SH7268/SH7269 グループ
グラフィックスライブラリ RGA

R01AN2840JJ0212
Rev. 2.12
2018.01.26

要旨

本アプリケーションノートでは、SH7268/SH7269 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。また HTML Canvas 2D Context の下位互換として動作できる C++API を提供
- アプリケーションが用意したメモリー領域を描画先、または、入力画像として使用可能
- 半透明な画像の描画と、アルファ マスクを使った半透明の描画
- 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）

対象デバイス

SH7268/SH7269 グループ

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、OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) がサポートしているベクター グラフィックスには対応していません。

ハードウェア アクセラレーションは、一部の描画に対してのみ使っています。(表 1.2),(5.11.1)

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。

目次

1. 仕様 4

2. ファイル構成 8

3. 動作確認条件 10

4. ハードウェア説明 11
4.1 ハードウェア構成例 11

5. ソフトウェア説明 12
5.1 動作概要 12
5.1.1 アプリケーション定義のバッファーに描画する場合 12
5.1.2 表示画面に描画する場合 14
5.2 必要メモリサイズ 16
5.3 定数一覧 17
5.4 型／クラス 18
5.4.1 基本型、値 18
5.4.2 エラーコード 19
5.4.3 C 言語専用の型 20
5.4.4 C++言語専用のクラス 21
5.4.5 C 言語/C++言語の両方で使用可能な型／クラス 24
5.4.6 文字列の書式 34
5.4.7 移植層の型 35
5.4.8 クラスの状態遷移 36
5.5 変数一覧 38
5.6 プロパティ 39
5.6.1 Canvas2D_ContextClass のプロパティ 39
5.6.2 Canvas2D_ImageClass のプロパティ 41
5.7 関数／メソッド 42
5.7.1 graphics_t クラスのメンバー関数に相当する関数 42
5.7.2 graphics_image_t クラスのメンバー関数に相当する関数 60
5.7.3 graphics_pattern_t クラスのメンバー関数に相当する関数 63
5.7.4 C++オブジェクトの生成に関する関数 64
5.7.5 Canvas2D_ContextClass のメンバー関数 65
5.7.6 Canvas2D_ImageClass に関連する関数 72
5.7.7 Canvas2D_PatternClass に関連する関数 73
5.7.8 WindowSurfacesClass のメンバー関数 74
5.7.9 window_surfaces_t クラスのメンバー関数に相当する関数 76
5.7.10 byte_per_pixel_t クラスに関連する関数 80
5.7.11 v_sync_t クラスに関連する関数 81
5.7.12 vram_ex_stack_t クラスに関連する関数 83
5.7.13 animation_timing_function_t クラスに関連する関数 84
5.7.14 その他の関数 86
5.7.15 文字列内の関数 88
5.8 移植層の関数 89
5.8.1 RGA のデフォルト設定に関する関数 90
5.8.2 キャッシュに関する関数 92
5.8.3 表示コントローラー割込みに関する関数 93
5.8.4 OSPL による移植層の関数 94
5.8.5 RGPNCG による移植層の関数 95
5.9 割込みハンドラーの登録 100
5.9.1 OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) の割込み 101
5.9.2 ビデオディスプレイコントローラ 4 (VDC4) の割込み 102
5.9.3 JPEG コーデックユニット (JCU) の割込み 103
5.10 セクション 104
5.11 補足 105

5.11.1 Canvas 2D との対応表、H/W アクセラレーション対応表 105
5.11.2 初期化関数の内部の動き 108
5.11.3 画像の形式の識別 109
5.11.4 フラッシュモード 110
5.11.5 サンプル画面制御のレイヤー構造 111
5.11.6 フラグド構造体パラメーター 112
5.11.7 デフォルト可能フラグ 113
5.11.8 内部変数を定数で初期化する関数について（*_initConst 関数） 115
5.11.9 終了処理について（*_Finalize 関数） 116
5.11.10 C++言語と JavaScript のオブジェクトの互換性について 118

6. ツール説明 119
6.1 画像フォーマット変換 ImagePackager 119
6.1.1 操作手順 119
6.1.2 ファイル一覧 119
6.1.3 サンプル 120
6.1.4 出力バイナリの種類（言語） 120
6.1.5 出力バイナリ内のファイルフォーマット 120
6.1.6 入力フォーマット 121
6.1.7 BinaryImageConfig.image.xml に記述できるパラメーター 121
6.1.8 XML の基本的な書き方 128
6.2 エラー情報検索 SearchErrorInformation 131
6.3 バイナリ変換 ConvertBin 132
6.4 画像ファイル作成 RawToBmp 133

7. ソフトウェア改訂内容 134
7.1 電源投入直後から描画できないことがある現象を改善 134
7.2 拡大して画像を描画する際の制限事項 134
7.3 透明なピクセルを持つ画像を描画する際の制限事項 134
7.4 完全不透過のピクセルを持つ画像を描画する際の制限事項 135

8. サンプルコード 136

9. 参考ドキュメント 137

ホームページとサポート窓口 137

改訂記録 138

製品ご使用上の注意事項 139

ご注意書き 140

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 1.1 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
OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)	グラフィックス描画
JPEG コーデックユニット (JCU)	JPEG 展開
ビデオディスプレイコントローラ 4 (VDC4)	画面表示

