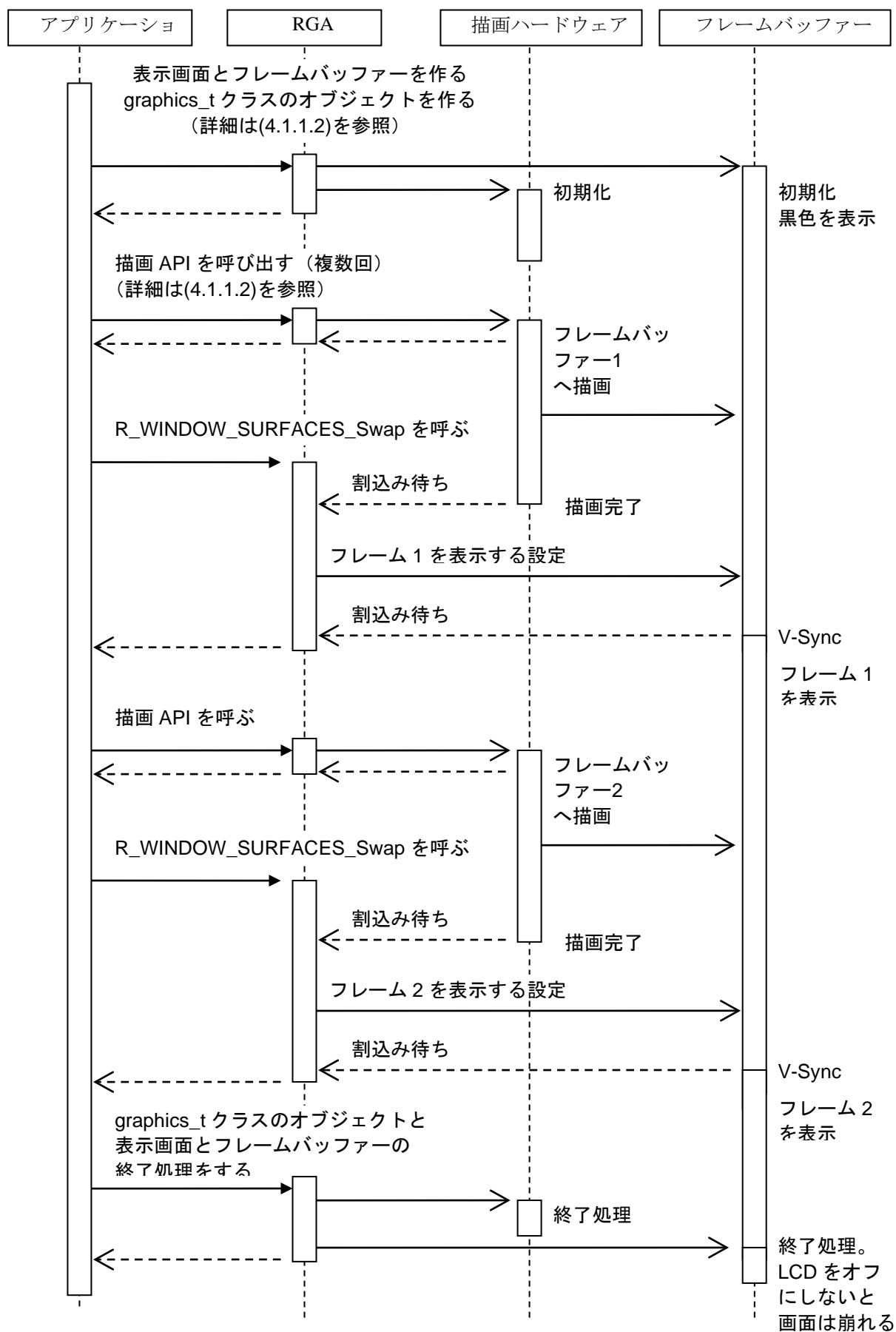


図4-4 動作タイミング

4.2 定数／型／クラス／関数

プロジェクトに付属の HTML ファイルを参照してください。

4.2.1 RZ/A1 用 RGA との比較

RZ/A2 用 RGA が定義する型や関数は、RZ/A1H シリーズ用 RGA の型や関数と同じです。ただし、すべての型や関数は、サポートしていません。サポートしている型の表を示します。C++ のクラスはサポートしていません。 --- が書かれている行はサポートしていません。

型／クラス

RGA for RZ/A2M	RGA for RZ/A1H
frame_buffer_t ^{*1}	frame_buffer_t
graphics_t	graphics_t
graphics_config_t ^{*1}	graphics_config_t
graphics_quality_flag_t	graphics_quality_flag_t
window_surfaces_t	window_surfaces_t
window_surfaces_config_t ^{*1}	window_surfaces_config_t
layer_attributes_t ^{*1}	layer_attributes_t
---	access_t
---	video_input_t
---	video_input_config_t
byte_per_pixel_t	byte_per_pixel_t
pixel_format_t	pixel_format_t
frame_buffer_delegate_t	frame_buffer_delegate_t
---	v_sync_t
---	vram_stack_t
---	vram_stack_config_t
---	vram_ex_stack_t
graphics_image_t	graphics_image_t
graphics_image_properties_t	graphics_image_properties_t
graphics_composite_operation_t	graphics_composite_operation_t
graphics_status_t	graphics_status_t
graphics_matrix_float_t	graphics_matrix_float_t
repetition_t	repetition_t
r8g8b8a8_t	r8g8b8a8_t
---	background_format_t
graphics_pattern_t	graphics_pattern_t
animation_timing_function_t	animation_timing_function_t
graphics_jpeg_decoder_t	graphics_jpeg_decoder_t
---	graphics_async_status_t

^{*1} 仕様に変更があります。

関数／メソッド

RGA for RZ/A2M	RGA for RZ/A1H
R_GRAPHICS_InitConst	R_GRAPHICS_InitConst
R_GRAPHICS_Initialize	R_GRAPHICS_Initialize
R_GRAPHICS_Finalize	R_GRAPHICS_Finalize
R_GRAPHICS_CONFIG_SetEmpty	---
R_GRAPHICS_SetFrameBuffer	R_GRAPHICS_SetFrameBuffer
R_GRAPHICS_GetFrameBuffer	R_GRAPHICS_GetFrameBuffer
R_GRAPHICS_Start	---
R_GRAPHICS_StartPreviousFrame	---
R_GRAPHICS_FinishPreviousFrame	---
R_GRAPHICS_Finish	R_GRAPHICS_Finish
---	R_GRAPHICS_FinishStart
R_GRAPHICS_GetHasFramePipeline	---
---	R_GRAPHICS_GetAsyncStatus
---	R_GRAPHICS_OnInterrupting
R_GRAPHICS_Save	R_GRAPHICS_Save
R_GRAPHICS_Restore	R_GRAPHICS_Restore
R_GRAPHICS_ResetMatrix	R_GRAPHICS_ResetMatrix
R_GRAPHICS_SetMatrix_2x3	R_GRAPHICS_SetMatrix_2x3
R_GRAPHICS_SetMatrix_3x3	R_GRAPHICS_SetMatrix_3x3
R_GRAPHICS_GetMatrix_3x3	R_GRAPHICS_GetMatrix_3x3
R_GRAPHICS_TranslateMatrixI	R_GRAPHICS_TranslateMatrixI
R_GRAPHICS_TranslateMatrix	R_GRAPHICS_TranslateMatrix
R_GRAPHICS_ScaleMatrix	R_GRAPHICS_ScaleMatrix
R_GRAPHICS_RotateMatrixDegree	R_GRAPHICS_RotateMatrixDegree
R_GRAPHICS_ShearMatrix	R_GRAPHICS_ShearMatrix
R_GRAPHICS_TransformMatrix	R_GRAPHICS_TransformMatrix
R_GRAPHICS_MultiplyMatrix	R_GRAPHICS_MultiplyMatrix
R_GRAPHICS_GetProjectiveMatrix	R_GRAPHICS_GetProjectiveMatrix
R_GRAPHICS_SetBackgroundColor	R_GRAPHICS_SetBackgroundColor
R_GRAPHICS_GetBackgroundColor	R_GRAPHICS_GetBackgroundColor
R_GRAPHICS_GetClearColor	R_GRAPHICS_GetClearColor
R_GRAPHICS_Clear	R_GRAPHICS_Clear
R_GRAPHICS_DrawImage	R_GRAPHICS_DrawImage
R_GRAPHICS_DrawImageResized	R_GRAPHICS_DrawImageResized
R_GRAPHICS_DrawImageChild	R_GRAPHICS_DrawImageChild
R_GRAPHICS_FillRect	R_GRAPHICS_FillRect
R_GRAPHICS_SetFillColor	R_GRAPHICS_SetFillColor
R_GRAPHICS_SetFillPattern	R_GRAPHICS_SetFillPattern
R_GRAPHICS_BeginPath	R_GRAPHICS_BeginPath
R_GRAPHICS_Rect	R_GRAPHICS_Rect
R_GRAPHICS_Clip	R_GRAPHICS_Clip
R_GRAPHICS_SetGlobalAlpha	R_GRAPHICS_SetGlobalAlpha
R_GRAPHICS_GetGlobalAlpha	R_GRAPHICS_GetGlobalAlpha
R_GRAPHICS_SetGlobalCompositeOperation	R_GRAPHICS_SetGlobalCompositeOperation
R_GRAPHICS_GetGlobalCompositeOperation	R_GRAPHICS_GetGlobalCompositeOperation
---	R_GRAPHICS_STATIC_SetOnInitialize
---	R_GRAPHICS_STATIC_SetOnFinalize

R_GRAPHICS_OnInitialize *1	R_GRAPHICS_STATIC_OnInitializeDefault
R_GRAPHICS_OnFinalize *1	R_GRAPHICS_STATIC_OnFinalizeDefault
R_GRAPHICS_SetQualityFlags	R_GRAPHICS_SetQualityFlags
R_GRAPHICS_GetQualityFlags	R_GRAPHICS_GetQualityFlags
R_GRAPHICS_SetStrokeColor	R_GRAPHICS_SetStrokeColor
R_GRAPHICS_StrokeRect	R_GRAPHICS_StrokeRect
R_GRAPHICS_BeginSoftwareRendering	R_GRAPHICS_BeginSoftwareRendering
R_GRAPHICS_EndSoftwareRendering	R_GRAPHICS_EndSoftwareRendering
R_GRAPHICS_EndRenderingInFin	R_GRAPHICS_EndRenderingInFin
---	R_GRAPHICS_BeginSoftwareRendering2
---	R_GRAPHICS_BeginSoftwareRenderingA
R_GRAPHICS_IMAGE_InitByShareFrameBuffer	R_GRAPHICS_IMAGE_InitByShareFrameBuffer
R_GRAPHICS_IMAGE_InitByShareFrameBufferEx	---
R_GRAPHICS_IMAGE_InitR8G8B8A8	R_GRAPHICS_IMAGE_InitR8G8B8A8
R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8	R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8
R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8	R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8
R_GRAPHICS_IMAGE_GetProperties	R_GRAPHICS_IMAGE_GetProperties
R_GRAPHICS_PATTERN_Initialize	R_GRAPHICS_PATTERN_Initialize
R_WINDOW_SURFACES_InitConst	R_WINDOW_SURFACES_InitConst
R_WINDOW_SURFACES_Initialize	R_WINDOW_SURFACES_Initialize
R_WINDOW_SURFACES_Finalize	R_WINDOW_SURFACES_Finalize
---	R_WINDOW_SURFACES_Dispose
R_WINDOW_SURFACES_CONFIG_SetEmpty	---
R_WINDOW_SURFACES_GetLayerFrameBuffer	R_WINDOW_SURFACES_GetLayerFrameBuffer
R_WINDOW_SURFACES_GetLayerCount	R_WINDOW_SURFACES_GetLayerCount
R_WINDOW_SURFACES_SwapBuffers	R_WINDOW_SURFACES_SwapBuffers
---	R_WINDOW_SURFACES_SwapBuffersStart
---	R_WINDOW_SURFACES_WaitForVSync
R_WINDOW_SURFACES_DoMessageLoop	R_WINDOW_SURFACES_DoMessageLoop
R_WINDOW_SURFACES_SetLayerAttributes *1	R_WINDOW_SURFACES_AccessLayerAttributes
R_LAYER_ATTRIBUTES_SetEmpty	---
R_WINDOW_SURFACES_AllocOffscreenStack	R_WINDOW_SURFACES_AllocOffscreenStack
R_WINDOW_SURFACES_FreeOffscreenStack	R_WINDOW_SURFACES_FreeOffscreenStack
---	R_VIDEO_INPUT_InitConst
---	R_VIDEO_INPUT_Initialize
---	R_VIDEO_INPUT_Finalize
R_RGA_BitPerPixelType_To_BytePerPixelType	R_RGA_BitPerPixelType_To_BytePerPixelType
R_RGA_BytePerPixelType_To_BitPerPixelType	R_RGA_BytePerPixelType_To_BitPerPixelType
R_BYTE_PER_PIXEL_IsInteger	R_BYTE_PER_PIXEL_IsInteger
---	R_V_SYNC_Initialize
---	R_V_SYNC_Finalize
---	R_V_SYNC_Wait
---	R_V_SYNC_WaitStart
---	R_V_SYNC_OnInterrupting
---	R_V_SYNC_GetAsyncStatus
---	R_VRAM_STACK_Initialize
---	R_VRAM_STACK_AllocateMemory
---	R_VRAM_STACK_FreeMemory
---	R_VRAM_EX_STACK_Initialize
---	R_VRAM_EX_STACK_Alloc
---	R_VRAM_EX_STACK_Free
R_Get_AnimationTimingFunction	R_Get_AnimationTimingFunction

R_ANIMATION_TIMING_FUNCTION_GetValue	R_ANIMATION_TIMING_FUNCTION_GetValue
R_RGA_Get_R8G8B8A8	R_RGA_Get_R8G8B8A8
---	R_RGA_CalcWorkBufferSize
R_RGA_CalcWorkBufferB_Size	R_RGA_CalcWorkBufferB_Size
---	(RGA C++ API)
r_co_function_t	---

*1 仕様に変更があります。

4.2.2 エラーコード

エラーが発生すると、R_ERROR_Set 関数が呼ばれるときに errnum_t 型の引数の値が 0 以外になります。
また、errno_t 型の値を返す関数が、0 以外の値を返します。

エラーが発生したコードは、ソースファイルをシンボルで検索するか、R_ERROR_Set 関数の中で
errno_t 型の引数の値で条件ブレイクすることで見つかります。

プロジェクトに付属の HTML ファイルでエラーコードの数値を検索することができます。

4.2.3 byte_per_pixel_t の図

1 ピクセルあたりのバイト数の型です。

1 ピクセルが 1 バイト未満の場合（BitPerPixel < 8 の場合）は、1 ピクセルあたりのビット数をシフトした値にします。

1 ピクセルが 1 バイト以上の場合：

	15	8	7	0
0	0	バイト数		

1 ピクセルが 1 バイト未満の場合：

	15	8	7	0
0	ビット数			0

4.3 移植ガイド

RGA が動作する環境から、RTOS や立ち上げプログラムを変更するときは、移植層のコードを変更する必要があります。



移植層は、以下のファイルで定義されています。

- r_drw_pl.c

RGA は、以下の移植層の関数をコールバックします。関数の定義内容を RTOS や立ち上げプログラムの種類に応じて変更してください。

種類	関数名	概要
初期化	R_DRW_OnInitialize	DRW を初期化するときに呼ばれる関数
	R_DRW_OnFinalize	DRW の終了処理をするときに呼ばれる関数
割込み	R_DRW_EnableInterrupt	DRW の割込みを許可します
	R_DRW_DisableInterrupt	DRW の割込みを禁止します
RTOS スレ ッド	R_DRW_WaitForInterruptEvent	DRW の割込み発生を待ちます
	R_DRW_SetInterruptEvent	DRW の割込み発生をスレッドに通知します
	R_DRW_CheckThread	ロックしているスレッドかどうかをチェックします
メモ リー	R_DRW_FlushCache	CPU の L1 キャッシュをフラッシュします
	R_DRW_ToPhysicalAddress	物理アドレスに変換します
	R_DRW_ToCachedAddress	キャッシュ領域のアドレスに変換します
	R_DRW_ToUncachedAddress	非キャッシュ領域のアドレスに変換します

移植層は、以下の API 関数を呼び出してください。

関数名	概要
R_DRW_OnInterrupting	DRW の割込みが発生したときの処理を実行します

関数の詳細は、プロジェクトに付属の HTML ファイルを参照してください。HTML ファイルに書かれていない図については、本書に記述しています。

4.4 補足

4.4.1 Canvas 2D との対応表、H/W アクセラレーション対応表

H/W の列の記号は、○=H/W を使用、×=H/W を非使用、－=H/W 対象外です。

Canvas2D API	RGA - C 言語 API	H/W
CanvasRenderingContext2D interface.	graphics_t	－
getContext()	R_GRAPHICS_Initialize	－
context.canvas	-	
CSS currentColor	-	
.save ()	R_GRAPHICS_Save	－
.restore()	R_GRAPHICS_Restore	－
.scale(x, y)	R_GRAPHICS_ScaleMatrix	○
.rotate(angle)	R_GRAPHICS_RotateMatrixRadian	○
.translate(x, y)	R_GRAPHICS_TranslateMatrix	○
.transform(a, b, c, d, e, f)	R_GRAPHICS_TransformMatrix	○
.setTransform(a, b, c, d, e, f)	R_GRAPHICS_SetMatrix_2x3	○
.lineWidth	-	
.lineCap, .lineJoin, .miterLimit	-	
	-	
.moveTo()	-	
.closePath()	-	
.lineTo()	-	
.quadraticCurveTo()	-	
.bezierCurveTo()	-	
.arcTo()	-	
.arc()	-	
.rect()	R_GRAPHICS_Rect	○
.fillStyle 単色	R_GRAPHICS_SetFillColor	○
.fillStyle グラデーション	-	
.fillStyle パターン	R_GRAPHICS_SetFillPattern	○
.strokeStyle 単色	-	
.strokeStyle グラデーション	-	
.strokeStyle パターン	-	
.createLinearGradient()	-	
.createRadialGradient()	-	
CanvasGradient.addColorStop()	-	