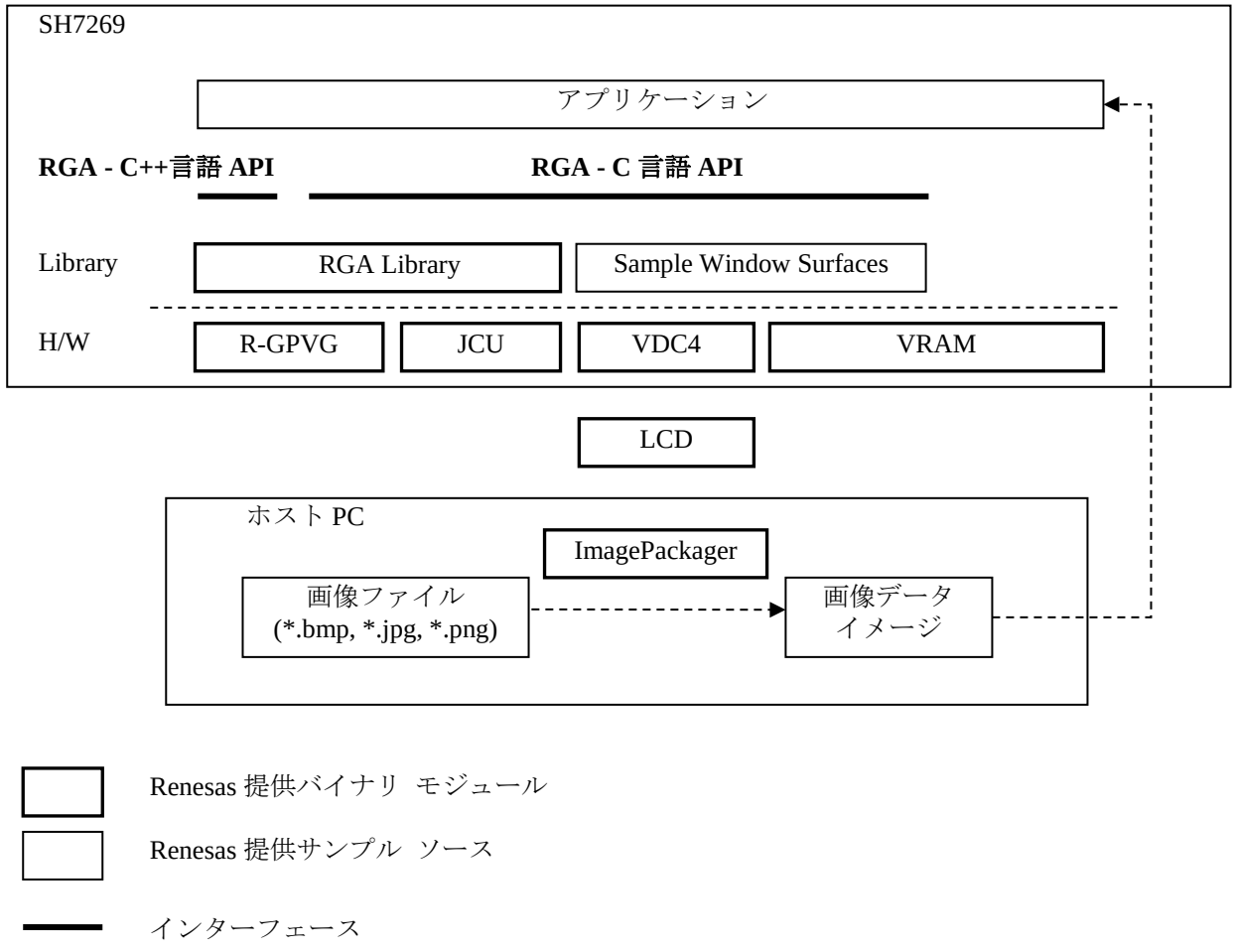


図 1.1 ブロック図

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ーの大きさの最大は、幅が 1280 ピクセル、高さが 1024 ピクセルです。ソース画像の大きさの最大も、1280×1024 ピクセルです。

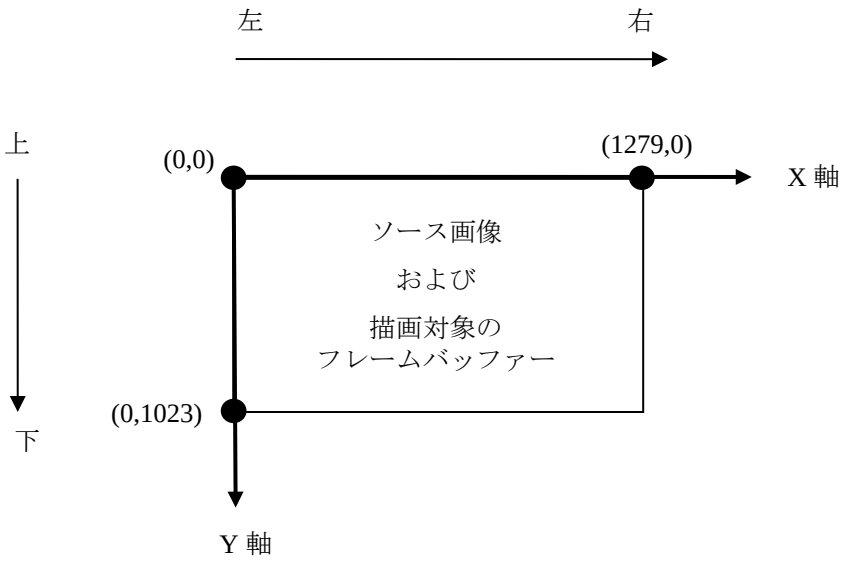


図 1.2　描画対象とソース画像の最大サイズ

表 1.2　対応している描画先のピクセルフォーマット

	XRGB 8888	ARGB 8888	RGB 565	ARGB 1555	ARGB 4444	YUV 422[1]	CLUT8	CLUT4	CLUT1
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(5.7.1.26 など)	○	○	○	○	○	○	△[2]	△[2]	△[2]
DrawImageChild(5.7.1.28)	○	○	○	○	○	○	×	×	×
矩形塗りつぶし	○	○	○	○	○	○	×	×	×

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先のピクセルフォーマットの組み合わせ

描画先 / ソース画像	XRGB 8888	ARGB 8888	RGB 565	ARGB 1555	ARGB 4444	YUV 422[1]	CLUT8	CLUT4	CLUT1
JPEG	○	○	○	○	○	×	×	×	×
XRGB8888	○	○	○	○	○	×	×	×	×
ARGB8888	○	○	○	○	○	×	×	×	×
RGB565	○	○	○	○	○	×	×	×	×
ARGB1555	○	○	○	○	○	×	×	×	×
ARGB4444	○	○	○	○	○	×	×	×	×

[1] YUV422 は、FourCC=UYVY。CbCr の中心（グレースケール）は 0x80。

[2] 制限事項があります。詳しくは関数の説明を参照してください。

ソース画像＼描画先	XRGB 8888	ARGB 8888	RGB 565	ARGB 1555	ARGB 4444	YUV 422[1]	CLUT8	CLUT4	CLUT1
R8G8B8A8[3]	○	○	○	○	○	×	×	×	×
YUV422[3]	×	×	×	×	×	○	×	×	×
CLUT8	×	×	×	×	×	×	○	×	×
CLUT4	×	×	×	×	×	×	×	○	×
CLUT1	×	×	×	×	×	×	×	×	○

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。SH7269 では、ハードウェアである JPEG コーデックユニット (JCU) と OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) を使います。 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。

A8,A4,A1 フォーマットは使えません。

表 1.4 対応している JPEG の形式

伸張モジュール	SH7269 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)

以下に、ピクセルフォーマットの詳細を示します。 SH7269 版の RGA は、ビッグエンディアンです。 たとえば、XRGB8888 の Red は、ピクセルの先頭アドレス＋1 のバイトの位置になります。

XRGB8888

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

ARGB8888

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

RGB565

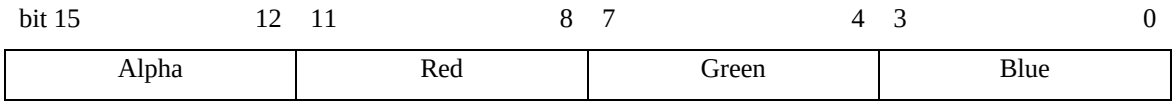
bit 15 – 11	bit 10 – 5	bit 4 – 0
Red	Green	Blue

ARGB1555

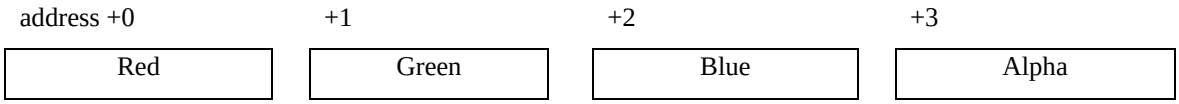
bit 15	bit 14 – 10	bit 9 – 5	bit 4 – 0
Alpha	Red	Green	Blue

[3] ソース画像が R8G8B8A8、YUV422 のときは、S/W レンダリングになります。

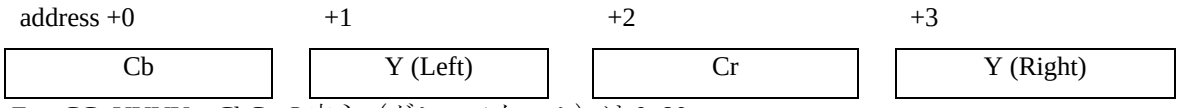
ARGB4444



R8G8B8A8

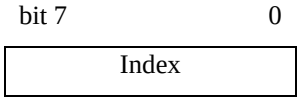


YCbCr422

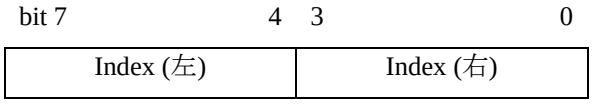


FourCC=UYVY。CbCr の中心（グレースケール）は 0x80。

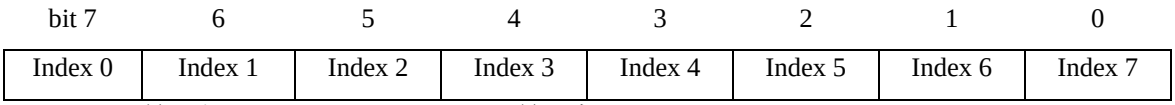
CLUT8



CLUT4

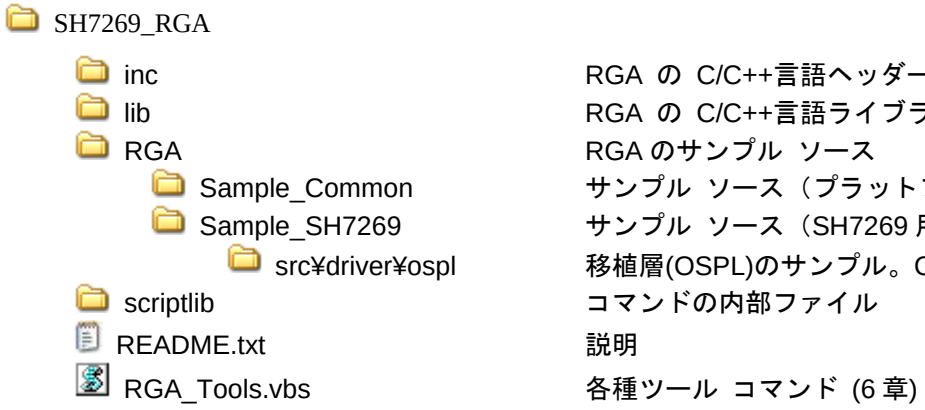


CLUT1



Index 0 は最も左のピクセル。 Index 7 は最も右のピクセル。

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ストールするときは、inc フォルダーと lib¥SH7269 フォルダーをすべてコピーしてください。また、使用しているドライバーや共通関数など、インクルードのエラーになったファイルや、リンクエラーになった関数が定義されているソースファイルもコピーしてください。

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。

ワークバッファーは、非キャッシュ領域に配置してください。また、メモリー領域からはみ出ないようにしてください。ワークバッファーのアドレス（R_GRAPHICS_STATIC_OnInitializeDefault 関数の work_buffer_address メンバー変数）は、RGA の初期化関数の内部で、r_ospl_memory.c ファイルの中にある R_OSPL_ToUncachedAddress 関数を呼び出すことで非キャッシュ領域に変換されるか、すでに非キャッシュ領域にあるかどうかをチェックし、R_OSPL_ToPhysicalAddress 関数でハードウェアがアクセスできる物理アドレスに変換されます。

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。

Map Section Information にセクションの設定を追加してください（参照：5.10 セクション）。

STBCR8 レジスタの bit4 を 0 に設定して、クロックを供給してください。

主なヘッダー ファイル一覧（inc フォルダー）：

ファイル名	内容
clib_drivers.h	共有コード
frame_buffer.h	フレームバッファー
ncg_*.h	移植層 RGPNCG
r_typedefs.h	基本型
RGA.h	RGA メイン
RGA_API.h	RGA：API 関係
RGA_Config.h	RGA：設定関係
RGA_Cpp.h	RGA：C++ API 関係
RGA_raw_image.h	Raw 形式画像
vsync.h	V-sync 制御
vsync_pl.h	V-sync 制御 移植層
window_surfaces.h	サンプル画面制御（window_surfaces_t）
window_surfaces.hpp	サンプル画面制御（window_surfaces_t）C++ API

主なライブラリ ファイル一覧（lib フォルダー）：

ファイル名	内容
RGA.lib	RGA の API 部
RGAH.lib	RGA 向け OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)ドライバー

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
使用マイコン	SH7269
動作周波数	266MHz
動作電圧	内部：1.25V、I/O：3.3V
統合開発環境	High-performance Embedded Workshop 4.09.01.007
C コンパイラー	C/C++ compiler package for SuperH RISC engine family V.9.04 Release 00
動作モード	
使用ボード	「(4.1)ハードウェア構成例」を参照
ジャンパー等の設定	CPU ボード（ボード型名：R0K572690C000BR） : JP4=Open, JP5～JP10=1-2, JP11=*. JP12～JP16=1-2, SW5=1110, SW6=010100 オーディオボード（ボード型名：R0K572690B000BR） : JP2=1-2, JP3=Open, JP4=1-2, JP5=2-3, JP6=Open, JP7=Open, JP8=1-2, JP9=2-3, JP10=Open, JP11=Open, JP12=* * は、電源の種類による。