Canvas2D API	RGA - C 言語 API	H/W
.createPattern()	R_GRAPHICS_PATTERN_Initialize	—
.beginPath()	R_GRAPHICS_BeginPath	—
.fill()	-	
.stroke()	-	
.drawSystemFocusRing()	-	
.drawCustomFocusRing()	-	
.scrollPathIntoView()	-	
.clip()	R_GRAPHICS_Clip (1 つの長方形のみ)	○
.isPointInPath()	-	
.clearRect()	R_GRAPHICS_Clear	○
.fillRect()	R_GRAPHICS_FillRect	○
.strokeRect()	-	
	-	
.drawImage(dx, dy)	R_GRAPHICS_DrawImage	○
.drawImage(dx, dy, dw, dh)	R_GRAPHICS_DrawImageResized	○
.drawImage(sx, sy, sw, sh, dx, dy, dw, dh)	R_GRAPHICS_DrawImageChild	○
.createImageData(Width, Height)	R_GRAPHICS_IMAGE_-InitR8G8B8A8	—
.createImageData(ImageData)	R_GRAPHICS_IMAGE_-InitSameSizeR8G8B8A8	—
ImageData.width	R_GRAPHICS_IMAGE_GetProperties	—
ImageData.height	R_GRAPHICS_IMAGE_GetProperties	—
ImageData.data	R_GRAPHICS_IMAGE_GetProperties	—
.getImageData()	R_GRAPHICS_IMAGE_-InitCopyFrameBufferR8G8B8A8	×
.putImageData()	R_GRAPHICS_DrawImage R_GRAPHICS_DrawImageChild	×
.globalAlpha	R_GRAPHICS_SetGlobalAlpha	○
.globalCompositeOperation	R_GRAPHICS_-SetGlobalCompositeOperation	○

4.4.2 初期化関数の内部の動き

RGA の初期化に関する関数コールツリーを示します。

```
R_GRAPHICS_Initialize
  R_GRAPHICS_OnInitialize
    R_GRAPHICS_H_Initialize // ハードウェアの初期化
      R_DRW_OnInitialize
        R_GRAPHICS_OnInitialized
```


4.4.3 画像の形式の識別

R_GRAPHICS_DrawImage、R_GRAPHICS_DrawImageResized、R_GRAPHICS_DrawImageChild 関数の引数に指定する graphics_image_t 型の構造体の先頭の数バイトは、画像の形式の識別に使われます。つまり、R_GRAPHICS_DrawImage 関数の graphics_image_t 型の引数に、直接 JPEG ファイルや PNG ファイルのデータを指定することができます。

画像の形式	先頭の数バイト	補足
JPEG	0xFF 0xD8	SOI セグメント
PNG	0x89 0x50 0x4E 0x47	PNG ヘッダー
Raw	その他	graphics_image_t 型の構造体 + Raw データ

4.4.4 デフォルト可能フラグ

デフォルト可能フラグは、論理型の配列変数を設定するときに、それぞれの配列要素に対して、オンに設定／オフに設定／設定しない（無変更） を選べる型です。 オンに設定するシンボルと、オフに設定するシンボルが定義され、それらのシンボルを使わないときは、設定を変更しません。

デフォルト可能フラグを格納する変数の型は、32 ビット整数型です。 下位 16 ビットを「オン」に設定するフラグ、上位 16 ビットを「オフ」に設定するフラグと定義します。 上位 16 ビットの構成要素は、下位 16 ビットの構成要素と同じです。

「オフ」に設定するフラグ (上位 16 ビット)	「オン」に設定するフラグ (下位 16 ビット)
-----------------------------	-----------------------------

```
typedef BitField DefaultableFlagsType; /* Flags of DefaultableFlagType */
enum DefaultableFlagType {
    /* Set to "ON" */
    ENABLE_SAMPLE_FLAG_A    = 0x0001,
    ENABLE_SAMPLE_FLAG_B    = 0x0002,
    ENABLE_SAMPLE_FLAG_C    = 0x0004,

    /* Set to "OFF" */
    DISABLE_SAMPLE_FLAG_A   = ENABLE_SAMPLE_FLAG_A << 16,
    DISABLE_SAMPLE_FLAG_B   = ENABLE_SAMPLE_FLAG_B << 16,
    DISABLE_SAMPLE_FLAG_C   = ENABLE_SAMPLE_FLAG_C << 16,
};
```

フラグを設定する関数（下記の SampleClass_setDefaultableFlags ）の引数がデフォルト可能フラグの場合、「オン」または「オフ」に設定するフラグについてのみ | で接続します。「オン」にも「オフ」にも設定しなかったフラグに関しては、設定を変更しません。 初期化関数の場合は、デフォルトの値が採用されます。

```
errnum_t main()
{
    DefaultableFlagsType flags;

    e= SampleClass_setDefaultableFlags( object,
        ENABLE_SAMPLE_FLAG_A | DISABLE_SAMPLE_FLAG_B ); IF(e)goto fin;
    /* SAMPLE_FLAG_C is not modified. */

    e= SampleClass_getDefaultableFlags( object, &flags ); IF(e)goto fin;
    if ( flags & ENABLE_SAMPLE_FLAG_A ) { ... }
    if ( flags & DISABLE_SAMPLE_FLAG_B ) { ... }
    if ( flags & ENABLE_SAMPLE_FLAG_C ) { ... }
}

errnum_t SampleClass_setDefaultableFlags( SampleClass* self,
DefaultableFlagsType Flags )
{
    self->Flags = self->Flags | ( Flags & 0x0000FFFF );
    self->Flags = self->Flags & ~( Flags >> 16 );
}

errnum_t SampleClass_getDefaultableFlags( SampleClass* self,
DefaultableFlagsType* out_Flags )
{
    BitField flags = ( self->Flags & 0x0000FFFF );
    *out_Flags = flags | ~( flags << 16 );
}
```

現在のフラグを取得する関数（上記の SampleClass_getDefaultableFlags ）の引数がデフォルト可能フラグの場合、 取得できる値の上位 16 ビットは、下位 16 ビットの反転した値になります。 これにより、オン

に設定するシンボルでも、オフに設定するシンボルでも、どちらでも判定文に使うことができます。現在のフラグを保持している内部変数は、内部の仕様なので、上位 16 ビットが無効である仕様でも構いません。

下位 16 ビットの反転 (上位 16 ビット)	現在のフラグ (下位 16 ビット)
-----------------------------	-----------------------

4.4.5 内部変数を定数で初期化する関数について (*_InitConst 関数)

終了処理関数 (*_Finalize 関数) が存在する C 言語のクラスは、最初に (*_Initialize 関数を呼び出す前に) *_InitConst 関数を呼び出す必要があります。これは、初期化される前に、別のオブジェクトからエラーが発生したときに、終了処理関数を呼び出しても例外が発生しないようにするためです。関数の中で初めてエラーが発生する可能性がある関数を呼び出す前に、関数の中だけに存在する (関数の中で生成と削除が行なわれる) すべてのオブジェクトに対して、最初にまとめて *_InitConst 関数を呼び出してください。

*_InitConst 関数を呼び出しても、*_Initialize 関数を呼び出すまで、多くの関数 (メソッド) は使えません。

本ライブラリの C++言語 API の場合、*_InitConst 関数がコンストラクターに対応します。*_Initialize 関数がオブジェクトを生成する関数 (例: R_RGA_New_Canvas2D_ContextClass 関数) に対応します。終了処理関数は destroy メンバー関数に対応します。たとえば、変数宣言によってコンストラクターが呼ばれた後、オブジェクトを生成する関数が呼ばれる前にエラーが発生したときに、destroy メンバー関数を呼び出しても例外は発生しません。

4.4.6 終了処理について (*_Finalize 関数)

名前の後半が*_Finalize の関数は、オブジェクトの終了処理をします。終了処理とは、ファイルのクローズ処理や、C++言語のデストラクターが行う処理、Java 言語の finally 節から呼び出す処理などです。

概 要	終了処理をします。	
宣 言	errnum_t *_Finalize(type* self, errnum_t e);	
引 数	type* self	終了処理の対象となるオブジェクト
	errnum_t e	これまでに発生したエラーコード。 関数によっては、エラーが発生していたかどうかでロールバック処理を行うかどうかの違いがあります。
リターン値	エラーコード または e、0=成功かつ e=0。 引数 e が 0 以外なら、その引数 e が返ります。 引数 e が 0 なら、終了処理のエラーコードが返ります。	

引数 e には、次のようにこれまでに発生したエラーコードを渡します。

```
errnum_t Func()  
{  
    errnum_t e;  
  
    CLASS_InitConst( &sample );  
    e= CLASS_Initialize( &sample ); IF(e){goto fin;}  
  
    e=0;  
fin:  
    e= CLASS_Finalize( &sample, e );  
    return e;  
}
```

終了処理が成功すると未初期化状態になります。

終了処理の内部でエラーが発生した場合、0 以外のエラーコードが返り、次のいずれかの状態になります。

エラー後の状態	説明、期待される動作
未初期化状態	<p>対象となるオブジェクトは、削除することができます。</p> <p>この状態になる場合、内部でエラーが発生しても、すべての内部オブジェクトについて終了処理が行われます。</p> <p>内部でエラーが発生しても、内部オブジェクトが未初期化状態になれば、内部オブジェクトは残らず、エラー復帰できます。</p> <p>元の状態に戻った内部オブジェクトは、後でリソース管理オブジェクトによって削除されるか、リセットされるまで残り続けます。もしくはロックされたままになります。元の状態に戻った内部オブジェクトに関して、何らかのコールバック関数が呼ばれることがあります。</p> <p>終了処理を呼び出した後で、エラーについてエンドユーザーへの通知やログへの記録などを行ってください。</p>
リセット状態	<p>内部から R_OSPL_RaiseUnrecoverable 関数 が呼ばれ、システムのリセットやプロセスの緊急終了処理などが行われます。その場合、終了処理関数から返りません。</p>
元の状態	<p>元の状態に戻ったオブジェクトは、削除できません。</p> <p>元の状態に戻ることがよくある終了処理は、finally 節（上記コードの fin:）から呼び出さないようにして、元に戻れるようにするとよいでしょう。</p> <p>finally 節から呼び出すと、オブジェクトは残り続けます。なるべくそうならないように、終了処理関数を呼び出す前に、何らかの調整をしてください。残ったオブジェクトを統合していたオブジェクトは未初期化状態になることがあります。その場合の対処方法は、本表の「未初期化状態」の説明を参照してください。</p>

5. 関数／メソッド

5.1 graphics_t クラスのメンバー関数に相当する関数

5.1.1 一覧

章	関数名	概要
5.1.2	R_GRAPHICS_STATIC_GetVersion	RGA のバージョン番号を取得します。
5.1.3	R_GRAPHICS_PrintRegisters	レジスタを print 表示します。
5.1.4	R_GRAPHICS_Start	バッファリングされているグラフィックスのハードウェア描画を開始します。
5.1.5	R_GRAPHICS_FinishPreviousFrame	GPU は "frame_buffer_t :: draw_buffer_index" に描画を終了する。
5.1.6	R_GRAPHICS_StartPreviousFrame	前のフレームのハードウェア描画を開始します。
5.1.7	R_GRAPHICS_GetHasFramePipeline	フレームパイプラインモードのオン・オフを返します
5.1.8	R_GRAPHICS_BeginSoftwareRendering2	ソフトウェア レンダリングを開始することを、グラフィックス ライブラリに通知します。
5.1.9	R_GRAPHICS_BeginSoftwareRenderingA	ソフトウェア レンダリングを開始することを、グラフィックス ライブラリに通知します。
5.1.10	R_GRAPHICS_CONFIG_SetEmpty	すべてのメンバー変数の値を空を意味する値で初期化します。
5.1.11	R_GRAPHICS_InitConst	定数部分を初期化します
5.1.12	R_GRAPHICS_Initialize	グラフィックス描画コンテキスト オブジェクトを初期化します。
5.1.13	R_GRAPHICS_Finalize	グラフィックス描画コンテキスト オブジェクトの終了処理をします
5.1.14	R_GRAPHICS_SetFrameBuffer	描画対象を変更します。
5.1.15	R_GRAPHICS_GetFrameBuffer	描画対象を取得します。
5.1.16	R_GRAPHICS_Finish	描画が完了するまで待ちます。
5.1.17	R_GRAPHICS_Save	コンテキストの設定値を指定されたステータス構造体に退避します。
5.1.18	R_GRAPHICS_Restore	ステータス構造体の内容を、コンテキストに戻します。
5.1.19	R_GRAPHICS_ResetMatrix	行列演算関数の対象となる行列を単位行列にリセットします。
5.1.20	R_GRAPHICS_SetMatrix_2x3	行列の各要素を設定します。(2x3)
5.1.21	R_GRAPHICS_SetMatrix_3x3	行列の各要素を設定します。(3x3)
5.1.22	R_GRAPHICS_GetMatrix_3x3	行列の各要素を取得します。
5.1.23	R_GRAPHICS_TranslateMatrixI	現在の行列から移動させます。(整数型指定)
5.1.24	R_GRAPHICS_TranslateMatrix	現在の行列から移動させます。(浮動小数型指定)
5.1.25	R_GRAPHICS_ScaleMatrix	現在の行列から拡大縮小させます。
5.1.26	R_GRAPHICS_RotateMatrixDegree	現在の行列から回転させます。回転の中心は (0,0)
5.1.27	R_GRAPHICS_ShearMatrix	現在の行列からせん断変形させます。
5.1.28	R_GRAPHICS_TransformMatrix	現在の行列から指定した 2 行 3 列の行列を掛けます。
5.1.29	R_GRAPHICS_MultiplyMatrix	現在の行列から指定した 3 行 3 列の行列を掛けます。
5.1.30	R_GRAPHICS_GetProjectiveMatrix	任意の形状の四角形から、任意の形状の四角形へ、変形するような行列を取得します。
5.1.31	R_GRAPHICS_SetBackgroundColor	背景色を設定します。
5.1.32	R_GRAPHICS_GetBackgroundColor	背景色を取得します。

5.1.33	R_GRAPHICS_GetClearColor	R_GRAPHICS_Clear するときの色を取得します。
5.1.34	R_GRAPHICS_Clear	長方形の領域をクリアします。
5.1.35	R_GRAPHICS_DrawImage	画像を描画します。
5.1.36	R_GRAPHICS_DrawImageResized	画像を長方形の中に拡大縮小して描画します。
5.1.37	R_GRAPHICS_DrawImageChild	画像の一部を描画します。
5.1.38	R_GRAPHICS_FillRect	長方形を塗りつぶします。
5.1.39	R_GRAPHICS_SetFillColor	現在の塗りつぶしのペイントオブジェクトに、単色塗りつぶしをするようにして、その色を設定します。
5.1.40	R_GRAPHICS_SetFillPattern	現在の塗りつぶしのペイントオブジェクトに、パターンを設定します。
5.1.41	R_GRAPHICS_BeginPath	デフォルト パスの内容を空にリセットします。
5.1.42	R_GRAPHICS_Rect	デフォルト パスに、長方形を追加します。
5.1.43	R_GRAPHICS_Clip	現在のデフォルト パスの形状をクリッピング領域にします。
5.1.44	R_GRAPHICS_SetGlobalAlpha	すべての描画に対してかける 1 つのアルファ値（非透明度）を設定します。
5.1.45	R_GRAPHICS_GetGlobalAlpha	すべての描画に対してかける 1 つのアルファ値（非透明度）を取得します。
5.1.46	R_GRAPHICS_SetGlobalCompositeOperation	アルファブレンドするときの演算方法を設定します。
5.1.47	R_GRAPHICS_GetGlobalCompositeOperation	アルファブレンドするときの演算方法を取得します。
5.1.48	R_GRAPHICS_SetQualityFlags	描画品質を設定します。
5.1.49	R_GRAPHICS_GetQualityFlags	現在の描画品質を取得します。
5.1.50	R_GRAPHICS_SetStrokeColor	現在の境界線のペイントオブジェクトを、単色塗りつぶしをするようにして、その色を設定します。
5.1.51	R_GRAPHICS_StrokeRect	長方形の辺を描画します。
5.1.52	R_GRAPHICS_BeginSoftwareRendering	ソフトウェア レンダリングを開始することを、グラフィックスライブラリに通知します。
5.1.53	R_GRAPHICS_EndSoftwareRendering	ソフトウェア レンダリングが終了したことを、グラフィックスライブラリに通知します。
5.1.54	R_GRAPHICS_EndRenderingInFin	ソフトウェア レンダリングを行なう関数の最後から本関数を呼び出してください。

5.1.2 R_GRAPHICS_STATIC_GetVersion

概 要	RGA のバージョン番号を取得します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_STATIC_GetVersion(uint32_t* out_Version);	
補 足	*out_Version == 310" なら、バージョン番号は 3.10 です。	
引 数	out_Version	(出力) RGA のバージョン番号
リターン値	エラーコード、正常=0	

5.1.3 R_GRAPHICS_PrintRegisters

概 要	レジスターを print 表示します。	
ヘッダー	RGA.h	
宣 言	void R_GRAPHICS_PrintRegisters(void);	
補 足		
引 数	なし	
リターン値	なし	

5.1.4 R_GRAPHICS_Start

概 要	バッファリングされているグラフィックスのハードウェア描画を開始します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_Start(graphics_t* self);	
補 足	フレームパイプラインモードがオフのときに呼ぶことができます。 CPU の処理とグラフィックスのハードウェア処理が並行して動作するようになります。	
引 数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.5 R_GRAPHICS_FinishPreviousFrame

概 要	GPU は "frame_buffer_t :: draw_buffer_index" に描画を終了する。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_FinishPreviousFrame(graphics_t* self);	
補 足	<graphics_config_t.has_frame_pipeline> = true のときだけ使えます。 フレームバッファをスワップする直前に呼び出してください。 本関数で描画が完了した前のフレームがディスプレイに表示されたら、 <R_GRAPHICS_StartPreviousFrame> を呼び出してください。	
引 数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.6 R_GRAPHICS_StartPreviousFrame

概 要	前のフレームのハードウェア描画を開始します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_StartPreviousFrame(graphics_t* self);	
補 足	<graphics_config_t.has_frame_pipeline> = true のときだけ使えます。 <frame_buffer_t.draw_buffer_index> のフレームバッファに描画します。 <R_GRAPHICS_SetFrameBuffer> 関数で次のフレームを描画する設定し、 表示するフレームバッファをスワップした直後に呼び出してください。	
引 数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.7 R_GRAPHICS_GetHasFramePipeline

概要	フレームパイプラインモードのオン・オフを返します	
ヘッダー	RGA.h	
宣言	bool_t R_GRAPHICS_GetHasFramePipeline(graphics_t* self);	
補足	フレームパイプラインモードのオン・オフの設定は、 <graphics_config_t.has_frame_pipeline> に設定します。	
引数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	フレームパイプラインモードのオン・オフ	

5.1.8 R_GRAPHICS_BeginSoftwareRendering2

概要	ソフトウェア レンダリングを開始することを、グラフィックス ライブラリに通知します。	
ヘッダー	RGA.h	
宣言	errnum_t R_GRAPHICS_BeginSoftwareRendering2(graphics_t* self);	
補足	本関数は移植性のためにあります。	
引数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.9 R_GRAPHICS_BeginSoftwareRenderingA

概要	ソフトウェア レンダリングを開始することを、グラフィックス ライブラリに通知します。	
ヘッダー	RGA.h	
宣言	errnum_t R_GRAPHICS_BeginSoftwareRenderingA(graphics_t* self, void* in_Address);	
補足	本関数は移植性のためにあります。	
引数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.10 R_GRAPHICS_CONFIG_SetEmpty

概要	すべてのメンバー変数の値を空を意味する値で初期化します。	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_CONFIG_SetEmpty(graphics_config_t* out);	
補足	本関数は移植性のためにあります。	
引数	graphics_config_t* out	メンバー変数の値を変更する構造体
リターン値	なし	

5.1.11 R_GRAPHICS_InitConst

概要	RGA が利用する定数部分を初期化します	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_InitConst(graphics_t* self);	
補足	参考：(4.4.5)内部変数を定数で初期化する関数について（*_InitConst関数）	
引数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	なし	

5.1.12 R_GRAPHICS_Initialize

概 要	グラフィックス描画コンテキスト オブジェクトを初期化します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_Initialize (graphics_t* self, graphics_config_t* config);	
補 足	内部変数を初期化します。 OpenVG™ 用ルネサスグラフィックスプロセッサ(R-GPVG2)やJPEG コーデックユニット (JCU) を初期化します。 組み込み版では、本関数を呼び出す前に、割り込みハンドラーを登録してください。 self を使わなくなったら、R_GRAPHICS_Finalize を呼び出してください。 コンテキストは同時に1つしか使用できません。複数のフレームバッファに描画するときは、R_GRAPHICS_SetFrameBuffer を使用してフレームバッファを切り替えてください。 参考：(4.4.2) 初期化関数の内部の動き	
引 数	graphics_t* self	グラフィックス描画コンテキスト
	graphics_config_t* config	graphics_t の設定
リターン値	エラーコード、正常=0	

5.1.13 R_GRAPHICS_Finalize

概 要	グラフィックス描画コンテキスト オブジェクトの終了処理をします	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_Finalize(graphics_t* self, errnum_t e);	
補 足	エラーが発生しても、内部オブジェクトが残ることはありません。	
引 数	graphics_t* self	グラフィックス描画コンテキスト
	errnum_t e	これまでに発生したエラーコード。 エラー無し=0。
リターン値	エラーコード または e、0=成功かつ e=0。	

5.1.14 R_GRAPHICS_SetFrameBuffer

概 要	描画対象を変更します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_SetFrameBuffer(graphics_t* self, frame_buffer_t* frame_buffer);	
補 足	本関数を呼び出した時点の、 frame_buffer->buffer_address[frame_buffer->draw_bufferIndex] が指す VRAM 領域を、描画対象にします。 本関数の内部で、R_GRAPHICS_Finish 関数と同じ処理を行います。	
引 数	graphics_t* self	グラフィックス描画コンテキスト
	frame_buffer_t* frame_buffer	描画対象のフレームバッファ
リターン値	エラーコード、正常=0	

5.1.15 R_GRAPHICS_GetFrameBuffer

概要 描画対象を取得します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_GetFrameBuffer(graphics_t* self, frame_buffer_t** out_frame_buffer);`

補 足

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>frame_buffer_t** out_frame_buffer</code>	(出力) 描画対象のフレームバッファ
リターン値	エラーコード、正常=0

5.1.16 R_GRAPHICS_Finish

概要 描画が完了するまで待ちます。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_Finish(graphics_t* self);`

補 足 RGA による描画、および、アプリケーションが直接フレームバッファに対して行った描画が完了するのを待って、表示できる状態にします。
RGA の API を呼び出した後で、CPU からフレームバッファの内容を、キャッシュ領域、非キャッシュ領域のどちらでも、直接リードまたはライトするときは、
`R_GRAPHICS_BeginSoftwareRendering`～`R_GRAPHICS_EndSoftwareRendering` の呼び出しで囲むようにしてください。これらの関数の内部では、必要な時だけ、ハードウェア レンダリングの完了待ちや、CPU キャッシュのフラッシュを行います。
非同期で描画完了を待つときは、`R_GRAPHICS_FinishStart` 関数を呼び出してください。

引 数

リターン値

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0

5.1.17 R_GRAPHICS_Save

概要 コンテキストの設定値を指定されたステータス構造体に退避します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_Save(graphics_t* self, graphics_status_t* out_status);`

補 足 例 :

```
graphics_status_t status = {0};
e= R_GRAPHICS_Save( &graphics, &status ); IF(e){goto fin;}
:
fin:
e= R_GRAPHICS_Restore( &graphics, &status, e );
```

引 数

リターン値

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_status_t* out_status</code>	(出力) コンテキストの設定値
リターン値	エラーコード、正常=0

5.1.18 R_GRAPHICS_Restore

概要 ステータス構造体の内容を、コンテキストに戻します。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_Restore(graphics_t* self, graphics_status_t* status, errnum_t e);`

補足

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_status_t* status</code>	コンテキストの設定値
<code>errnum_t e</code>	これまでに発生したエラーコード。 エラー無し=0

リターン値 エラーコード または e、0=成功かつ e=0

5.1.19 R_GRAPHICS_ResetMatrix

概要 行列演算関数の対象となる行列を単位行列にリセットします。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_ResetMatrix(graphics_t* self);`

補足

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
-------------------------------	-----------------

リターン値 エラーコード、正常=0

5.1.20 R_GRAPHICS_SetMatrix_2x3

概要 現在行列の各要素を設定します。(2x3)
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_SetMatrix_2x3(graphics_t* self, graphics_matrix_float_t sx, graphics_matrix_float_t ky, graphics_matrix_float_t kx, graphics_matrix_float_t sy, graphics_matrix_float_t tx, graphics_matrix_float_t ty);`

補足

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t sx, graphics_matrix_float_t ky, graphics_matrix_float_t kx, graphics_matrix_float_t sy, graphics_matrix_float_t tx, graphics_matrix_float_t ty</code>	2 行 3 列の行列 $\begin{pmatrix} sx & kx & tx \\ ky & sy & ty \\ 0 & 0 & 1 \end{pmatrix}$

リターン値 エラーコード、正常=0

5.1.21 R_GRAPHICS_SetMatrix_3x3

概要 現在行列(Matrix[])の各要素を設定します。(3x3)
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_SetMatrix_3x3(graphics_t* self, graphics_matrix_float_t * matrix);`

補足

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t * matrix</code>	3 行 3 列の行列 (配列) $\begin{pmatrix} \text{Matrix}[0] & \text{Matrix}[3] & \text{Matrix}[6] \\ \text{Matrix}[1] & \text{Matrix}[4] & \text{Matrix}[7] \\ \text{Matrix}[2] & \text{Matrix}[5] & \text{Matrix}[8] \end{pmatrix}$

リターン値 エラーコード、正常=0

5.1.22 R_GRAPHICS_GetMatrix_3x3

概要 現在行列(Matrix[])の各要素を取得します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_GetMatrix_3x3(graphics_t* self, graphics_matrix_float_t * out_matrix);`

補足
引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t * out_matrix</code>	(出力) 3 行 3 列の行列 (配列) $\begin{pmatrix} \text{Matrix}[0] & \text{Matrix}[3] & \text{Matrix}[6] \\ \text{Matrix}[1] & \text{Matrix}[4] & \text{Matrix}[7] \\ \text{Matrix}[2] & \text{Matrix}[5] & \text{Matrix}[8] \end{pmatrix}$

リターン値 エラーコード、正常=0

5.1.23 R_GRAPHICS_TranslateMatrixI

概要 現在行列(M)を平行移動します。(整数型指定)

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_TranslateMatrixI(graphics_t* self, int_t tx, int_t ty);`

補足

$$M=M \cdot \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>int_t tx, int_t ty</code>	移動量。左上が原点なら、X のプラスは右方向、Y のプラスは下方向。

リターン値 エラーコード、正常=0

5.1.24 R_GRAPHICS_TranslateMatrix

概要 現在行列(M)を平行移動します。(浮動小数型指定)

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_TranslateMatrix(graphics_t* self, graphics_matrix_float_t tx, graphics_matrix_float_t ty);`

補足

$$M=M \cdot \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

引数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t tx</code> <code>graphics_matrix_float_t ty</code>	移動量。左上が原点なら、X のプラスは右方向、Y のプラスは下方向。

リターン値 エラーコード、正常=0

5.1.25 R_GRAPHICS_ScaleMatrix

概要 現在行列(M)を拡大縮小します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_ScaleMatrix(graphics_t* self,
graphics_matrix_float_t sx, graphics_matrix_float_t sy);`

補足

$$M=M \cdot \begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

移植した際は、演算誤差が発生する可能性に注意してください。

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>graphics_matrix_float_t tx</code>	倍率。拡大縮小の中心は原点
	<code>graphics_matrix_float_t ty</code>	

リターン値 エラーコード、正常=0

5.1.26 R_GRAPHICS_RotateMatrixDegree

概要 現在行列(M)を回転します。回転の中心座標は (0,0)です。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_RotateMatrixDegree(graphics_t* self,
graphics_matrix_float_t angle);`

補足

$$M=M \cdot \begin{pmatrix} \cos(\text{angle}) & -\sin(\text{angle}) & 0 \\ \sin(\text{angle}) & \cos(\text{angle}) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

移植した際は、演算誤差が発生する可能性に注意してください。

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>graphics_matrix_float_t angle</code>	回転する角度。単位は度。左上が原点なら、プラスが時計回り

リターン値 エラーコード、正常=0

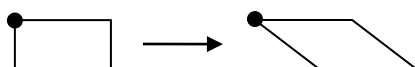
5.1.27 R_GRAPHICS_ShearMatrix

概要 現在行列(M)をせん断変形します。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_ShearMatrix(graphics_t* self,
 graphics_matrix_float_t shx, graphics_matrix_float_t shy);`

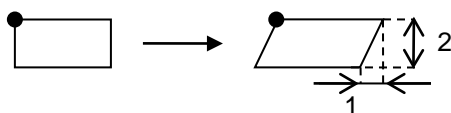
補足

$$M = M \cdot \begin{pmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

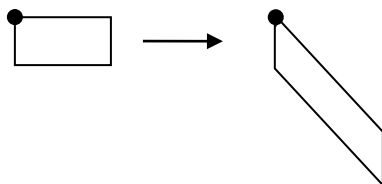
(shx, shy) = (1.0, 0.0) の場合、垂直の辺が 45 度に傾く平行四辺形になります。
 ただし、原点が長方形の左上になれば、移動を伴ってしまうので注意してください。



(shx, shy) = (-0.5, 0.0) の場合、底辺 : 高さ = 1 : 2 の三角形の斜辺のような平行四辺形になります。



(shx, shy) = (0.0, 1.0) の場合、水平の辺が 45 度に傾く平行四辺形になります。



移植した際は、演算誤差が発生する可能性に注意してください。

引 数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>graphics_matrix_float_t shx</code> <code>graphics_matrix_float_t shy</code>	せん断の割合。せん断の中心は原点
リターン値	エラーコード、正常=0	

5.1.28 R_GRAPHICS_TransformMatrix

概要 現在行列(M)に対して指定した 2 行 3 列の行列を積算します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_TransformMatrix(graphics_t* self,
graphics_matrix_float_t sx, graphics_matrix_float_t ky,
graphics_matrix_float_t kx, graphics_matrix_float_t sy,
graphics_matrix_float_t tx, graphics_matrix_float_t ty);`

補 足

$$M=M \cdot \begin{pmatrix} sx & kx & tx \\ ky & sy & ty \\ 0 & 0 & 1 \end{pmatrix}$$

移植した際は、演算誤差が発生する可能性に注意してください。

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t sx</code> <code>graphics_matrix_float_t ky</code> <code>graphics_matrix_float_t kx</code> <code>graphics_matrix_float_t sy</code> <code>graphics_matrix_float_t tx</code> <code>graphics_matrix_float_t ty</code>	掛ける行列。 2 行 3 列の行列

リターン値

エラーコード、正常=0

5.1.29 R_GRAPHICS_MultiplyMatrix

概要 現在行列(M)に対して指定した 3 行 3 列の行列(Matrix[])を積算します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_MultiplyMatrix(graphics_t* self, graphics_matrix_float_t
* matrix);`

補 足

$$M=M \cdot \begin{pmatrix} Matrix[0] & Matrix[3] & Matrix[6] \\ Matrix[1] & Matrix[4] & Matrix[7] \\ Matrix[2] & Matrix[5] & Matrix[8] \end{pmatrix}$$

移植した際は、演算誤差が発生する可能性に注意してください。

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>graphics_matrix_float_t * matrix</code>	掛ける行列。 3 行 3 列の行列。 要素数 9 の配列

リターン値

エラーコード、正常=0

5.1.30 R_GRAPHICS_GetProjectiveMatrix

概 要	任意の形状の四角形から、任意の形状の四角形へ、変形するような行列を取得します。	
ヘッダー	RGA.h	
宣 言	<pre>errnum_t R_GRAPHICS_GetProjectiveMatrix(graphics_matrix_float_t source_top_left_x, graphics_matrix_float_t source_top_left_y, graphics_matrix_float_t source_top_right_x, graphics_matrix_float_t source_top_right_y, graphics_matrix_float_t source_bottom_left_x, graphics_matrix_float_t source_bottom_left_y , graphics_matrix_float_t source_bottom_right_x, graphics_matrix_float_t source_bottom_right_y, graphics_matrix_float_t destination_top_left_x, graphics_matrix_float_t destination_top_left_y, graphics_matrix_float_t destination_top_right_x, graphics_matrix_float_t destination_top_right_y, graphics_matrix_float_t destination_bottom_left_x, graphics_matrix_float_t destination_bottom_left_y, graphics_matrix_float_t destination_bottom_right_x, graphics_matrix_float_t destination_bottom_right_y, graphics_matrix_float_t * out_matrix);</pre>	
補 足	移植した際は、演算誤差が発生する可能性に注意してください。	
引 数	graphics_matrix_float_t source*	変換前の 4 点の座標。設定に制限はありません
	graphics_matrix_float_t destination*	変換後の 4 点の座標。設定に制限はありません
	graphics_matrix_float_t * out_matrix	(出力) 3 行 3 列の行列。配列要素数は 9
リターン値	エラーコード、正常=0	

5.1.31 R_GRAPHICS_SetBackgroundColor

概 要	背景色を設定します。	
ヘッダー	RGA.h	
宣 言	<pre>errnum_t R_GRAPHICS_SetBackgroundColor(graphics_t* self, r8g8b8a8_t color);</pre>	
補 足	<p>アルファなしのフレームバッファを描画対象にしているときは、背景色とクリア色は同じになります。背景色のデフォルトは、白です。</p> <p>アルファ付きのフレームバッファを描画対象にしているときは、背景色は引数に指定した色になりますが、描画対象のフレームバッファに描くクリア色は、常に透明な黒になります。背景色を、奥のレイヤーに設定することで正しく表示されます。</p>	
引 数	graphics_t* self	グラフィックス描画コンテキスト
	r8g8b8a8_t color	背景色。取得は R_RGA_Get_R8G8B8A8() 使用してください
リターン値	エラーコード、正常=0	

5.1.32 R_GRAPHICS_GetBackgroundColor

概要 背景色を取得します。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_GetBackgroundColor(graphics_t* self, r8g8b8a8_t* out_color);`

補 足

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>r8g8b8a8_t* out_color</code>	(出力) 背景色

リターン値

エラーコード、正常=0

5.1.33 R_GRAPHICS_GetClearColor

概要 R_GRAPHICS_Clear するときの色を取得します。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_GetClearColor(graphics_t* self, r8g8b8a8_t* out_color);`
 補 足 クリア色の設定は、R_GRAPHICS_SetBackgroundColor()を使用してください。

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>r8g8b8a8_t* out_color</code>	(出力) クリア色

リターン値

エラーコード、正常=0

5.1.34 R_GRAPHICS_Clear

概要 長方形の領域をクリアします。
 ヘッダー RGA.h
 宣言 `errnum_t R_GRAPHICS_Clear(graphics_t* self, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);`
 補 足 クリア色の設定は、R_GRAPHICS_SetBackgroundColor()を使用してください。
 ダブルバッファーのときは、描画バッファーの中をクリアするので、本関数を呼び出しただけでは、クリアした内容は表示されません。表示を反映するためには R_WINDOW_SURFACES_SwapBuffers()を使用してください。
 本関数は、クリッピングや行列の影響を受けます。

引 数

<code>graphics_t* self</code>	グラフィックス描画コンテキスト
<code>int_fast32_t min_x,</code> <code>int_fast32_t min_y,</code> <code>int_fast32_t width,</code> <code>int_fast32_t height</code>	長方形の領域。設定に制限はありません

リターン値

エラーコード、正常=0

5.1.35 R_GRAPHICS_DrawImage

概 要	画像(image)を座標(min_x, min_y)へ等倍で描画します。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_DrawImage(graphics_t* self, graphics_image_t* image, int_fast32_t min_x, int_fast32_t min_y);
補 足	image 引数に指定する画像データは、(6.1)画像フォーマット変換 ImagePackagerを使って作成できます。 引数に直接 JPEG データなどを指定できます。

R_GRAPHICS_SetGlobalAlpha の影響を受けます。

現在の行列とクリッピングの影響を受けます。

画像にアルファ成分が含まれていて、描画対象のフレームバッファにアルファ成分が含まれていないときは、フレームバッファに描かれる RGB 成分は、アルファ成分で乗算済みの値にブレンドされます。 描画対象のフレームバッファにアルファ成分が含まれているときは、RGB 成分はアルファ成分で乗算される前の値にブレンドされます。

アルファ成分が含まれているピクセルフォーマットの例：

ARGB8888, ARGB4444

アルファ成分が含まれていないピクセルフォーマットの例：

XRGB8888, RGB565

ラスターフォントで文字を描画するときは、ピクセルフォーマットが A8 のアルファのみの画像を image 引数に指定してください。A8 画像は拡大縮小ができます。

CLUT 形式のフレームバッファに描画するときは、フレームバッファと同じビット数を持つ CLUT 形式の画像を指定してください。描画位置は、min_x = 0, min_y = 0 しか指定できません。画像の幅がバイト単位ではないときは、エラーになります。表示コントローラーの CLUT の色を描画した画像に合わせるには、graphics_image_properties_t の CLUT を表示コントローラーに設定してください。