4. ハードウェア説明

4.1 ハードウェア構成例

図 4.1 に接続例を示します。

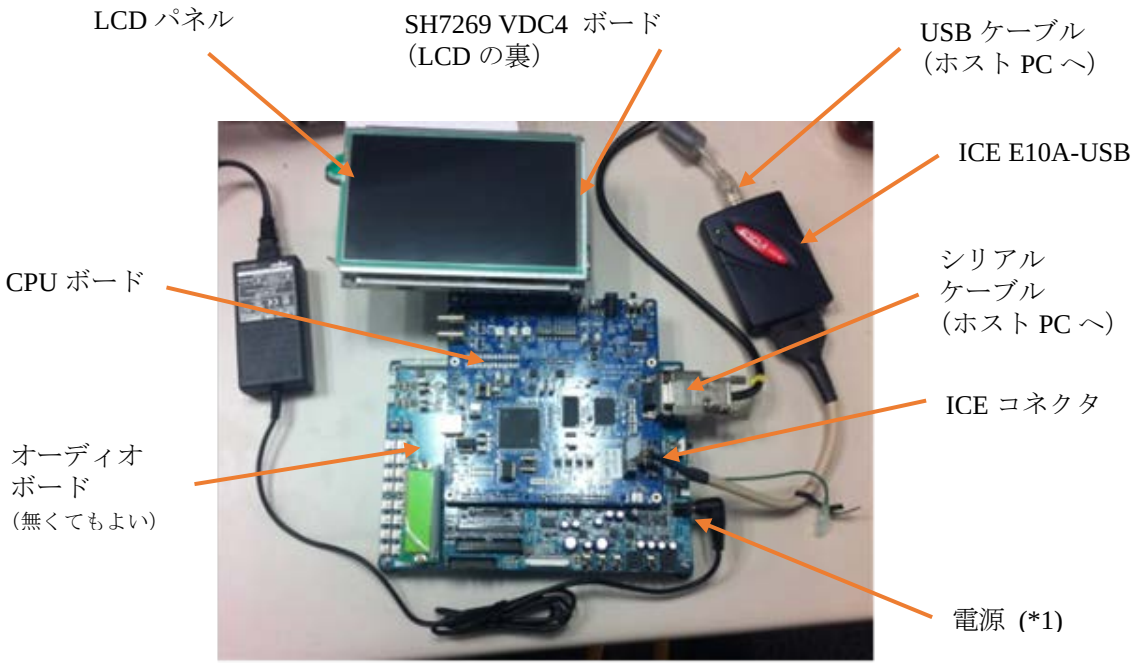


図 4.1 接続例

(*1) 写真の電源の位置は、オーディオボードを接続するときの位置です。オーディオボードを接続しないときは、ボードのユーザーズマニュアルを参考に、ジャンパーの設定を変更して、CPU ボードに電源を接続してください。

5. ソフトウェア説明

5.1 動作概要

5.1.1 アプリケーション定義のバッファーに描画する場合

5.1.1.1 フローチャート

図 5.1 にアプリケーションが定義したフレームバッファー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。

C++言語 API を使うときは、以下に示す C 言語 API を使った説明と、「(5.11.1)Canvas 2D との対応表」に書かれた C 言語 API と C++言語 API の対応関係を参考にしてください。