Image 引数に指定した画像データが、アプリケーションが用意した配列変数の中にあるときは、R_GRAPHICS_DrawImage 関数を呼び出す前にフラッシュが必要です。ただし、ROM にある画像データを使うときは、フラッシュする必要はありません。

フラッシュするときは、直接 CPU のキャッシュフラッシュを行なうか、画像データにリードおよびライトする処理を、R_GRAPHICS_BeginSoftwareRendering ～ R_GRAPHICS_EndSoftwareRendering で囲むようにしてください。

引 数	graphics_t* self	グラフィックス描画コンテキスト
	graphics_image_t* image	画像
	int_fast32_t min_x, int_fast32_t min_y	X, Y 座標の最小値。設定に制限はありません
リターン値	エラーコード、正常=0	

5.1.36 R_GRAPHICS_DrawImageResized

概要	画像(image)を指定した長方形の中に拡大縮小して描画します。	
ヘッダー	RGA.h	
宣言	errnum_t R_GRAPHICS_DrawImageResized(graphics_t* self, graphics_image_t* image, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);	
補足	R_GRAPHICS_DrawImage 関数の説明を参照 (5.1.35)。 image 引数に JPEG データは指定できません。	
引数	graphics_t* self	グラフィックス描画コンテキスト
	graphics_image_t* image	画像
	int_fast32_t min_x, int_fast32_t min_y	X, Y 座標の最小値。設定に制限はありません
	int_fast32_t width, int_fast32_t height	描画対象の幅と高さ。[設定範囲] 0 以上
リターン値	エラーコード、正常=0	

5.1.37 R_GRAPHICS_DrawImageChild

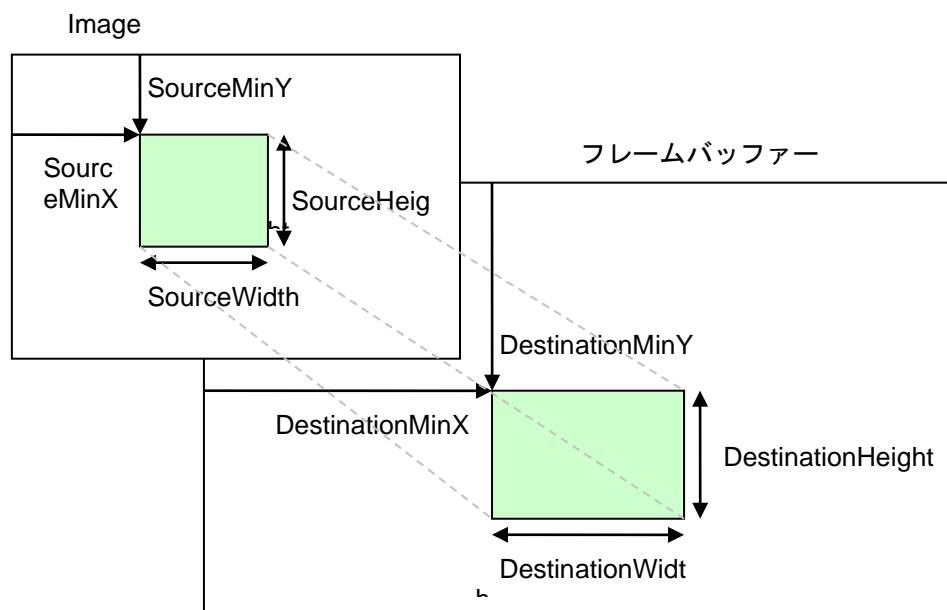
概要 画像(image)の一部を描画します。

ヘッダー RGA.h

宣言

```
errnum_t R_GRAPHICS_DrawImageChild( graphics_t* self, graphics_image_t*
image,
int_fast32_t source_min_x, int_fast32_t source_min_y,
int_fast32_t source_width, int_fast32_t source_height,
int_fast32_t destination_min_x, int_fast32_t destination_min_y,
int_fast32_t destination_width, int_fast32_t destination_height );
```

補足



source_width \neq destination_width または source_height \neq destination_height のときは、拡大縮小します。

平行移動と拡大縮小以外の行列演算を行って画像の一部を描画することはできません。

その他、R_GRAPHICS_DrawImage 関数の説明を参照 (5.1.35)。

引 数

graphics_t* self	グラフィックス描画コンテキスト
graphics_image_t* image	画像
int_fast32_t source_min_x int_fast32_t source_min_y	画像の中の、X, Y 座標の最小値。設定に制限はありません
int_fast32_t source_width int_fast32_t source_height	画像の中の、幅と高さ。[設定範囲] 0 以上
int_fast32_t destination_min_x int_fast32_t destination_min_y	描画対象の X, Y 座標の最小値。設定に制限はありません
int_fast32_t destination_width int_fast32_t destination_height	描画対象の幅と高さ。[設定範囲] 0 以上
エラーコード、正常=0	

リターン値

5.1.38 R_GRAPHICS_FillRect

概 要	引数で指定される長方形領域を塗りつぶします。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_FillRect(graphics_t* self, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);
補 足	現在の行列と現在の塗りつぶしとクリッピングの影響を受けます。 境界線は描画しません。 R_GRAPHICS_SetGlobalAlpha の影響を受けます。

引 数	graphics_t* self	グラフィックス描画コンテキスト
	int_fast32_t min_x, int_fast32_t min_y	長方形の X, Y 座標の最小値。設定に制限はありません
	int_fast32_t width, int_fast32_t height	長方形の幅と高さ。[設定範囲] 0 以上
リターン値	エラーコード、正常=0	

5.1.39 R_GRAPHICS_SetFillColor

概 要	現在の塗りつぶしのペイントオブジェクトを単色塗りつぶしに変更し、塗りつぶし色を設定します。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_SetFillColor(graphics_t* self, r8g8b8a8_t Color);
補 足	初期値は、不透明な黒です。

引 数	graphics_t* self	グラフィックス描画コンテキスト
	r8g8b8a8_t color	塗りつぶしの色。設定には R_RGA_Get_R8G8B8A8() を使用してください。
リターン値	エラーコード、正常=0	

5.1.40 R_GRAPHICS_SetFillPattern

概 要	現在の塗りつぶしのペイントオブジェクトに、パターンを設定します。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_SetFillPattern(graphics_t* self, graphics_pattern_t* pattern);
補 足	描画するときは、R_GRAPHICS_FillRect を呼び出してください。 パターンを使わないときは、R_GRAPHICS_SetFillColor を呼び出してください。

引 数	graphics_t* self	グラフィックス描画コンテキスト
	graphics_pattern_t* pattern	パターンオブジェクトの初期化は R_GRAPHICS_PATTERN_Initialize() を使用してください。
リターン値	エラーコード、正常=0	

5.1.41 R_GRAPHICS_BeginPath

概 要	デフォルト パスの内容を空にリセットします。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_BeginPath(graphics_t* self);
補 足	

引 数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.42 R_GRAPHICS_Rect

概要 デフォルト パスに、長方形を追加します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_Rect(graphics_t* self, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);`

補足 本関数は、クリッピング領域を設定するために使います。

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>int_fast32_t min_x,</code> <code>int_fast32_t min_y</code>	長方形の X, Y 座標の最小値。設定に制限はありません
	<code>int_fast32_t width,</code> <code>int_fast32_t height</code>	長方形の幅と高さ。[設定範囲] 0 以上
リターン値	エラーコード、正常=0	

5.1.43 R_GRAPHICS_Clip

概要 現在のデフォルト パスの形状をクリッピング領域にします。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_Clip(graphics_t* self);`

補足 現在のデフォルト パスが、空、または、1つの長方形ではないときは、エラーになります。

現在のデフォルト パスが空のときは、どこにも描けないようになります。

現在のクリッピング領域がフレームバッファの一部のときに、本関数を呼び出すと、これまでのクリッピング領域とデフォルト パスに共通する領域が、新しいクリッピング領域になります。

全体を描けるように戻すには、`R_GRAPHICS_Restore` を呼び出します。

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.44 R_GRAPHICS_SetGlobalAlpha

概要 すべての描画に対してかける1つのアルファ値（非透明度）を設定します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_SetGlobalAlpha(graphics_t* self, uint8_t alpha_value);`

補足 デフォルト値は、255 です。

次の描画関数に影響します。

`R_GRAPHICS_FillRect` などの図形の塗りつぶし

`R_GRAPHICS_FillRect` のパターン描画

`R_GRAPHICS_StrokeRect` などの図形の境界線の描画

`R_GRAPHICS_DrawImage` などの画像の描画

次の描画関数には影響しません。

`R_GRAPHICS_Clear`

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>uint8_t alpha_value</code>	アルファ値。[設定範囲] 0 ~ 255。小さいほど薄く描画する
リターン値	エラーコード、正常=0	

5.1.45 R_GRAPHICS_GetGlobalAlpha

概要 すべての描画に対してかける 1 つのアルファ値（非透明度）を取得します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_GetGlobalAlpha(graphics_t* self, uint8_t* out_alpha_value);`

補足

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>uint8_t* out_alpha_value</code>	（出力）アルファ値。[設定範囲] 0 ～ 255。小さいほど薄く描画する
リターン値	エラーコード、正常=0	

5.1.46 R_GRAPHICS_SetGlobalCompositeOperation

概要 アルファブレンドするときの演算方法を設定します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_SetGlobalCompositeOperation(graphics_t* self, graphics_composite_operation_t composite_operation);`

補足 GRAPHICS_SOURCE_OVER 以外に設定できるのは、次のケースです。
R_GRAPHICS_DrawImage などの画像の描画

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>graphics_composite_operation_t composite_operation</code>	演算方法
リターン値	エラーコード、正常=0	

5.1.47 R_GRAPHICS_GetGlobalCompositeOperation

概要 アルファブレンドするときの演算方法を取得します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_GetGlobalCompositeOperation(graphics_t* self, graphics_composite_operation_t* out_composite_operation);`

補足

引数	<code>graphics_t* self</code>	グラフィックス描画コンテキスト
	<code>graphics_composite_operation_t* out_composite_operation</code>	（出力）演算方法
リターン値	エラーコード、正常=0	

5.1.48 R_GRAPHICS_SetQualityFlags

概要 描画品質を設定します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_SetQualityFlags(graphics_t* self, graphics_quality_flags_t qualities);`

補足

引数	<code>graphics_t* self,</code>	グラフィックス描画コンテキスト
	<code>graphics_quality_flags_t qualities</code>	
リターン値	エラーコード、正常=0	

5.1.49 R_GRAPHICS_GetQualityFlags

概 要	現在の描画品質を取得します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_GetQualityFlags(graphics_t* self, graphics_quality_flags_t* out_qualities);	
補 足		
引 数	graphics_t* self,	グラフィックス描画コンテキスト
	graphics_quality_flags_t* out_qualities	
リターン値	エラーコード、正常=0	

5.1.50 R_GRAPHICS_SetStrokeColor

概 要	現在の境界線のペイントオブジェクトを、単色塗りつぶしをするようにして、その色を設定します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_SetStrokeColor(graphics_t* self, r8g8b8a8_t color);	
説 明		
引 数	graphics_t* self,	グラフィックス描画コンテキスト
	r8g8b8a8_t color	境界線の色
リターン値	エラーコード、正常=0	

5.1.51 R_GRAPHICS_StrokeRect

概 要	長方形の辺を描画します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_StrokeRect(graphics_t* self, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);	
説 明	線の幅と境界線の色とクリッピングの影響を受けます。 塗りつぶしはしません。 現在の行列が、単位行列でなければ、エラーになります。 R_GRAPHICS_SetGlobalAlpha の影響を受けます。	
引 数	graphics_t* self,	グラフィックス描画コンテキスト
	int_fast32_t min_x, int_fast32_t min_y	長方形の X, Y 座標の最小値。設定に制限はありません
	int_fast32_t width, int_fast32_t height	長方形の幅と高さ。[設定範囲] 0 以上
リターン値	エラーコード、正常=0	

5.1.52 R_GRAPHICS_BeginSoftwareRendering

概 要	ソフトウェア レンダリングを開始することを、グラフィックス ライブラリに通知します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_BeginSoftwareRendering(graphics_t* self);	
補 足	内蔵 RAM に対してソフトウェア レンダリングするときに本関数を使います。条件によっては、内部で描画完了待ち (R_GRAPHICS_Finish 関数相当) が発生します。	
引 数	graphics_t* self,	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.53 R_GRAPHICS_EndSoftwareRendering

概 要	ソフトウェア レンダリングが終了したことを、グラフィックス ライブラリに通知します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_EndSoftwareRendering(graphics_t* self);	
補 足	内蔵 RAM に対してソフトウェア レンダリングするときに本関数を使います。エラー処理に対応した関数の中から本関数を呼び出すときは、関数の最後で R_GRAPHICS_EndRenderingInFin も呼び出すようにしてください。	
引 数	graphics_t* self,	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.1.54 R_GRAPHICS_EndRenderingInFin

概 要	ソフトウェア レンダリングを行なう関数の最後から本関数を呼び出してください。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_EndRenderingInFin(graphics_t* self, errnum_t e);	
補 足	内蔵 RAM に対してソフトウェア レンダリングするときに本関数を使います。エラー処理に対応した関数の中から本関数を呼び出すときは、関数の最後で R_GRAPHICS_EndRenderingInFin も呼び出すようにしてください。	
引 数	graphics_t* self,	グラフィックス描画コンテキスト
	errnum_t e	これまでに発生したエラーコード。 エラー無し=0
リターン値	エラーコード、正常=0	

5.2 graphics_image_t クラスのメンバー関数に相当する関数

5.2.1 一覧

章	関数名	概要
5.2.2	R_GRAPHICS_IMAGE_InitByShareFrameBuffer	フレームバッファの内容をイメージとして初期化します。
5.2.3	R_GRAPHICS_IMAGE_InitR8G8B8A8	r8g8b8a8_t のイメージオブジェクトを初期化します。
5.2.4	R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8	同じ幅と高さに、イメージオブジェクトを初期化します。
5.2.5	R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8	表示中のフレームバッファの一部をコピーした、イメージオブジェクトを初期化します。
5.2.6	R_GRAPHICS_IMAGE_GetProperties	画像のプロパティを取得します
5.2.7	R_GRAPHICS_IMAGE_InitByShareFrameBufferEx	フレームバッファの内容をイメージとして初期化します。
5.2.8	R_GRAPHICS_IMAGE_GetImageFormat	イメージフォーマットを取得する。

5.2.2 R_GRAPHICS_IMAGE_InitByShareFrameBuffer

概 要 フレームバッファの内容をイメージとして初期化します。

ヘッダー RGA.h

宣 言 `errnum_t R_GRAPHICS_IMAGE_InitByShareFrameBuffer(graphics_image_t* self, frame_buffer_t* frame_buffer);`

補 足 内部変数を初期化します。

本関数を呼び出した時点の

`frame_buffer->buffer_address[frame_buffer->show_buffer_index]` が指す VRAM 領域を、イメージ (self) と共有します。

[条件] CLUT 形式のイメージは作成できません。CLUT 形式は、ImagePackager による静的なイメージのみ対応しています。

[条件] stride のアラインメントが合っていないエラーになるときは、エラーにならないような幅のバッファを指定し、そのバッファに画像をコピーしておいてください。RGA はそのコピーができません。そして、R_GRAPHICS_DrawImageChild 関数で 画像 (バッファの一部) を描画してください。

引 数	<code>graphics_image_t* self</code>	画像
	<code>frame_buffer_t* frame_buffer</code>	イメージの内容が入っているフレームバッファ
リターン値	エラーコード、正常=0	

5.2.3 R_GRAPHICS_IMAGE_InitR8G8B8A8

概 要	r8g8b8a8_t のイメージオブジェクトを初期化します。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_IMAGE_InitR8G8B8A8(graphics_image_t* self, void* image_data_array, size_t image_data_array_size, int_fast32_t width, int_fast32_t height);
補 足	内部変数を初期化します。 取得できるイメージデータの並びは、uint8_t 型の配列で、Red, Green, Blue, Alpha の順に、左上のピクセルから右下のピクセルの順に並んでいます。 R8G8B8A8 フォーマットは、ARGB8888 フォーマットと異なること(図 1-4)と、ハードウェア レンダリングできないことに注意してください。

その他のピクセル フォーマットのイメージを作成するときは、
R_GRAPHICS_IMAGE_InitByShareFrameBuffer (5.2.2) を呼び出してください。

引 数	graphics_image_t* self	画像
	void* image_data_array	イメージを格納するメモリー領域とする配列の先頭アドレス
	size_t image_data_array_size	image_data_array が指すメモリー領域のサイズ (バイト)
	int_fast32_t width, int_fast32_t height	イメージの幅と高さ
リターン値	エラーコード、正常=0	

5.2.4 R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8

概 要	同じ幅と高さに、イメージオブジェクトを初期化します。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8(graphics_image_t* self, void* image_data_array, size_t image_data_array_size, graphics_image_t* same_size_image);
補 足	内部変数を初期化します。 R8G8B8A8 フォーマットは、ARGB8888 フォーマットと異なること(図 1-4)と、ハードウェア レンダリングできないことに注意してください。

その他のピクセル フォーマットのイメージを作成するときは、
R_GRAPHICS_IMAGE_InitByShareFrameBuffer (5.2.2) を呼び出してください。

引 数	graphics_image_t* self	画像
	void* image_data_array	イメージを格納するメモリー領域とする配列の先頭アドレス
	size_t image_data_array_size	image_data_array が指すメモリー領域のサイズ (バイト)
	graphics_image_t* same_size_image	幅と高さを参照するイメージオブジェクト
リターン値	エラーコード、正常=0	

5.2.5 R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8

概要 表示中のフレームバッファの一部をコピーした、イメージオブジェクトを初期化します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8(graphics_image_t* self, void* image_data_array, size_t image_data_array_size, graphics_t* context, int_fast32_t min_x, int_fast32_t min_y, int_fast32_t width, int_fast32_t height);`

補足 内部変数を初期化します。

取得できるイメージデータの並びは、uint8_t 型の配列で、Red, Green, Blue, Alpha の順に、左上のピクセルから右下のピクセルの順に並んでいます。

R8G8B8A8 フォーマットは、ARGB8888 フォーマットと異なること(図 1-4)と、ハードウェア レンダリングできないことに注意してください。

その他のピクセル フォーマットのイメージを作成するときは、

R_GRAPHICS_IMAGE_InitByShareFrameBuffer (5.2.2) を呼び出してください。

引 数	<code>graphics_image_t* self</code>	画像
	<code>void* image_data_array</code>	イメージを格納するメモリー領域とする配列の先頭アドレス
	<code>size_t image_data_array_size</code>	<code>image_data_array</code> が指すメモリー領域のサイズ (バイト)
	<code>graphics_t* context</code>	コピー元のフレームバッファを描画対象にしたコンテキスト
	<code>int_fast32_t min_x, int_fast32_t min_y</code>	取得する範囲の X, Y 座標の最小値 (フレームバッファ座標)。設定に制限はありません
	<code>int_fast32_t width, int_fast32_t height</code>	取得する範囲の幅と高さ。[設定範囲] 0 以上
リターン値	エラーコード、正常=0	

5.2.6 R_GRAPHICS_IMAGE_GetProperties

概要 画像のプロパティを取得します。

ヘッダー RGA.h

宣言 `errnum_t R_GRAPHICS_IMAGE_GetProperties(graphics_image_t* self, graphics_image_properties_t* out_properties);`

補足 `out_properties->address` に出力されるアドレスが指す配列にリードまたはライトするときは、フラッシュが必要です。ただし、ROM データにアクセスするときは、フラッシュする必要はありません。

フラッシュするときは、直接 CPU のキャッシュフラッシュを行うか

R_GRAPHICS_Finish 関数を呼び出すか、画像データにリードおよびライトする処理を、R_GRAPHICS_BeginSoftwareRendering ~

R_GRAPHICS_EndSoftwareRendering で囲むようにしてください。

引 数	<code>graphics_image_t* self</code>	画像
	<code>graphics_image_properties_t* out_properties</code>	(出力) 画像のプロパティ
リターン値	エラーコード、正常=0	

5.2.7 R_GRAPHICS_IMAGE_InitByShareFrameBufferEx

概 要	フレームバッファの内容をイメージとして初期化します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_IMAGE_InitByShareFrameBufferEx(graphics_image_t* self, frame_buffer_t* in_FrameBuffer, graphics_t* in_GraphicsContext);	
補 足	フレームパイプラインモードがオンのときでも、正しく動作します。 機能は <R_GRAPHICS_IMAGE_InitByShareFrameBuffer> と同じです。	
引 数	graphics_image_t* self	初期化しようとしているイメージ
	frame_buffer_t* in_FrameBuffer	イメージの内容が入っているフレームバッファ
	graphics_t* in_GraphicsContext	グラフィックス描画コンテキスト
	エラーコード、正常=0	
リターン値	エラーコード、正常=0	

5.2.8 R_GRAPHICS_IMAGE_GetImageFormat

概 要	イメージフォーマットを取得する。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_IMAGE_GetImageFormat(const graphics_image_t* self, pixel_format_t* out_Format);	
補 足		
引 数	graphics_image_t* self	初期化しようとしているイメージ
	pixel_format_t* out_Format	イメージの内容が入っているフレームバッファ
リターン値	エラーコード、正常=0	

5.3 graphics_pattern_t クラスのメンバー関数に相当する関数

5.3.1 一覧

章	関数名	概要
5.3.2	R_GRAPHICS_PATTERN_Initialize	パターンオブジェクトを初期化します。

5.3.2 R_GRAPHICS_PATTERN_Initialize

概 要	パターンオブジェクトを初期化します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_PATTERN_Initialize(graphics_pattern_t* self, graphics_image_t* image, repetition_t repetition, graphics_t* context);	
補 足	内部変数を初期化します。 GraphicsPatternClass のオブジェクトは、R_GRAPHICS_SetFillPattern に設定してください。	
引 数	graphics_pattern_t* self	パターンオブジェクト
	graphics_image_t* image	パターンの構成要素である画像
	repetition_t repetition	繰り返しの設定。通常、GRAPHICS_REPEAT
	graphics_t* context	所属するコンテキスト
リターン値	エラーコード、正常=0	

5.4 window_surfaces_t クラスのメンバー関数に相当する関数

5.4.1 一覧

章	関数名	概要
5.4.2	R_WINDOW_SURFACES_InitConst	内部変数を定数で初期化します。
5.4.3	R_WINDOW_SURFACES_Initialize	表示デバイス、またはウィンドウを初期化して、グラフィックスの表示を開始します。
5.4.4	R_WINDOW_SURFACES_Finalize	表示デバイスの終了処理をします。
5.4.5	R_WINDOW_SURFACES_GetLayerFrameBuffer	指定したレイヤーのフレームバッファ構造体へのポインタを取得します。
5.4.6	R_WINDOW_SURFACES_GetLayerCount	レイヤーの数を取得します。
5.4.7	R_WINDOW_SURFACES_SwapBuffers	指定したレイヤーのバッファをスワップして、描画していた内容を表示します。
5.4.8	R_WINDOW_SURFACES_DoMessageLoop	メッセージループに入ります。
5.4.9	R_WINDOW_SURFACES_AccessLayerAttributes	指定した表示レイヤーの属性にアクセスします。
5.4.10	R_WINDOW_SURFACES_AllocOffscreenStack	オフスクリーン バッファを VRAM のスタックから確保します
5.4.11	R_WINDOW_SURFACES_FreeOffscreenStack	オフスクリーン バッファを解放し、VRAM のスタックに返却します
5.4.12	R_WINDOW_SURFACES_SetLayerAttributes	指定した表示レイヤーの属性を設定します。
5.4.13	R_WINDOW_SURFACES_CONFIG_SetEmpty	すべてのメンバー変数の値を空を意味する値で初期化します。
5.4.14	R_LAYER_ATTRIBUTES_SetEmpty	すべてのメンバー変数の値を空を意味する値で初期化します。

5.4.2 R_WINDOW_SURFACES_InitConst

概 要	内部変数を定数で初期化します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	void R_WINDOW_SURFACES_InitConst(window_surfaces_t* self);	
補 足		
引 数	window_surfaces_t* self	フレームバッファと画面表示
リターン値	なし	

5.4.3 R_WINDOW_SURFACES_Initialize

概 要	表示デバイス、またはウィンドウを初期化します。
ヘッダー	RGA.h, window_surfaces.h
宣 言	errnum_t R_WINDOW_SURFACES_Initialize(window_surfaces_t* self, window_surfaces_config_t* in_out_config);
補 足	独自に表示を制御するときは、window_surfaces_t クラスを使う代わりに、直接 frame_buffer_t 構造体を使ってください。 内部変数を初期化します。 初期化後は画面全体が黒くなります。 R_WINDOW_SURFACES_SwapBuffers を呼び出すと表示を開始します。

引 数	window_surfaces_t* self	フレームバッファと画面表示
	window_surfaces_config_t* in_out_config	
リターン値	エラーコード、正常=0	

5.4.4 R_WINDOW_SURFACES_Finalize

概 要	表示デバイスの終了処理をします。
ヘッダー	RGA.h, window_surfaces.h
宣 言	errnum_t R_WINDOW_SURFACES_Finalize (window_surfaces_t* self, errnum_t e);
補 足	エラーが発生しても、内部オブジェクトが残ることはありません。

引 数	window_surfaces_t* self	フレームバッファと画面表示
	errnum_t e	これまでに発生したエラーコード。 エラー無し=0。
リターン値	エラーコード または e、0=成功かつ e=0。	

5.4.5 R_WINDOW_SURFACES_GetLayerFrameBuffer

概 要	指定したレイヤーのフレームバッファ構造体へのポインターを取得します。
ヘッダー	RGA.h, window_surfaces.h
宣 言	errnum_t R_WINDOW_SURFACES_GetLayerFrameBuffer(window_surfaces_t* self, int_t layer_num, frame_buffer_t** out_frame_buffer);
補 足	フレームバッファの属性を変更したいときは、R_WINDOW_SURFACES_AccessLayerAttributes 関数を使用してください。

引 数	window_surfaces_t* self	フレームバッファと画面表示
	int_t layer_num	レイヤー番号。 0 が最も奥。 +1 が1つ手前。
	frame_buffer_t** out_frame_buffer	(出力) フレームバッファ構造体
リターン値	エラーコード、正常=0	

5.4.6 R_WINDOW_SURFACES_GetLayerCount

概 要	レイヤーの数を取得します。
ヘッダー	RGA.h, window_surfaces.h
宣 言	errnum_t R_WINDOW_SURFACES_GetLayerCount(window_surfaces_t* self, int_t* out_layer_count);
補 足	

引 数	window_surfaces_t* self	フレームバッファと画面表示
	int_t* out_layer_count	(出力) レイヤーの数
リターン値	エラーコード、正常=0	

5.4.7 R_WINDOW_SURFACES_SwapBuffers

概 要	指定したレイヤーのバッファをスワップして、描画していた内容を表示します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_SwapBuffers(window_surfaces_t* self, int_t layer_num, graphics_t graphics);	
補 足	<p>シングルバッファのときは、スワップしません。 本関数を呼ばなくても表示されていますが、その代わり描画途中の様子が表示されてしまいます。</p> <p>ダブルバッファのとき、スワップした直後のフレームバッファの内容は、2 フレーム前の表示内容になっています。R_GRAPHICS_Clear 関数を呼び出すか、表示中のフレームバッファに対して R_GRAPHICS_IMAGE_InitByShareFrameBuffer 関数を呼び出し、R_GRAPHICS_DrawImage 関数で1フレーム前の内容をコピーしてください。</p> <p>Graphics = NULL を指定した場合、バッファをスワップする前に、描画の完了を待たなくなります。</p>	
引 数	window_surfaces_t* self	フレームバッファと画面表示
	int_t layer_num	レイヤー番号。 0 が最も奥。 +1 が1つ手前。
	graphics_t graphics	描画を行ったグラフィックス コンテキスト、NULL 可能
リターン値	エラーコード、正常=0	

5.4.8 R_WINDOW_SURFACES_DoMessageLoop

概 要	メッセージループに入ります。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_DoMessageLoop(window_surfaces_t* self);	
補 足	<p>アプリケーションが終了したら、本関数から返ります。</p> <p>終了方法は、サブクラスの仕様を参照してください。</p>	
引 数	window_surfaces_t* self	フレームバッファと画面表示
リターン値	エラーコード、正常=0	

5.4.9 R_WINDOW_SURFACES_AccessLayerAttributes

概 要	指定した表示レイヤーの属性にアクセスします。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_AccessLayerAttributes(window_surfaces_t* self, layer_attributes_t* in_out_attributes);	
補 足	すべての属性が存在するわけではありません。デバイスやサポート状況によります。	
引 数	window_surfaces_t* self	フレームバッファと画面表示
	layer_attributes_t* in_out_attributes	(入出力) レイヤーの属性 access メンバー変数にリードかライトを指定してください。
リターン値	エラーコード、正常=0)	

5.4.10 R_WINDOW_SURFACES_AllocOffscreenStack

概 要	オフスクリーン バッファを VRAM のスタックから確保します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_AllocOffscreenStack(window_surfaces_t* self, frame_bufer_t* in_out_frame_buffer);	
補 足	<p>入力する frame_bufer_t のメンバー変数は、stride、height、buffer_count です。</p> <p>出力する frame_bufer_t のメンバー変数は、buffer_address 配列の全要素です。</p> <p>メモリー不足になったときは、buffer_count が 0 になり、E_FEW_ARRAY エラーが返ります。</p> <p>R_WINDOW_SURFACES_Finalize が呼ばれたら、確保したオフスクリーン バッファも解放されます。</p>	
引 数	window_surfaces_t* self	フレームバッファと画面表示
	frame_bufer_t* in_out_frame_buffer	(入出力) オフスクリーン バッファ 入力する変数と出力する変数については上記補足を参照。
リターン値	エラーコード、正常=0	

5.4.11 R_WINDOW_SURFACES_FreeOffscreenStack

概 要	オフスクリーン バッファを解放し、VRAM のスタックに返却します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_FreeOffscreenStack(window_surfaces_t* self, frame_bufer_t* frame_buffer);	
補 足	<p>入力する frame_bufer_t のメンバー変数は、buffer_count、buffer_address です。</p> <p>返却する順番が、確保した順番と逆の順番（スタックと同じ順番）でなかったときは、E_ACCESS_DENIED エラーが返ります。 buffer_count が 0 のときは、何もしません。</p>	
引 数	window_surfaces_t* self	フレームバッファと画面表示
	frame_bufer_t* frame_buffer	解放するオフスクリーン バッファ
リターン値	エラーコード、正常=0	

5.4.12 R_WINDOW_SURFACES_SetLayerAttributes

概 要	指定した表示レイヤーの属性を設定します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	errnum_t R_WINDOW_SURFACES_SetLayerAttributes(window_surfaces_t* self, int_fast32_t const in_LayerNum, layer_attributes_t* in_Attributes);	
補 足		
引 数	window_surfaces_t* self	フレームバッファと画面表示
	int_fast32_t const in_LayerNum	設定する対象となる表示レイヤーの番号
	layer_attributes_t* in_Attributes	レイヤーの属性
リターン値	エラーコード、正常=0	

5.4.13 R_WINDOW_SURFACES_CONFIG_SetEmpty

概 要	すべてのメンバー変数の値を空を意味する値で初期化します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	void R_WINDOW_SURFACES_CONFIG_SetEmpty(window_surfaces_config_t* out);	
補 足	空を意味する値については、<Section_ValueMeaningEmptyAndFlaggedParameter>を参照。	
引 数	window_surfaces_config_t* out	メンバー変数の値を変更する構造体
リターン値	なし	

5.4.14 R_LAYER_ATTRIBUTES_SetEmpty

概 要	すべてのメンバー変数の値を空を意味する値で初期化します。	
ヘッダー	RGA.h, window_surfaces.h	
宣 言	void R_LAYER_ATTRIBUTES_SetEmpty(layer_attributes_t* out);	
補 足	空を意味する値については、<Section_ValueMeaningEmptyAndFlaggedParameter>を参照。	
引 数	layer_attributes_t* out	メンバー変数の値を変更する構造体
リターン値	なし	

5.5 byte_per_pixel_t クラスに関連する関数

5.5.1 一覧

章	関数名	概要
5.5.2	R_RGA_BitPerPixelType_To_BytePerPixelType	1 ピクセルあたりのビット数を、1 ピクセルあたりのバイト数（小数部あり）に変換します。
5.5.3	R_RGA_BytePerPixelType_To_BitPerPixelType	1 ピクセルあたりのバイト数（小数部あり）を、1 ピクセルあたりのビット数に変換します。
5.5.4	R_BYTE_PER_PIXEL_IsInteger	1 ピクセルあたりのバイト数が整数かどうかを返します。
5.5.5	R_BIT_PER_PIXEL_GetBytePerPixel	1 ピクセルあたりのバイトから 1 ピクセルあたりのビットへの変換。
5.5.6	R_BYTE_PER_PIXEL_GetBitPerPixel	1 ピクセルあたりのビットから 1 ピクセルあたりのバイトへの変換。
5.5.7	R_RGA_BYTE_PER_PIXEL_IsInteger	1 ピクセルあたりのバイト数が整数かどうかを返します。

5.5.2 R_RGA_BitPerPixelType_To_BytePerPixelType

概 要	1 ピクセルあたりのビット数を、1 ピクセルあたりのバイト数（小数部あり）に変換します。	
ヘッダー	RGA.h	
宣 言	byte_per_pixel_t R_RGA_BitPerPixelType_To_BytePerPixelType(int_t bit_per_pixel);	
補 足		
引 数	int_t bit_per_pixel	1 ピクセルあたりのビット数
リターン値	1 ピクセルあたりのバイト数（小数部あり）	

5.5.3 R_RGA_BytePerPixelType_To_BitPerPixelType

概 要	1 ピクセルあたりのバイト数（小数部あり）を、1 ピクセルあたりのビット数に変換します。	
ヘッダー	RGA.h	
宣 言	int_t R_RGA_BytePerPixelType_To_BitPerPixelType(byte_per_pixel_t byte_per_pixel);	
補 足		
引 数	byte_per_pixel_t byte_per_pixel	1 ピクセルあたりのバイト数（小数部あり）
リターン値	1 ピクセルあたりのビット数	

5.5.4 R_BYTE_PER_PIXEL_IsInteger

概 要	1 ピクセルあたりのバイト数が整数かどうかを返します。	
ヘッダー	RGA.h	
宣 言	bool_t R_BYTE_PER_PIXEL_IsInteger(byte_per_pixel_t byte_per_pixel);	
補 足		
引 数	byte_per_pixel_t byte_per_pixel	1 ピクセルあたりのバイト数（小数部あり）
リターン値	1 ピクセルあたりのビット数が整数かどうか	

5.5.5 R_BIT_PER_PIXEL_GetBytePerPixel

概 要	1 ピクセルあたりのバイトから 1 ピクセルあたりのビットへの変換。	
ヘッダー	RGA.h	
宣 言	byte_per_pixel_t R_BIT_PER_PIXEL_GetBytePerPixel(int_fast32_t in_BitPerPixel);	
補 足		
引 数	in_BitPerPixel	
リターン値	The value as byte_per_pixel_t	

5.5.6 R_BYTE_PER_PIXEL_GetBitPerPixel

概 要	1 ピクセルあたりのビットから 1 ピクセルあたりのバイトへの変換。	
ヘッダー	RGA.h	
宣 言	int_fast32_t R_BYTE_PER_PIXEL_GetBitPerPixel(byte_per_pixel_t in_BytePerPixel);	
補 足		
引 数	in_BytePerPixel	
リターン値	BitPerPixel	

5.5.7 R_RGA_BYTE_PER_PIXEL_IsInteger

概 要	1 ピクセルあたりのバイト数が整数かどうかを返します。	
ヘッダー	RGA.h	
宣 言	bool_t R_RGA_BYTE_PER_PIXEL_IsInteger(byte_per_pixel_t in_BytePerPixel);	
補 足		
引 数	in_BytePerPixel	1 ピクセルあたりのバイト数（小数部あり）
リターン値	1 ピクセルあたりのビット数が整数かどうか	