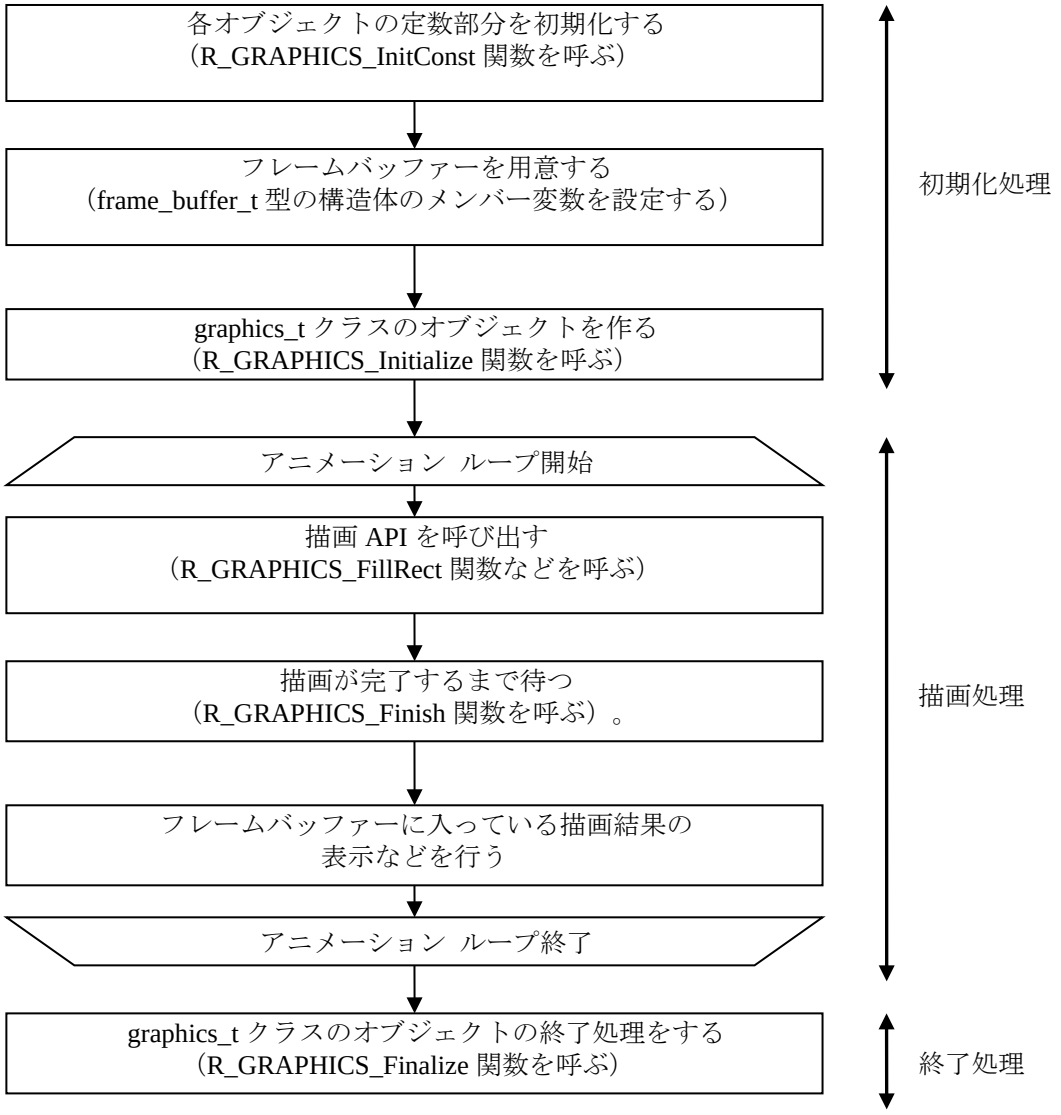


図 5.1 アプリケーション定義のバッファーに描画する場合 - フローチャート

5.1.1.2 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ー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下図に示します。

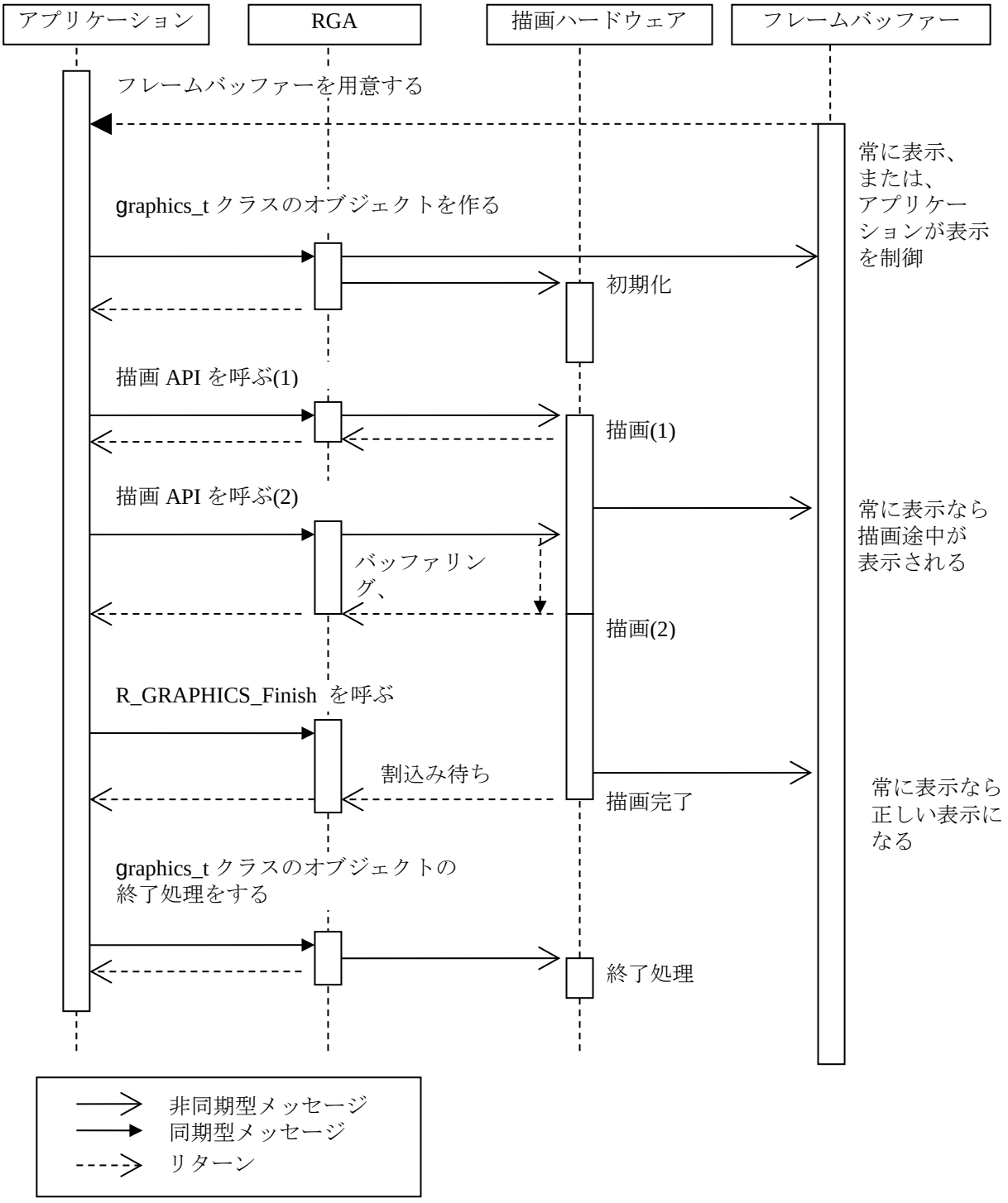

アプリケーション
RGA
描画ハードウェア
フレームバッファー
フレームバッファーを用意する
常に表示、
または、
アプリケー
ションが表示
を制御
graphics_t クラスのオブジェクトを作る
初期化
描画 API を呼ぶ(1)
描画(1)
描画 API を呼ぶ(2)
常に表示なら
描画途中が
表示される
バッファリン
グ、
描画(2)
R_GRAPHICS_Finish を呼ぶ
割込み待ち
描画完了
常に表示なら
正しい表示に
なる
graphics_t クラスのオブジェクトの
終了処理をする
終了処理
非同期型メッセージ
同期型メッセージ
リターン

図 5.2　動作タイミング

5.1.2 表示画面に描画する場合

5.1.2.1 フローチャート

図 5.1 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。

C++言語 API を使うときは、以下に示す C 言語 API を使った説明と、「(5.11.1)Canvas 2D との対応表」に書かれた C 言語 API と C++言語 API の対応関係を参考にしてください。

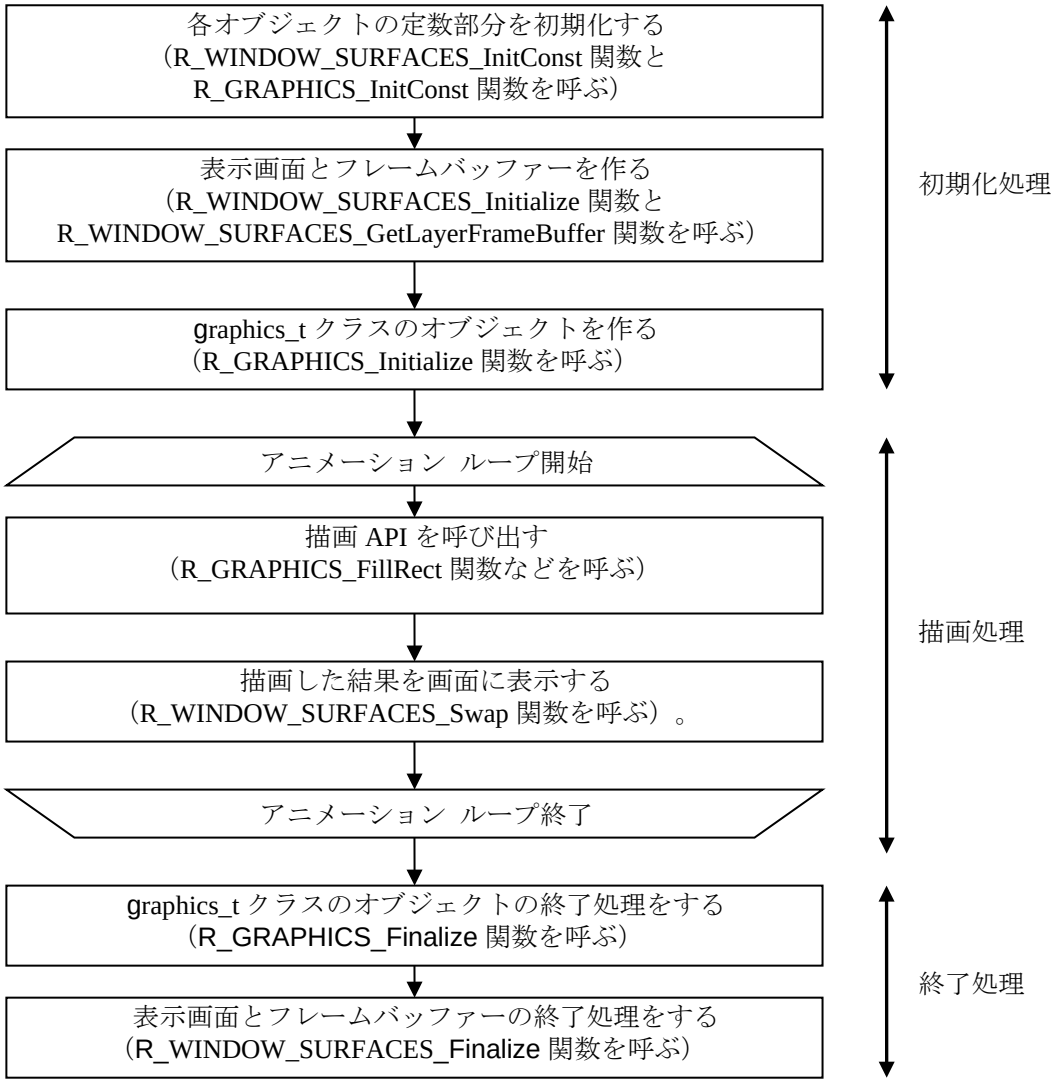


図 5.3 表示画面に描画する場合 - フローチャート処理

5.1.2.2 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ー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 5.4 に示します。