5.6 animation_timing_function_t クラスに関連する関数

5.6.1 一覧

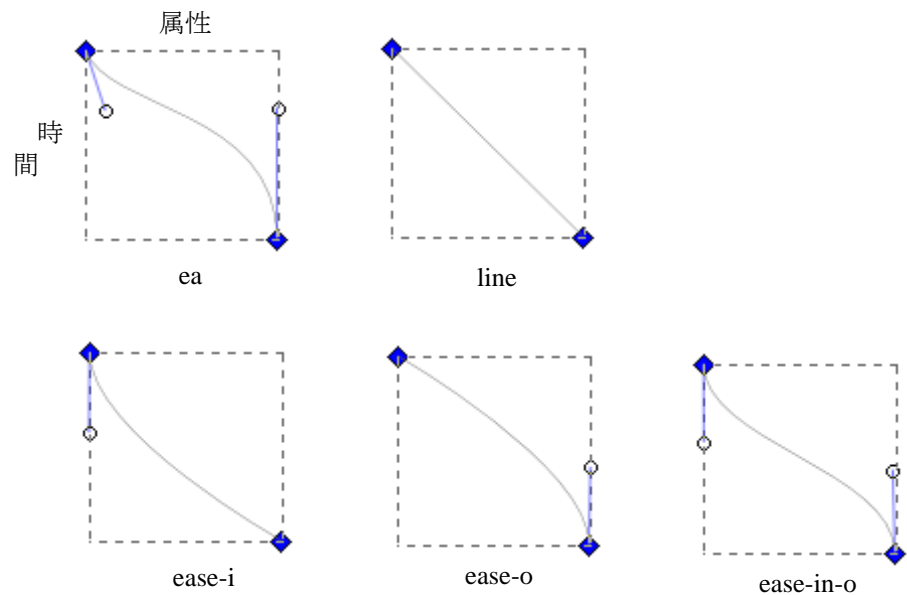
章	関数名	概要
5.6.2	R_Get_AnimationTimingFunction	定義済みのアニメーションタイミングを取得します
5.6.3	R_ANIMATION_TIMING_FUNCTION_GetValue	アニメーションの経過時間における属性の値を計算します

5.6.2 R_Get_AnimationTimingFunction

概 要	定義済みのアニメーションタイミングを取得します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_Get_AnimationTimingFunction(char* timing_name, animation_timing_function_t** out_timing);	
補 足 引 数	char* timing_name	アニメーションタイミングの名前。[条件] 以下を指定できます。 "ease", "linear", "ease-in", "ease-out", "ease-in-out"
	animation_timing_function_t** out_timing	(出力) アニメーションタイミングのオブジェクトのアドレス
リターン値	エラーコード、正常=0	

5.6.3 R_ANIMATION_TIMING_FUNCTION_GetValue

概 要	アニメーションの経過時間における属性の値を計算します。
ヘッダー	RGA.h
宣 言	float32_t R_ANIMATION_TIMING_FUNCTION_GetValue(animation_timing_function_t* self, float32_t clamp_time, float32_t value_of_previous_keyframe, float32_t value_of_next_keyframe);
補 足	属性の値とは、時間によって変化する、座標値や色の値など本関数を使うユーザーが定義した値です。



例 :
timing_name="linear", value_of_previous_keyframe=10,
value_of_next_keyframe=20, clamp_time=0.5
を指定したときのリターン値は 15 です。

引 数	animation_timing_function_t* self	アニメーションタイミング
	float32_t clamp_time	前のキーフレーム (clamp_time=0.0) から後のキーフレーム (clamp_time=1.0) までの経過時間の割合。[条件] 0.0～1.0 の小数
	float32_t value_of_previous_keyframe	前のキーフレームにおける属性の値
	float32_t value_of_next_keyframe	後のキーフレームにおける属性の値
リターン値	clamp_time の時間における属性の値	

5.7 JCU ドライバ OS ポーティングレイヤ

5.7.1 一覧

章	関数名	概要
5.7.2	R_GRAPHICS_OnInitialize	<graphics_config_t> のデフォルト値を設定するコールバック関数
5.7.3	R_GRAPHICS_OnFinalize	<R_GRAPHICS_OnInitialize> 関数の内部で確保したメモリなどを開放します。
5.7.4	R_GRAPHICS_OnInitialized	<R_GRAPHICS_Initialize> の最後から呼ばれるコールバック関数
5.7.5	R_GRAPHICS_JCU_ClearCodecEvent	JCU の割り込み発生イベントをクリアします。
5.7.6	R_GRAPHICS_JCU_SetCodecEvent	JCU の割り込み応答処理の完了をスレッドに通知します。
5.7.7	R_GRAPHICS_JCU_WaitForCodecEvent	JCU の割り込みを待ちます。
5.7.8	R_GRAPHICS_JCU_ClearTerminateEvent	JCU の終了処理イベントをクリアします。
5.7.9	R_GRAPHICS_JCU_SetTerminateEvent	JCU の終了処理の完了をスレッドに通知します。
5.7.10	R_GRAPHICS_JCU_WaitForTerminateEvent	JCU の終了処理の完了を待ちます。

5.7.2 R_GRAPHICS_OnInitialize

概 要	<graphics_config_t> のデフォルト値を設定するコールバック関数。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_OnInitialize(graphics_t* self, graphics_config_t* in_out_Config, void** out_CalleeDefined);	
補 足	out_CalleeDefined 引数の例を以下に挙げます。 - 本関数の中で確保したメモリのアドレス	
引 数	graphics_t* self	初期化を始めるオブジェクトのアドレス（未初期化状態）
	graphics_config_t* in_out_Config,	デフォルト値の設定対象
	void** out_CalleeDefined	本関数が定義した値。内部で確保したメモリなど
	リターン値 エラーコード、正常=0	

5.7.3 R_GRAPHICS_OnFinalize

概 要	<R_GRAPHICS_OnInitialize> 関数の内部で確保したメモリなどを開放します。	
ヘッダー	RGA.h	
宣 言	errnum_t R_GRAPHICS_OnFinalize(graphics_t* self, void* in_CalleeDefined, errnum_t e);	
補 足		
引 数	graphics_t* self	終了処理が完了したオブジェクトのアドレス
	void* in_CalleeDefined	<R_GRAPHICS_OnInitialize> 関数の *out_CalleeDefined 引数の出力値
	errnum_t e	これまでに発生したエラーコード。エラー無し=0
	リターン値 エラーコード、正常=0	

5.7.4 R_GRAPHICS_OnInitialized

概要	<R_GRAPHICS_Initialize> の最後から呼ばれるコールバック関数。	
ヘッダー	RGA.h	
宣言	errnum_t R_GRAPHICS_OnInitialized(graphics_t* self);	
補足		
引数	graphics_t* self	グラフィックス描画コンテキスト
リターン値	エラーコード、正常=0	

5.7.5 R_GRAPHICS_JCU_ClearCodecEvent

概要	JCU の割込み発生イベントをクリアします。	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_JCU_ClearCodecEvent(void);	
補足	本関数は、<R_GRAPHICS_JCU_SetCodecEvent> で設定されたイベントをクリアします。	
引数	なし	
リターン値	なし	

5.7.6 R_GRAPHICS_JCU_SetCodecEvent

概要	JCU の割込み応答処理の完了をスレッドに通知します。	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_JCU_SetCodecEvent(void);	
補足		
引数	なし	
リターン値	なし	

5.7.7 R_GRAPHICS_JCU_WaitForCodecEvent

概要	JCU の割込みを待ちます。	
ヘッダー	RGA.h	
宣言	errnum_t R_GRAPHICS_JCU_WaitForCodecEvent(void);	
補足	本関数は、<R_GRAPHICS_JCU_SetTerminateEvent> で設定されたイベントをクリアします。	
引数	なし	
リターン値	なし	

5.7.8 R_GRAPHICS_JCU_ClearTerminateEvent

概要	JCU の終了処理イベントをクリアします。	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_JCU_ClearTerminateEvent(void);	
補足	本関数は、<R_GRAPHICS_JCU_SetTerminateEvent> で設定されたイベントをクリアします。	
引数	なし	
リターン値	なし	

5.7.9 R_GRAPHICS_JCU_SetTerminateEvent

概要	JCU の終了処理の完了をスレッドに通知します。	
ヘッダー	RGA.h	
宣言	void R_GRAPHICS_JCU_SetTerminateEvent(void);	
補足		
引数	なし	

リターン値	なし
-------	----

5.7.10 R_GRAPHICS_JCU_WaitForTerminateEvent

概 要	JCU の終了処理の完了を待ちます。
ヘッダー	RGA.h
宣 言	errnum_t R_GRAPHICS_JCU_WaitForTerminateEvent(void);
補 足	
引 数	なし
リターン値	エラーコード、正常=0

5.8 DRW ドライバ OS ポーティングレイヤ

5.8.1 一覧

章	関数名	概要
5.8.2	R_DRW_OnInitialize	DRW を初期化するときに呼ばれる関数
5.8.3	R_DRW_OnFinalize	DRW の終了処理をするときに呼ばれる関数
5.8.4	R_DRW_EnableInterrupt	DRW の割込みを許可します
5.8.5	R_DRW_DisableInterrupt	DRW の割込みを禁止します
5.8.6	R_DRW_WaitForInterruptEvent	DRW の割込み発生を待ちます
5.8.7	R_DRW_SetInterruptEvent	DRW の割込み発生をスレッドに通知します
5.8.8	R_DRW_CheckThread	ロックしているスレッドかどうかをチェックします
5.8.9	R_DRW_FlushCache	CPU の L1 キャッシュをフラッシュします
5.8.10	R_DRW_ToPhysicalAddress	物理アドレスに変換します
5.8.11	R_DRW_ToCachedAddress	キャッシュ領域のアドレスに変換します
5.8.12	R_DRW_ToUncachedAddress	非キャッシュ領域のアドレスに変換します
5.8.13	R_DRW_OnInterrupting	DRW の割込みが発生したときの処理を実行します

5.8.2 R_DRW_OnInitialize

概 要	DRW を初期化するときに呼ばれる関数	
ヘッダー	r_drw_pl.h	
宣 言	errnum_t R_DRW_OnInitialize(void);	
補 足	DRW ドライバーは、本関数で以下の処理を行うことを期待します。 DRW のクロック供給を開始する DRW の割込みが入ったときに R_DRW_OnInterrupting 関数を呼び出すようにする DRW の割込みの優先度を設定する	
引 数	なし	
リターン値	エラーコード、正常=0	

5.8.3 R_DRW_OnFinalize

概 要	DRW の終了処理をするときに呼ばれる関数	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_OnFinalize(void);	
補 足	DRW ドライバーは、本関数で以下の処理を行うことを期待します。 DRW のクロック供給を終了する	
引 数	なし	
リターン値	なし	

5.8.4 R_DRW_EnableInterrupt

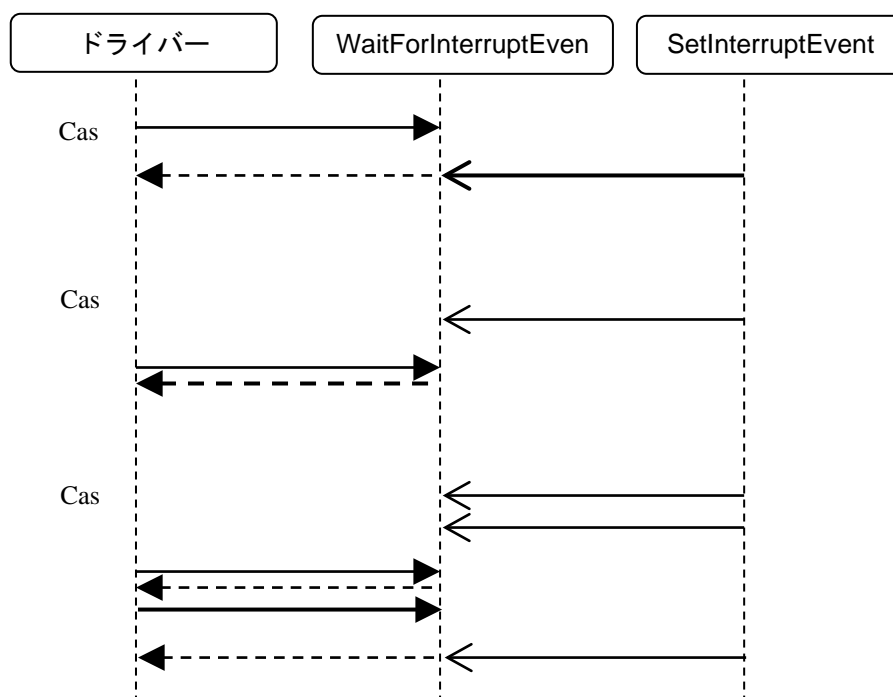
概 要	DRW の割込みを許可します	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_EnableInterrupt(void);	
補 足		
引 数	なし	
リターン値	なし	

5.8.5 R DRW DisableInterrupt

概 要	DRW の割込みを禁止します	
ヘッダー	r_drw_pl.h	
宣 言	bool_t R_DRW_DisableInterrupt(void);	
補 足		
引 数	なし	
リターン値	今まで DRW の割込みを許可していたかどうかの値	

5.8.6 R_DRW_WaitForInterruptEvent

概要	DRW の割込み発生を待ちます
ヘッダー	r_drw_pl.h
宣言	void R_DRW_WaitForInterruptEvent(void);
補足	<p>OS レス環境では、内部でポーリングを行います。RTOS を使う環境の場合、本関数の内部からバイナリー セマフォ、イベント グループ、スレッド付属イベントなどの API を呼び出すコードに変更してください。</p> <p>本関数を呼び出すスレッドは、R_DRW_OnInitialize 関数を呼び出したスレッドと同じスレッドです。</p> <p>本関数は、R_DRW_SetInterruptEvent 関数が呼ばれた後で返ります（下図の Case1）。本関数呼び出す前に R_DRW_SetInterruptEvent 関数が呼ばれていたら、すぐに返ります（下図の Case2）。</p> <p>前回本関数を呼び出したときより前にだけ R_DRW_SetInterruptEvent 関数が呼ばれていたら、何回呼ばれていても、すぐに返りません。その仕様を満たさなければ、2 つ以上の要因が連続して発生したときに、次の待ちで待たなくなってしまいます。（下図の Case3）。</p>



引 数	なし	
リターン値	なし	

5.8.7 R_DRW_SetInterruptEvent

概 要	DRW の割込み発生をスレッドに通知します	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_SetInterruptEvent(void);	
補 足	<p>本関数は、R_DRW_WaitForInterruptEvent 関数を呼び出すことで待っている RTOS のスレッドに通知します。</p> <p>割込みから本関数が呼び出されるときは、R_DRW_OnInterrupting 関数の内部から呼び出されます。</p> <p>割込みからだけでなく任意のスレッドからでも、本関数を呼び出します。</p> <p>本関数を連続して呼び出しても、R_DRW_WaitForInterruptEvent 関数から連続して返ることはありません。</p>	
引 数	なし	
リターン値	なし	

5.8.8 R_DRW_CheckThread

概 要	ロックしているスレッドかどうかをチェックします	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_CheckThread(void);	
補 足	<p>本関数を呼び出したスレッドが、R_DRW_OnInitialize 関数を呼び出したスレッド（DRW をロックしたスレッド）と異なるスレッドであるなら、排他制御に関する問題が発生する可能性があります。</p> <p>スレッド間通信などを使って、RGA の API を呼び出すスレッドを 1 つにしてください。</p> <p>OS レス環境では、本関数では何もしません。</p>	
引 数	なし	
リターン値	なし	

5.8.9 R_DRW_FlushCache

概 要	CPU の L1 キャッシュをフラッシュします	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_FlushCache(void* in_StartAddress, uint32_t in_Size);	
補 足	<p>本関数のフラッシュ操作は、ライトバックして破棄することです。</p> <p>引数を見捨て、キャッシュ全体をフラッシュしても構いません。</p>	
引 数	void* in_StartAddress	フラッシュする範囲の開始アドレス
	uint32_t in_Size	フラッシュする範囲のサイズ
リターン値	なし	

5.8.10 R_DRW_ToPhysicalAddress

概 要	物理アドレスに変換します
ヘッダー	r_drw_pl.h
宣 言	errnum_t R_DRW_ToPhysicalAddress(void* in_Address, uintptr_t* out_PhysicalAddress);
補 足	<p>例：物理アドレスが整数型の場合</p> <pre>uintptr_t physical_address; e= R_DRW_ToPhysicalAddress(address, &physical_address); if(e){goto fin;}</pre>

例：物理アドレスがポインター型の場合

```
uintptr_t physical_address;
void* pointer;
e= R_DRW_ToPhysicalAddress( address, &physical_address );
if(e){goto fin;}
pointer = (void*) physical_address;
```

本関数の定義は、MMU の設定に合わせて変更してください。

引 数	void* in_Address	仮想アドレス
	uintptr_t* out_PhysicalAddress	(出力) 物理アドレス
リターン値	エラーコード、正常=0	

5.8.11 R_DRW_ToCachedAddress

概 要

ヘッダー

宣 言

補 足

引 数

リターン値

キャッシュ領域のアドレスに変換します

r_drw_pl.h

errnum_t R_DRW_ToCachedAddress(void* in_Address, void* out_CachedAddress);

本関数の定義は、MMU の設定に合わせて変更してください。

out_CachedAddress 引数は、ポインターへのアドレスを指定してください。

ミラー領域がない場合、in_Address に非キャッシュ領域のアドレスを指定すると、エラーになります。

エラーになったとき、in_Address 引数に渡されたアドレスに対応する変数が何かは、map ファイルから分かるかもしれません。

ミラー領域がないメモリーマップにした環境では、アドレス変換の関数を呼び出す必要はありませんが、指定したアドレスがキャッシュ領域か非キャッシュ領域かをチェックすることに使えます。

void* in_Address

void* out_CachedAddress

エラーコード、正常=0

仮想アドレス

(出力) キャッシュ領域のアドレス

5.8.12 R_DRW_ToUncachedAddress

概 要	非キャッシュ領域のアドレスに変換します	
ヘッダー	r_drw_pl.h	
宣 言	errnum_t R_DRW_ToUncachedAddress(void* in_Address, void* out_UncachedAddress);	
補 足	<p>本関数の定義は、MMU の設定に合わせて変更してください。</p> <p>out_UncachedAddress 引数は、ポインターへのアドレスを指定してください。</p> <p>ミラー領域がない場合、in_Address に非キャッシュ領域のアドレスを指定すると、エラーになります。</p> <p>エラーになったとき、in_Address 引数に渡されたアドレスに対応する変数が何かは、map ファイルから分かるかもしれません。</p> <p>ミラー領域がないメモリーマップにした環境では、アドレス変換の関数を呼び出す必要はありませんが、指定したアドレスがキャッシュ領域か非キャッシュ領域かをチェックすることに使えます。</p>	
引 数	void* in_Address	仮想アドレス
	void* out_UncachedAddress	(出力) 非キャッシュ領域のアドレス
リターン値	エラーコード、正常=0	

5.8.13 R_DRW_OnInterrupting

概 要	DRW の割込みが発生したときの処理を実行します	
ヘッダー	r_drw_pl.h	
宣 言	void R_DRW_OnInterrupting(void);	
補 足	DRW の割込みが発生したときに本関数を呼び出すように、移植層をコーディングしてください。	
引 数	なし	
リターン値	なし	

5.9 その他の関数

5.9.1 一覧

章	関数名	概要
5.9.2	R_RGA_Get_R8G8B8A8	R8G8B8A8 カラーの値を返します。
5.9.3	R_RGA_CalcWorkBufferB_Size	ワークバッファBに必要なサイズを計算します。

5.9.2 R_RGA_Get_R8G8B8A8

概 要	R8G8B8A8 カラーの値を返します。	
ヘッダー	RGA.h	
宣 言	r8g8b8a8_t R_RGA_Get_R8G8B8A8(int_t red, int_t green, int_t blue, int_t alpha);	
補 足		
引 数	int_t red, int_t green, int_t blue, int_t alpha	各色成分。[設定範囲] 0～255
リターン値	R8G8B8A8 カラーの値	

5.9.3 R_RGA_CalcWorkBufferB_Size

概 要	ワークバッファBに必要なサイズを計算します。
ヘッダー	RGA.h
宣 言	size_t R_RGA_CalcWorkBufferB_Size(int_t MaxWidthOfJPEG, int_t MaxHeightOfJPEG, int_t MaxBytePerPixelOfFrameBuffer);
補 足	将来、パラメーターが変わる可能性があります。 R_RGA_CalcWorkBufferB_Size は #define マクロです。

ワークバッファBに必要なサイズ : $\text{ceil_16}(\text{MaxWidthOfJPEG}) * \text{ceil_16}(\text{MaxHeightOfJPEG}) * \text{MaxBytePerPixelOfFrameBuffer}$ [Bytes]
ただし、ceil_16 は、16 の倍数に切り上げ。

JPEG/PNG を描画しないときは、ワークバッファB を 0 にできます。

PNG を伸張するときは、LibPNG のワーク領域（サイズは work_size_for_libPNG メンバー変数に指定）もワークバッファB から確保しますが、R_RGA_CalcWorkBufferB_Size マクロは、このサイズを含みません。

JPEG コーデックユニット (JCU) (RZ/A2M に内蔵の周辺機能) が直接出力できる下記の条件をすべて満たすとき、または、JPEG 画像を描画しないときは、ワークバッファB は不要です。

JPEG 画像の左上の位置の描画対象のアドレスが 8 で割り切れるとき。

JPEG 画像のサイズが MCU (Minimum Coded Unit) の倍数のとき。そのサイズは、JPEG データのピクセルフォーマットによる

JPEG 画像が YCbCr422 のとき、16 ピクセル×8 ラインの倍数

JPEG 画像が YCbCr420 のとき、16 ピクセル×16 ラインの倍数

JPEG 画像が YCbCr444 のとき、8 ピクセル×8 ラインの倍数

JPEG 画像が YCbCr411 のとき、32 ピクセル×8 ラインの倍数

行列が単位行列または平行移動のとき

リターン値は、graphics_config_t 型に指定します。

引 数	int_t MaxWidthOfJPEG	JPEG 画像の最大の幅。[設定範囲] 0 以上
	int_t MaxHeightOfJPEG	JPEG 画像の最大の高さ。[設定範囲] 0 以上
	int_t MaxBytePerPixelOfFrameBuffer	描画対象になるフレームバッファの最大の色のバイト数。 ただし、行列が単位行列でも平行移動でもないときは、4。
リターン値	ワークバッファBに必要なサイズ (バイト)	

6. ツール説明

6.1 画像フォーマット変換 ImagePackager

複数の画像ファイルを、1つのバイナリ ファイル（実機向け）、または、ソース（実機/PC 向け）にまとめます。まとめる際に、画像ファイルは、任意のピクセルフォーマットの Raw 形式に変換することもできます。まとめたものは、バイナリ、または、C 言語のデータのソースになります。XML ファイルに、まとめる画像ファイルを指定します。ファイル名および拡張子は英文字小文字を区別します。

ImagePackager は、RGA_Tools.vbs の中の 1 つのコマンドです。内部に必要な vbs ファイルや exe ファイルを呼び出しています。

6.1.1 操作手順

1. .image.xml ファイル（6.1.3）を作成します
2. \${RGA}¥src¥renesas¥application¥tool にある RGA_Tools.vbs ファイルをダブルクリックして開いたウィンドウから ImagePackager コマンドを選び、.image.xml ファイルを指定すると、ヘッダーファイルと、バイナリ、または、C 言語のデータのソースができます。

```
-----  
RGA Tools - Copyright(c) 2012-2015 Renesas Electronics Corporation  
  1. 画像フォーマット変換 [RunImagePackager]  
番号またはコマンド >1  
-----  
((( [RunImagePackager_sth] )))  
Renesas Image Packager - Copyright(c) 2012-2015 Renesas Electronics Corporation  
複数の画像などのファイルから、1つのバイナリ ファイルを作成します。  
Enter のみ：サンプル・フォルダーを開く  
設定ファイル(ImagePackagerConfig.xml) >
```

3. できたヘッダーファイルを #include し、その中で定義されている画像のシンボルを R_GRAPHICS_DrawImage 関数などに指定してください。
4. バイナリを出力したときは、.image.xml ファイルに指定したアドレスにバイナリを配置してから、画像の描画処理をしてください。アドレスは、.image.xml ファイルの /ImagePackager/OutputBinary/OutputHeader/@address に指定したアドレスです。できたヘッダーファイルにも書かれています。

ImagePackager は、コマンドプロンプトから、以下のように入力して起動することもできます。

```
>cd src¥renesas¥application¥RGA_Canvas2D¥Images  
>cscript //nologo ..¥..¥RGA_Tools.vbs RunImagePackager  
BinaryImageConfig.image.xml
```

6.1.2 ファイル一覧

ファイル	内容
RGA_Tools.vbs	ImagePackager を含むスクリプト・ファイル
*.image.xml	ImagePackager の設定ファイル
(*.bmp, *.jpg, *.png)	入力画像ファイル
(出力：バイナリ ファイル)	ターゲットボードにダウンロードする、複数の画像やファイルをまとめたファイル
(出力：ソース・コード)	プログラムの中のデータにする、複数の画像やファイルをまとめたファイル

6.1.3 サンプル

BinaryImageConfig.image.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<ImagePackager>

<OutputBinary path="BinaryImage_RZ_A1H.c" language="C"
symbol="g_RGA_Sample_BinaryImage"
    source_template="$ {ImagePackagerLib}¥SourceTemplate.xml#default"
    super_class="$ {ImagePackagerLib}¥SuperClass.xml#default">
<OutputHeader path="BinaryImage_RZ_A1H.h"
include_define="BINARYIMAGE_RZ_A1H_H"/>
</OutputBinary>

<InputFiles>
<File path="BinaryHeader.txt" type="char[]" symbol="g_BinaryHeader"/>
<Image path="picture.bmp" type="graphics_image_t*" symbol="g_Picture_bmp"
output_format="RGB565"/>
<Image path="smile32.bmp" type="graphics_image_t*" symbol="g_Smile_bmp"
output_format="ARGB4444"/>
<File path="JPEG.jpg" type="graphics_image_t*" symbol="g_JPEG_jpg"/>
</InputFiles>

</ImagePackager>

```

XML や XPath の基本的な書き方は、(6.1.8) を参照してください。

相対パスは、XML ファイルがあるフォルダーが基準です。

Image/@path に、フォルダーのパスや、ワイルドカードも指定できます。

先頭に、固定値のヘッダーを入れておけば、バイナリが入っているかどうかをチェックできます。

6.1.4 出力バイナリの種類（言語）

- ・バイナリ形式（C 言語の外部参照シンボル） + C 言語ヘッダー
- ・バイナリ形式・S レコード形式（フラッシュなどのアドレス直接） + C 言語ヘッダー
- ・C 言語ソース + C 言語ヘッダー

/ImagePackager/OutputBinary/@language や /ImagePackager/OutputHeader/@address に指定

6.1.5 出力バイナリ内のファイルフォーマット

Format	Description
オフセット テーブル	画像データやバイナリ データへの（出力バイナリの先頭からの）オフセットが、4 バイト整数型で、出力バイナリの先頭に並んでいます。
Raw 形式画像	/ImagePackager/InputFiles/Image/@output_format に指定 "ARGB8888", "XRGB8888"(X 成分は 0x00), "RGB565", "ARGB1555", "ARGB4444", "YUV422", "CLUT8", "CLUT4", "CLUT1", "A8", "A4", "A1" 詳しくは、下記「Raw 形式画像のフォーマット」を参照。
バイナリ形式	JPEG をターゲットボードで伸長するときなど、ファイルをそのまま格納。 /ImagePackager/InputFiles/File に指定

Raw 形式画像のフォーマット :

オフセット	サイズ	説明
0x00	4byte	画像の種類を表すビットフラグ [bit0] 1 固定 [bit1] (予約領域) [bit2] 1=Premultiplied alpha。0=非 Premultiplied alpha [bit3~bit15] (予約領域) [bit16~bit31] 値は 0 @endian の設定によって、エンディアンが異なる。
0x04	4byte	Raw 形式画像の先頭から、画像データへのオフセット @endian の設定によって、エンディアンが異なる。
0x08	4byte	(予約領域)
0x0C	2byte	画像の幅 (ピクセル) @endian の設定によって、エンディアンが異なる。
0x0E	2byte	画像の高さ (ピクセル) @endian の設定によって、エンディアンが異なる。
0x10	1byte	画像の種類 0=RGB565、2=ARGB1555、3=ARGB4444、6=CLUT8、7=CLUT4、8=CLUT1、 9=XRGB8888、10=ARGB8888、11=YUV422、20=A8、21=A4、22=A1
0x11	7byte	(予約領域)
*		画像データ @endian の設定によって、RGB 系はエンディアンが異なる。YUV 系は @endian の設定によらない。 @raw_image_alignment の設定によって、画像データのオフセットの値が異なる。 @raw_stride_alignment などの設定によって、1 行あたりのバイト数が異なる。 CLUT (Color Look Up Table、パレット) があるときは、CLUT の後に画像データが続く。CLUT の要素は、ARGB8888 形式で A=0xFF 固定、@endian の設定によってエンディアンが異なる。要素数は、CLUT8 なら 256、CLUT4 なら 16、CLUT1 なら 2。 RGA の制限 : RGA で描画する画像データの先頭アドレスは、32 の倍数にする必要があります。 画像のそれぞれの行末にパディングは入れないでください。

6.1.6 入力フォーマット

- ・ BMP 形式 : 32 ビット、24 ビット、CLUT 用に 256/16/2 色
- ・ PNG 形式 : アルファ付き対応
- ・ JPEG 形式

“/ImagePackager/InputFiles/Image” の @path の説明を参照してください。

6.1.7 BinaryImageConfig.image.xml に記述できるパラメーター

/ImagePackager/OutputBinary: 1 つ以上。複数は、@language="Binary" と "C" の両方を出力するときなど

@path	バイナリ ファイルの出力先のパス。 必須。
@symbol	バイナリ ファイル全体に対応する C 言語の外部参照シンボル（グローバル変数名）。 必須。 複数のバイナリ ファイルを出力するときは、それぞれのバイナリで変えないでください。/ImagePackager/OutputHeader/@address を設定したときは、アドレスを値に持つマクロ名になります。
@language	出力されるバイナリデータのプログラミング言語。 オプション。 "Binary", "C", "SRec"(S-record) が設定可能。 デフォルトは "Binary"。 この設定を変更したら、@table_format のデフォルトが変わるため、画像データを参照するソースを再コンパイルしてください。
@section	セクションの指定を出力ソースに埋め込みます。 例: section="BinaryImage" ... BinaryImage セクションができます。/ImagePackager/SourceTemplate の中の \${Section} に埋め込まれます。
@endian	エンディアン。 オプション。 "LittleEndian", "BigEndian" が設定可能。 デフォルトは @super_class が指す先の @endian。
@raw_image_alignment	Raw 形式の画像データの先頭アドレスのアラインメント（バイト）。 オプション。 [設定範囲] 2 の n 乗のみ設定可能。 デフォルトは @super_class が指す先の @raw_image_alignment。複数のバイナリ ファイルを出力するときは、それぞれのバイナリで変えないでください。 Raw 形式のヘッダー部分は、この設定ではアラインメントされません。
@raw_image_alignment_symbol	@raw_image_alignment の値が格納された #define シンボル名。 オプション。 生成されるヘッダーファイルに、#define シンボル名が出力されるので、そのヘッダーを使って、バイナリを使うプログラムをコンパイルしてください。 例: raw_image_alignment_symbol="GRAPHICS_RAW_IMAGE_ALIGNMENT" ヘッダーファイルに入るコード: #define RAW_IMAGE_ALIGNMENT 1 デフォルトは、シンボル名の #define 文を出力しない。 複数のバイナリ ファイルを出力するときは、それぞれのバイナリで変えないでください。
@raw_stride_alignment	出力する画像の中の、1 行下の行までのバイト数のアラインメン

	<p>ト。オプション。(value) に指定した値の倍数に、バイト数が繰り上がります。</p> <p>デフォルトは、@super_class が指す先の @raw_stride_alignment です。</p> <p>例: raw_stride_alignment="1"</p> <p>ハード描画版 RGA は、RGA 3.xx から、32 から 1 に変更されました。1 は、行末にパディングを入れないという意味です。</p> <p>[設定範囲] 1 以上</p>
@raw_stride_alignment_4	<p>1 行下の行までのバイト数のアラインメントを 4 にする出力画像のピクセルフォーマット。CSV 形式で複数指定可能。オプション。</p> <p>本設定は、@raw_stride_alignment の設定より優先します。デフォルトは、@super_class が指す先の @raw_stride_alignment_4。</p> <p>例: raw_stride_alignment_4="YUV422"</p>
@alpha_raw_image_width	<p>A8,A4,A1 の Raw 形式の画像を出力するときの幅 (ピクセル)。オプション。</p> <p>このオプションは、A8,A4,A1 の Raw 形式のソース画像の幅を描画対象の幅に合わせる必要がある RGA の制限事項に対応するためのオプションです。入力画像の幅に関わらず、指定された幅で出力します。デフォルトは、入力画像の幅と出力画像の幅を同じにします。入力画像より幅が小さい指定をしたときはエラーになります。A8,A4,A1 以外の形式ではこのオプションを無視します。</p>
@table_format	<p>テーブルのフォーマット。オプション。</p> <ul style="list-style-type: none"> ● "Offset": 画像やファイルのサイズが変わっても、再コンパイルが不要になるように、バイナリ ファイルに実体だけでなく実体へのオフセットのテーブルも埋め込みます。 <p>テーブルも埋め込んだ場合、画像のサイズが変わっても再コンパイルする必要はありません。また、XML タグの変更があったときでも、その変更より上に記述されている XML タグが生成する変数に関しても、再コンパイルする必要はありません。</p> <ul style="list-style-type: none"> ● "Embed": 画像やファイルをそのまま順次埋めていきます。 <p>デフォルトは、次の通り。</p> <ul style="list-style-type: none"> ● @language="Binary" なら @table_format="Offset" ● @language="C" なら @table_format="Embed"
@source_template	<p>@language="C" としたときに出力されるソースファイルのテンプレートが書かれた ID。オプション。</p> <p>/ImagePackager/SourceTemplate/@id の値を指定します。</p> <p>他の XML ファイルの値を参照することもできます。</p> <p>\${ImagePackagerLib} を指定すると、ImagePackagerLib.vbs があるフォルダーのパスに置き換わります。</p> <p>例: source_template="\${ImagePackagerLib}¥SourceTemplate.xml#default"</p>
@super_class	<p>バイナリ ファイルに適用するスーパークラスの ID。オプション。</p> <p>/ImagePackager/SuperClass/@id の値を指定します。</p> <p>他の XML ファイルの値を参照することもできます。</p>

	<p><code>\${ImagePackagerLib}</code> を指定すると、<code>ImagePackagerLib.vbs</code> があるフォルダーのパスに置き換わります。</p> <p>例： <code>super_class="\${ImagePackagerLib}¥SuperClass.xml#default"</code></p>
--	---

/ImagePackager/OutputBinary/OutputHeader： 1 つ以上。

/ImagePackager/OutputHeader： 0 または 1 つ。

@path	ヘッダーファイルの出力先のパス。 必須。
@include_define	ヘッダーファイルを二重インクルードしないための #define シンボル名。 オプション。 デフォルトは、"__BINARY_IMAGE__"
@address	バイナリ ファイルを配置するメモリアドレス。 オプション。 0x00000000 形式で指定してください。リンクするプログラムイメージとは別に、直接フラッシュ等に配置するときに使います。生成されるヘッダーファイルに、設定したアドレスが出力されるので、そのヘッダーを使って、バイナリデータを参照するプログラムをコンパイルしてください。省略すると、 /ImagePackager/OutputBinary/@symbol に指定した C 言語の外部参照シンボル（グローバル変数名）を使うようになります。

/ImagePackager/InputFiles : 1 つのみ

@base_folder	/ImagePackager/InputFiles/Image/@path の基準となるパス。オプション。 相対パスの基準は、BinaryImageConfig.image.xml ファイルがあるフォルダー。デフォルトは "."
--------------	---

/ImagePackager/InputFiles/Image : 複数可能

Raw 形式の画像データに変換してバイナリ ファイルに埋め込みます。ターゲットボードで JPEG, PNG データを伸張するときは、File タグを使ってください。

@path	入力する画像ファイルのパス。必須。 相対パスの基準は、/ImagePackager/InputFiles/@base_folder フォルダのパスや、ワイルドカードを指定したときは、サブフォルダーも入力します。拡張子が、png または jpg のときは、非圧縮に伸長したものがバイナリに出力されます。ターゲットボード上で展開するときは、/ImagePackager/InputFiles/File に JPEG ファイルを指定してください。アルファチャンネルが無い画像、またはアルファ値がすべて 0xFF の画像は、アルファブレンディングが必要ないという情報が、出力に埋め込まれます。この情報によって高速に描画できる可能性があります。
@type	ヘッダーファイルに記述されるシンボルの型。必須。 @symbol の型。通常、graphics_image_t* を指定します。 uint8_t[] を指定すると、画像データの前にある 0x18 バイト長のヘッダーは、なくなります。
@symbol	ヘッダーファイルに記述されるシンボル。必須。 グローバルスコープにある #define シンボル。@type 型 \${...} を使うとファイル名などに置き換わります。 例 : g_\${BaseName}_\${Extension}_\${Format} → g_Sample_jpg_ARGB8888

@output_format	<p>出力する画像ファイルのフォーマット。 必須。</p> <p>24 ビット/32 ビット Windows ビットマップファイル、JPEG ファイル、PNG ファイル（アルファあり・なし）に対して指定できる値は、"ARGB8888", "XRGB8888"(X 成分は 0x00), "RGB565", "ARGB1555", "ARGB4444", "YUV422", "A8", "A4", "A1" です。</p> <p>256 色、16 色、モノクロの Windows ビットマップファイルに対して指定できる値は、"CLUT8", "CLUT4", "CLUT1", "A8", "A4", "A1" です。"CLUT8"を指定した場合は、256 色の、"CLUT4"を指定した場合は、16 色の、"CLUT1"を指定した場合は、モノクロの Windows ビットマップファイルを入力してください。</p> <p>出力する画像のアルファ成分の値について下記の表に示します。入力画像が 256 色、16 色、モノクロの Windows ビットマップファイルの形式の場合、CLUT を参照した後の色を入力画像の色として下記の表を参照してください。PNG 形式を Windows 7 のペイントブラシで作成すると、常にアルファありになるので注意してください。</p> <table><tr><th>入力画像</th><th>出力画像</th><th>出力の A 成分</th></tr><tr><td>A あり</td><td>任意</td><td>入力の A 成分</td></tr><tr><td rowspan="2">A なし</td><td>ARGB</td><td>0xFF</td></tr><tr><td>A 成分のみ</td><td>入力を RGB→YCbCr 変換した Y 成分（輝度）</td></tr></table> <p>CSV 形式で複数指定することができます。 ただし、そのときは @symbol に \${Format} を含めてください。 \${Format}が出力された画像ファイルのフォーマットの名前に置き換わります。</p>	入力画像	出力画像	出力の A 成分	A あり	任意	入力の A 成分	A なし	ARGB	0xFF	A 成分のみ	入力を RGB→YCbCr 変換した Y 成分（輝度）
入力画像	出力画像	出力の A 成分										
A あり	任意	入力の A 成分										
A なし	ARGB	0xFF										
	A 成分のみ	入力を RGB→YCbCr 変換した Y 成分（輝度）										
@premultiplied_alpha	<p>RGB 成分を Alpha 成分で乗算済みにするかどうか。 オプション</p> <ul style="list-style-type: none">● "no" : 乗算しない（デフォルト）● "yes": 乗算済みに変換して出力する。 描画対象にアルファ成分が無いときのみ可能● "already_yes": 入力画像が乗算済み。 描画対象にアルファ成分が無いときのみ可能											

/ImagePackager/InputFiles/File : 複数可能

ファイルをそのまま埋め込みます。

@path	<p>入力するファイルのパス。 必須。 相対パスの基準は、/ImagePackager/InputFiles/@base_folder ワイルドカードを指定したときは、サブフォルダーも入力します。</p>
@type	<p>ヘッダーファイルに記述されるシンボルの型。 オプション @symbol の型。 ファイルの内容が配列のときは、型名の後に [] を付けてください。 デフォルトは uint8_t[]。</p>
@symbol	<p>ヘッダーファイルに記述されるシンボル。 必須。 グローバルスコープにある #define シンボル。 uint8_t[] 型</p>

@alignment	ファイルの先頭を配置するアドレスのアラインメント。オプション。 デフォルトは、/ImagePackager/OutputBinary/@super_class が指す先の @alignment。 [設定範囲] 1 以上
------------	---

/ImagePackager/InputFiles/Var : 複数可能

XML に記述した値を埋め込みます。

@type	ヘッダーファイルに記述されるシンボルの型。必須。 指定できる型は、int32_t, uint32_t, int16_t, uint16_t, int8_t, uint8_t, void**, void* 以外のポインター型です。 ポインター型のときは、@value にポインターのターゲットがある位置（オフセット）を設定してください。どの型のポインター型でも、4 バイトの値がバイナリに埋め込まれます。@symbol に指定したポインター型のシンボルを C 言語で使うと、その値はアドレスになります。
@symbol	ヘッダーファイルで記述されるシンボルの名前。必須。 ここに指定した名前が、バイナリ ファイルに埋め込まれる値を初期値としたグローバル変数の名前になります。
@value	バイナリ ファイルに埋め込む値。必須。 バイナリを ROM に配置すると値は定数になり、RAM に配置すると変数になります。 整数か下記の特特殊形が指定できます。 <ul style="list-style-type: none"> ● 整数の例 : "10", "-10", "0xFF" ● 特殊形 "(new Image('file_path')).width" : file_path の画像の幅 ● 特殊形 "(new Image('file_path')).height" : file_path の画像の高さ ● 特殊形 "symbol.offset" : シンボルがある位置（オフセット） file_path には、画像ファイルのパスを指定してください。 symbol には、Image, File, Var タグの @symbol に設定したシンボルを指定してください。 エンディアンは、@endian の設定によります。

/ImagePackager/SourceTemplate/ : 0, 1 または 複数

@id	SourceTemplate タグの ID。必須。 /ImagePackager/OutputBinary/@source_template から参照されます。
Source/text()	ソースファイルのテンプレート。必須。 テンプレートの中に入れることができるタグは次の通り。 <ul style="list-style-type: none"> ● \${Section} : セクション名。 /ImagePackager/OutputBinary/@section の値 ● \${Symbol} : 変数名。/ImagePackager/OutputBinary/@symbol の値 ● \${Size} : バイナリ ファイルのサイズ（バイト）

	<ul style="list-style-type: none"> ● <code>#{BinaryData}</code>: バイナリデータ ●
<code>SourceWithSection/text()</code>	<p>セクション指定があるときのソースファイルのテンプレート。オプション。</p> <p><code>SourceWithSection</code> タグが省略されたときは、セクション指定があっても <code>Source</code> タグの内容が使用されます。</p>
<code>Header/text()</code>	<p>ヘッダーファイルのテンプレート。必須。</p> <p>テンプレートの中に入れることができるタグは次の通り。</p> <ul style="list-style-type: none"> ● <code>#{include_define}</code>: 二重インクルード防止のマクロ名。 /ImagePackager/OutputBinary/OutputHeader/@include_define の値 ● <code>#{DeclareBinaryImageSymbol}</code>: バイナリデータのシンボルの宣言。 <code>DeclareVariable/text()</code> または <code>DeclareAddress/text()</code> の内容と、<code>@raw_image_alignment_symbol</code> などによる <code>#define</code> 定義。 ● <code>#{Variables}</code>: 変数に相当する <code>#define</code> の一覧 ● <code>#{Section}</code>: セクション名。 /ImagePackager/OutputBinary/@section の値 ● <code>#{Symbol}</code>: 変数名。/ImagePackager/OutputBinary/@symbol の値 ● <code>#{Size}</code>: バイナリ ファイルのサイズ (バイト) ● <code>#{StartAddress}</code>: バイナリの先頭アドレス ● <code>#{LastAddress}</code>: バイナリの最終アドレス
<code>HeaderWithSection/text()</code>	<p>セクション指定があるときのヘッダーファイルのテンプレート。オプション。</p> <p><code>HeaderWithSection</code> タグが省略されたときは、セクション指定があっても <code>Header</code> タグの内容が使用されます。</p>
<code>DeclareVariable/text()</code>	<p><code>@language="C"</code> のときの <code>#{DeclareBinaryImageSymbol}</code> に入れるテンプレート。</p> <p>テンプレートの中に入れることができるタグは次の通り。</p> <ul style="list-style-type: none"> ● <code>#{Section}</code>: セクション名。 /ImagePackager/OutputBinary/@section の値 ● <code>#{Symbol}</code>: 変数名。/ImagePackager/OutputBinary/@symbol の値 ● <code>#{Size}</code>: バイナリ ファイルのサイズ (バイト)
<code>DeclareAddress/text()</code>	<p><code>@language="Binary"</code> のときの <code>#{DeclareBinaryImageSymbol}</code> に入れるテンプレート。</p> <p>テンプレートの中に入れることができるタグは次の通り。</p> <ul style="list-style-type: none"> ● <code>#{Section}</code>: セクション名。 /ImagePackager/OutputBinary/@section の値 ● <code>#{Symbol}</code>: 変数名。/ImagePackager/OutputBinary/@symbol の値 ● <code>#{StartAddress}</code>: バイナリの先頭アドレス ● <code>#{LastAddress}</code>: バイナリの最終アドレス

/ImagePackager/SuperClass : 0, 1 または 複数

@id	SuperClass タグの ID /ImagePackager/OutputBinary/@super_class から参照されます。
-----	---

/ImagePackager/SuperClass/OutputBinary : 0 または 1

@endian	/ImagePackager/OutputBinary/@endian のデフォルト値。デフォルトは "LittleEndian"。
@raw_image_alignment	/ImagePackager/OutputBinary/@raw_image_alignment のデフォルト値。デフォルトは 4。
@raw_stride_alignment	/ImagePackager/OutputBinary/@raw_stride_alignment のデフォルト値。デフォルトは、1。
@raw_stride_alignment_4	/ImagePackager/OutputBinary/@raw_stride_alignment_4 のデフォルト値。デフォルトは、""

/ImagePackager/SuperClass/InputFiles/File : 0, 1 または 複数

@path	SuperClass を適用する対象となるファイル。ワイルドカードを使用可能。 必須。 例 : path="*.jpg"
@alignment	/ImagePackager/InputFiles/File/@alignment のデフォルト値。デフォルトは 4。

6.1.8 XML の基本的な書き方

本節では、ツールに設定するデータ形式である XML、XPath、#フラグメント の基本的な書き方を説明します。

(1) XML

XML ファイルはテキスト エディターで編集ができるテキスト ファイルの一種です。ここでは、基本的な書き方のみ示します。

XML ファイルの先頭には下記の XML 宣言を記述します。XML 宣言が省略されたファイルは、一般に encoding="UTF-8" の文字コードセットになります。以下はサンプルです。

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

XML のタグは、< と > で囲み、開始タグと終了タグのペアにします。終了タグは開始タグの名前の先頭にスラッシュを追加したものです。プログラムが対応していない名前のタグは無視され、スペルミスがあっても警告されません。タグ名に大文字と小文字の違いがあると、別の名前として扱われます。