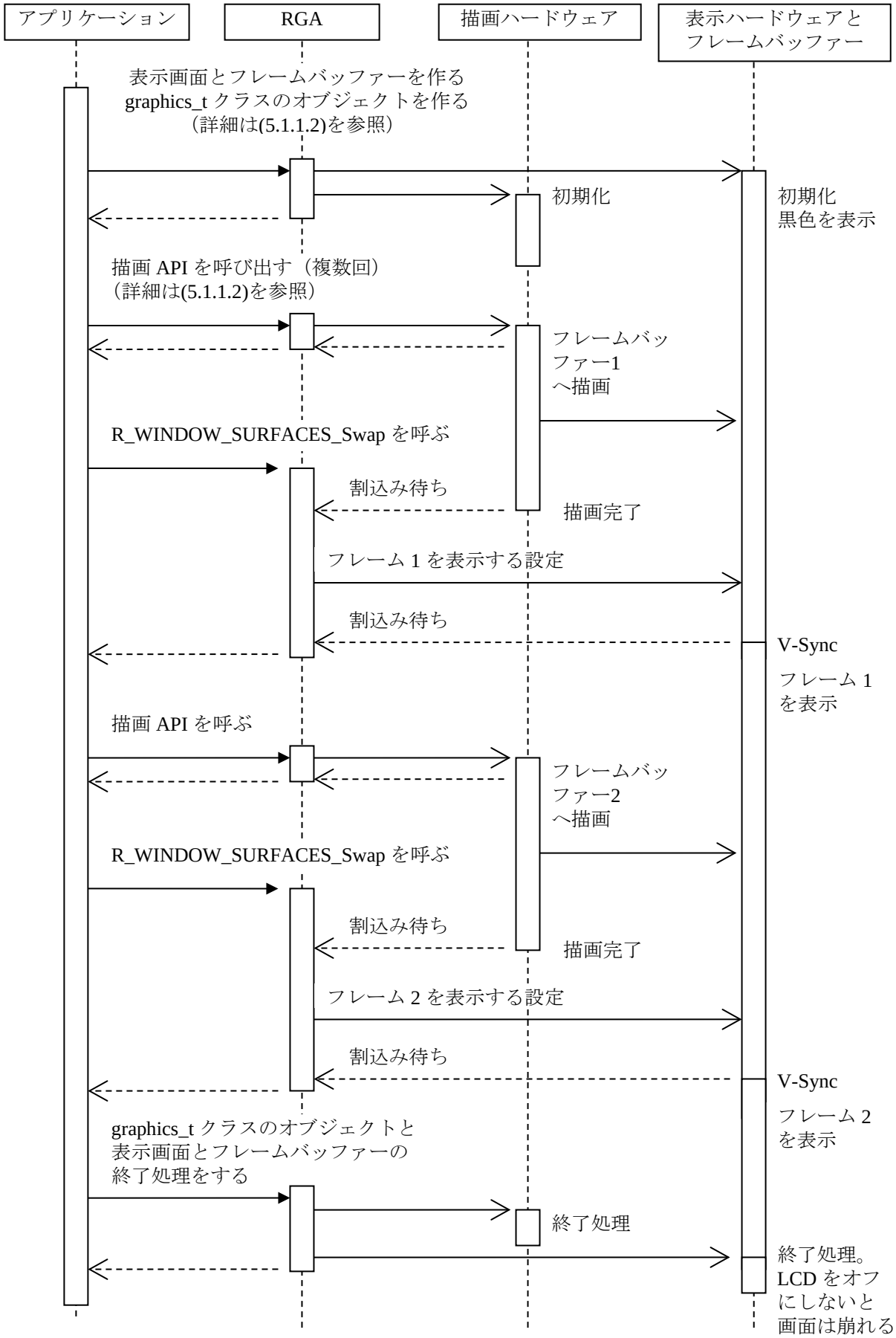

アプリケーション
RGA
描画ハードウェア
表示ハードウェアと
フレームバッファー
表示画面とフレームバッファーを作る
graphics_t クラスのオブジェクトを作る
（詳細は(5.1.1.2)を参照）
初期化
初期化
黒色を表示
描画 API を呼び出す（複数回）
（詳細は(5.1.1.2)を参照）
フレームバッ
ファー1
へ描画
R_WINDOW_SURFACES_Swap を呼ぶ
割込み待ち
描画完了
フレーム 1 を表示する設定
割込み待ち
V-Sync
フレーム 1
を表示
描画 API を呼ぶ
フレームバッ
ファー2
へ描画
R_WINDOW_SURFACES_Swap を呼ぶ
割込み待ち
描画完了
フレーム 2 を表示する設定
割込み待ち
V-Sync
フレーム 2
を表示
graphics_t クラスのオブジェクトと
表示画面とフレームバッファーの
終了処理をする
終了処理
終了処理。
LCD をオフ
にしないと
画面は崩れる

図 5.4　動作タイミング

5.2 必要メモリサイズ

表 5.1 に必要メモリサイズを示します。

表 5.1 必要メモリサイズ

使用メモリー	サイズ	備考
ROM	43114	セクション *_RGA, *_RGAH, *_JCU、 * は、P, C, D
RAM	1644	セクション *_RGA, *_RGAH, *_JCU、 * は、B, R
	R_RGA_CalcWorkBufferSize を参照 (5.7.14.3)	ワークバッファー
	R_RGA_CalcWorkBufferB_Size を参照 (5.7.14.4)	ワークバッファーB
	800x480x16bit x2 なら、1536000	フレームバッファー
	800x480x16bit x3 なら、2304000 不要なら 0	アプリケーションが使うバックバッファー
	1700	サンプルが使うスタック使用量

【注】 必要メモリサイズは C コンパイラーのバージョンやコンパイルオプションにより異なります。

5.3 定数一覧

表 5.2 にサンプルコードで使用する定数を示します。

表 5.2 サンプルコードで使用する定数

定数名	設定値	内容
RGA_VERSION	210	RGA のバージョン
RGA_VERSION_STRING	"2.10"	RGA のバージョン文字列
RGA_FRAME_BUFFER_ADDRESS_ALIGNMENT	32	描画先のフレームバッファーのアドレスのアラインメント（バイト）
RGA_SOURCE_IMAGE_STRIDE_ALIGNMENT	32	ソース画像の中の、次の行までのバイト数（Stride）のアラインメント（バイト）
RGA_DESTINATION_STRIDE_ALIGNMENT	32	描画先のフレームバッファーの中の、次の行までのバイト数（Stride）のアラインメント（バイト）
RGA_JPEG_ADDRESS_ALIGNMENT	8	JPEG データのアドレスのアラインメント（バイト）
RGA_JPEG_MAX_WIDTH_ALIGNMENT	16	ソース JPEG 画像の MCU の幅の最大（ピクセル）
RGA_JPEG_MAX_HEIGHT_ALIGNMENT	16	ソース JPEG 画像の MCU の高さの最大（ピクセル）
RGA_VDC4_BUFFER_ADDRESS_ALIGNMENT	64	ビデオディスプレイコントローラ 4 (VDC4) が表示するフレームバッファーの先頭アドレスのアラインメント（バイト）
RGA_WORK_BUFFER_STRIDE	64	ワークバッファーの 1 行あたりのサイズ（バイト）
RGA_WORK_BUFFER_ADDRESS_ALIGNMENT	64	ワークバッファーの先頭アドレスのアラインメント（バイト）
RGA_WORK_BUFFER_HEIGHT_ALIGNMENT	8	ワークバッファーの高さのアラインメント（ピクセル）
RGA_WORK_BUFFER_B_ADDRESS_ALIGNMENT	32	ワークバッファーB の先頭アドレスのアラインメント（バイト）

5.4 型／クラス

5.4.1 基本型、値

シンボル	内容
int_t	高速な整数、符号あり、本ライブラリでは 32 ビット整数。
uint32_t	32 ビット整数、符号なし
int32_t	32 ビット整数、符号あり
uint16_t	16 ビット整数、符号なし
int16_t	16 ビット整数、符号あり
uint8_t	8 ビット整数、符号なし
int8_t	8 ビット整数、符号あり
uint64_t	64 ビット整数、符号なし
int64_t	64 ビット整数、符号あり
bit_field_t	ビット フィールド、本ライブラリでは符号なし 32 ビット整数
bit_field32_t	32 ビットのビット フィールド、符号なし整数
bool_t	論理型。値は true(=1), false(=0)
true	論理型整数値 1
false	論理型整数値 0
float32_t	32 ビット浮動小数
float64_t	64 ビット浮動小数
float128_t	128 ビット浮動小数
char_t	8 ビット文字
physical_address32_t	32 ビット物理アドレス
errnum_t	エラーコード、0=エラーなし。参考：(5.4.2)

5.4.2 エラーコード

シンボル	値	内容
0	0	エラーなし
E_OTHERS	0x0001=1	パラメーター エラー。参考：(6.2)エラー情報検索 SearchErrorInformation
E_LIMITATION	0x040F=1039	制限事項により使えない機能を使おうとした
E_FEW_ARRAY	0x0411=1041	配列の要素数が足りない
E_NOT_SUPPORTED_PIXEL_FORMAT	0x9400=37888	サポートしていないピクセルフォーマット
E_ACCESS_DENIED	0x0417=1047	処理が拒否された

5.4.3 C 言語専用の型

RGA が提供する、C 言語のみで使われる型です。

本章に示す型定義は C++言語でも使えますが、C++言語で使用可能なクラスが別途存在します。詳細は(5.4.4)を参照してください。

C 言語で使える型については、「(5.4.5)C 言語/C++言語の両方で使用可能な型／クラス」も参照してください。

5.4.3.1 一覧

表 5.3 RGA が提供する C 言語専用の型

章	型名	内容
5.4.3.2	graphics_t	グラフィックス描画コンテキスト
5.4.3.3	graphics_image_t	入力画像
5.4.3.4	graphics_pattern_t	画像を並べたパターン