```
<Root>  
</Root>
```

XML のタグは、木構造にする必要があります。また、最も根元のタグ（ルート）は複数にできません。なお、タグの間の改行は、空白やタブ文字に変えても、タグの間を削除しても、プログラムに渡すデータは変わりません。

```
<Root>  
    <LeafA></LeafA>  
    <LeafB></LeafB>  
</Root>
```

開始タグと終了タグの間に何も無いときは、名前の末尾にスラッシュを追加した 1 つのタグに置き換えることができます。このとき、プログラムに渡すデータは変わりません。

```
<Root>  
    <LeafA/>  
    <LeafB/>  
</Root>
```

開始タグと終了タグの間にテキストを入れることができます。プログラムが定義するタグの仕様によっては、このテキストが、設定値になります。大文字と小文字の違いがあるテキストが別のデータとして扱われるかどうかは、タグの仕様によります。

```
<Root>  
    <LeafA>ABC</LeafA>  
    <LeafB>DEF</LeafB>  
</Root>
```

開始タグの中に属性の形で、設定値を記述することができます。属性の値（イコールの右）は、必ず " " または ' ' で囲む必要があります。" " と ' ' に違いはありません。終了タグには属性を記述できません。指定できる属性は使用するプログラムのタグの仕様によります。仕様のない属性は無視され、スペルミスがあっても警告されません。属性名に大文字と小文字の違いがあると、別の名前として扱われます。属性値に大文字と小文字の違いがあったときに別のデータとして扱われるかどうかは、プログラムの仕様によります。

```
<Root>
  <Leaf attribute="1" other="no"/>
</Root>
```

タグによっては、同じ名前のタグを複数並べることができます。1つしか指定できないタグでは、2つ目以降のタグは無視されます。1つしか指定できないかどうかはプログラムが定義するタグの仕様によります。

```
<Root>
  <Leaf attribute="1" other="no"/>
  <Leaf attribute="2" other="no"/>
  <Leaf attribute="3" other="yes"/>
</Root>
```

(2) XPath

XPath は、XML テキストの中の一部を指し示すアドレスの一種です。ファイルのパスに近い文法で記述します。ここでは、基本的な書き方のみ示します。

XML テキストが下記のようにあるとき、

```
<Root>
  <LeafA>ABC</LeafA>
  <LeafB attribute="1" other="no"/>
</Root>
```

テキスト ABC の値の場所を指す XPath は、下記ようになります。XML ノードの間は、スラッシュで区切ります。タグの間のテキストを指すときは、text() と記述し、末尾の () が必要です。大文字と小文字が異なると、異なる名前を指定したことになります。下記はサンプルです。

```
/Root/LeafA/text()
```

属性値 1 の場所を指す XPath は、下記ようになります。タグ名と属性名の上にノードの区切りであるスラッシュと、属性名の先頭に @ が必要です。

```
/Root/LeafB/@attribute
```

先頭が / でないときは、相対パスになります。

```
LeafB/@attribute
@attribute
```

(3) # フラグメント

プログラムの仕様によっては、設定するファイル名やパスの後に # と ID 名（フラグメント）を指定することができます。以下はサンプルです。

```
Folder¥File.xml#Leaf1
```

フラグメントは、ファイルの中のすべての XML タグの id 属性の値と比較します。大文字と小文字が異なると、異なる名前を指定したことになります。フラグメント付きのパスは、一致する id 属性を持つ XML タグを指します。たとえば、Folder¥File.xml#Leaf1 は、Folder¥File.xml ファイルの中（下記）の LeafA タグを指します。

```
<Root>
  <LeafA id="Leaf1">ABC</LeafA>
  <LeafB id="Leaf2">ABC</LeafB>
</Root>
```


6.2 バイナリ変換 ConvertBin

バイナリ ファイルを、C 言語の配列や、S レコード形式（モトローラ S レコード形式）に変換します。

コマンド名	内容
BinToC	バイナリ ファイルを C 言語の配列に変換します。 変数名を指定することができます。
BinToSRec	バイナリ ファイルを S レコード形式に変換します。 コメントとロードアドレスと実行アドレスを指定することができます。 データのためのバイナリでは、実行アドレスは 0 を指定してください。
SRecToBin	S レコード形式をバイナリ ファイルに変換します。

RGA_Tools.vbs をダブルクリックした後のウィンドウ：

RGA Tools - Copyright(c) 2012-2015 Renesas Electronics Corporation

1. 画像フォーマット変換 [RunImagePackager]
2. エラー情報検索 [SearchErrorInformation]
3. バイナリ変換 [ConvertBin]

番号またはコマンド >3

(([ConvertBin]))

1. バイナリ→C 言語変換 [BinToC]
2. バイナリ→S レコード形式変換 [BinToSRec]
3. S レコード形式→バイナリ変換 [SRecToBin]

番号またはコマンド >

6.3 画像ファイル作成 RawToBmp

フレームバッファにある Raw データを BMP ファイルに変換します。

RGA_Tools.vbs をダブルクリックした後のウィンドウ：

RGA Tools - Copyright(c) 2012-2015 Renesas Electronics Corporation

1. 画像フォーマット変換 [RunImagePackager]
2. エラー情報検索 [SearchErrorInformation]
3. バイナリ変換 [ConvertBin]
4. BMP ファイルに変換 [RawToBmp]

番号またはコマンド >4

設定ファイルのパス >C:¥Folder¥RawToBmp.ini

設定ファイルのサンプル：

```
RawPath = Image.bin
OutBmpPath = Image.bmp
Stride = 1600
Format = RGB565
```

設定ファイルの内容：

属性名	内容
RawPath	フレームバッファにあった Raw データを保存したファイルのパス
OutBmpPath	出力先の BMP ファイルのパス
Stride	1 ライン下の同じ x 座標をもつピクセルへのバイト数。
Format	ピクセルフォーマット。ARGB8888, RGB565, ARGB1555, ARGB4444, YCbCr422, A8, A4, A1 のいずれか。参考：表1.3。
ReadOffset	8 バイト分をリードする順番のオフセットを並べた値。 例：01234567(Little endian), 76543210(Big endian) ただし、YCbCr422 では使えません。

7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RZ/A2Mグループ ユーザーズマニュアル ハードウェア編

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK7921053C00000BE（RZ/A2M CPUボード）ユーザーズマニュアル

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK79210XXB00000BE（RZ/A2M SUBボード）ユーザーズマニュアル

（最新版をルネサス エレクトロニクスホームページから入手してください。）

Arm Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C

（最新版を Arm ホームページから入手してください。）

Arm Cortex™-A9 Technical Reference Manual Revision: r4p1

（最新版を Arm ホームページから入手してください。）

Arm Generic Interrupt Controller Architecture Specification - Architecture version2.0

（最新版を Arm ホームページから入手してください。）

Arm CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3

（最新版を Arm ホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：統合開発

統合開発環境 e2 studio のユーザーズマニュアルは、ルネサス エレクトロニクスホームページから入手してください。

（最新版をルネサス エレクトロニクスホームページから入手してください。）

7.1 リファレンス シンボル

このドキュメントを参照するソース ファイルや HTML 文書には、章・図・表を識別するための番号の代わりにシンボル（リファレンス シンボル）が書かれています。リファレンス シンボルと章・図・表の対応関係を以下に示します。

リファレンス シンボル	章・図・表
Section_RGA_Initialize	4.4.2 初期化関数の内部の動き
Section_RGA_Finalize	4.4.6 終了処理について（*_Finalize関数）
Section_RGA_ImageFormat	6.1.5 出力バイナリ内のファイルフォーマット
Section_byte_per_pixel_t	4.2.3 byte_per_pixel_t
Figure_RGA_PixelFormat	図 1-4 ピクセル フォーマットの構造
Section_DefaultableFlags	4.4.4 デフォルト可能フラグ
Section_RGA_ImagePackager	6.1 画像フォーマット変換 ImagePackager
Section_RGA_IdentifyingImageFormat	4.4.3 画像の形式の識別
Figure_RGA_DrawImageChild	5.1.37 R_GRAPHICS_DrawImageChild

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.11	2020.9.30	p13, p15	図 1-4図4-1, 図4-3 レイアウト修正
		p1	制限事項追加 本ライブラリは、アフィン変化にのみ対応しています。
1.10	2019.5.17	p12	表 7.1 動作確認条件 コンパイラオプション"-mthumb-interwork"を削除
1.03	2019.4.15	p9	対応している JPEG の形式の対象の文章を RZ/A1 から RZ/A2M に修正
		p11	フォルダーツリーを変更
		p11	ユーザーのプロジェクトに RGA ライブラリをインストールする手順を変更
		p12	動作確認を行った統合開発環境のバージョンを更新
		p44	以前からサポートしていなかったピクセルフォーマットを削除
		p74	文章修正 : RZ/A1H ⇒ RZ/A2M
1.02	2018.12.28	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力ブルアップ電源を入れないでください。入力信号や入出力ブルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、 V_{IL} (Max.) から V_{IH} (Min.) までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、 V_{IL} (Max.) から V_{IH} (Min.) までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違っていると、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含まれます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

標準水準：コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。

6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。