5.4.3.2 graphics_t

```
#include <RGA.h>
typedef struct _graphics_t graphics_t;
```

グラフィックス描画コンテキストの型です。

SH7269 では、内部で OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)、またはソフトウェアレンダリングを使います。

アプリケーションは、メンバー変数にアクセスしないでください。

メンバー関数に相当する関数：(5.7.1)

5.4.3.3 graphics_image_t

```
#include <RGA.h>
typedef struct _graphics_image_t graphics_image_t;
```

画像の型です。

ImagePackager ツールは、画像ファイルから変換して、この型のデータを出力します。

ImagePackager ツールを使用せず、動的に graphics_image_t クラスのオブジェクトを生成することもできます。

アプリケーションは、メンバー変数にアクセスしないでください。

メンバー関数に相当する関数：(5.7.1.39)

5.4.3.4 graphics_pattern_t

```
#include <RGA.h>
typedef struct _graphics_pattern_t graphics_pattern_t;
```

画像を並べたパターンの型です。

アプリケーションは、メンバー変数にアクセスしないでください。

メンバー関数に相当する関数：(5.7.3)

5.4.4 C++言語専用のクラス

RGA が提供する C++言語のクラスです。

RGA が提供するクラスは、JavaScript と互換があるように作られているため、コーディングに注意してください。詳細は「(5.11.10)C++言語と JavaScript のオブジェクトの互換性について」を参照してください。

C++言語で使えるクラス／型については、「(5.4.5)C 言語/C++言語の両方で使用可能な型／クラス」も参照してください。

5.4.4.1 一覧

表 5.4 RGA が提供する C++言語専用の型

章	型名	内容
5.4.4.2	Canvas2D_ContextClass	グラフィックス描画コンテキスト
5.4.4.3	Canvas2D_ContextConfigClass	Canvas2D_ContextClass の設定をする型
5.4.4.4	Canvas2D_ImageClass	入力画像
5.4.4.5	Canvas2D_PatternClass	画像を並べたパターン
5.4.4.6	WindowSurfacesClass	フレームバッファーと画面表示
5.4.4.7	WindowSurfacesConfigClass	WindowSurfacesClass::initialize メンバー関数の引数
5.4.4.8	LayerAttributesClass	WindowSurfacesClass::access_layer_attributes メンバー関数の引数

5.4.4.2 Canvas2D_ContextClass

```
#include <RGA.h>
class Canvas2D_ContextClass;
```

グラフィックス描画コンテキストのクラスです。

C++専用です。JavaScript のコーディングルールに合わせています。

SH7269 では、内部で、OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)、または、ソフトウェア レンダリングを使います。

プロパティの説明は、プロパティ仕様(5.6.1)を参照してください。

メンバー関数の説明は、関数仕様(5.7.3)、(5.7.4.2)を参照してください。

c_LanguageContext プロパティを使えば、graphics_t 型のメンバー関数が扱えます。(5.7.1)を参照してください。

サンプル：

```
Canvas2D_ContextClass canvas2d = R_RGA_New_Canvas2D_ContextClass( frame );

/* JavaScript 互換部分の開始 */
canvas2d.clearRect( 0, 0, frame_width, frame_height );
canvas2d.Style = "#0f0";
canvas2d.fillRect( 100, 100, 200, 100 );
/* JavaScript 互換部分の終了 */

if ( canvas2d.errNum != 0 ) { /* error */ }

R_WINDOW_SURFACES_SwapBuffers( ... );
canvas2d.destroy();
```

5.4.4.3 Canvas2D_ContextConfigClass

概　要	C++言語で Canvas2D_ContextClass の設定をする型
ヘッダー	RGA.h
説　明	コンストラクターによって、デフォルト値が入ります。

メンバー変数

frame_buffer_t* frame_buffer	描画対象のフレームバッファー。設定が必須です。 frame_buffer_t 型構造体のアドレスを設定してください。
bool_t is_fast_manual_flush	高速手動フラッシュモード(5.11.4)かどうか。デフォルトは、false

5.4.4.4 Canvas2D_ImageClass

```
#include <RGA.h>
class Canvas2D_ ImageClass;
```

内容を参照できる画像のオブジェクトの型です。

C++専用です。JavaScript のコーディングルールに合わせています。

プロパティの説明は、プロパティ仕様(5.6.2)を参照してください。

メンバー関数の説明は、関数仕様(5.7.6)を参照してください。

5.4.4.5 Canvas2D_PatternClass

```
#include <RGA.h>
class Canvas2D_PatternClass;
```

画像を並べたパターンのクラスです。

C++専用です。JavaScript のコーディングルールに合わせています。

メンバー関数の説明は、関数仕様(5.7.7)を参照してください。

5.4.4.6 WindowSurfacesClass

```
#include "RGA.h"
class WindowSurfacesClass;
```

フレームバッファーと画面表示のクラスです。

C++専用です。mbed C++ のコーディングルールに合わせています。

メンバー関数の説明は、関数仕様(5.7.8)を参照してください。

5.4.4.7 WindowSurfacesConfigClass

```
#include "RGA.h"
class WindowSurfacesConfigClass;
```

WindowSurfacesClass::initialize メンバー関数の引数のクラスです。

C++専用です。mbed C++ のコーディングルールに合わせています。

メンバー変数は、window_surfaces_config_t 型 (5.4.5.6) から継承されます。ただし、コンストラクターによってデフォルト値が設定されます。 flags メンバー変数を設定する必要はありません。

5.4.4.8 LayerAttributesClass

```
#include "RGA.h"
class LayerAttributesClass;
```

WindowSurfacesClass::access_layer_attributes メンバー関数の引数のクラスです。

C++専用です。mbed C++ のコーディングルールに合わせています。

メンバー変数は、layer_attributes_t 型 (5.4.5.7) から継承されます。ただし、コンストラクターによってデフォルト値が設定されます。 flags メンバー変数を設定する必要はありません。

5.4.5 C 言語/C++言語の両方で使用可能な型／クラス

C 言語から使用可能な型は、「(5.4.3)C 言語専用の型」も参照してください。

C++言語から使用可能な型は、「(5.4.4)C++言語専用のクラス」も参照してください。

5.4.5.1 一覧

表 5.5 RGA が提供する C 言語/C++言語の両方で使用可能な型／クラス

章	型名	内容
5.4.5.2	frame_buffer_t	描画先フレームバッファー
5.4.5.3	graphics_config_t	graphics_t の設定をする型
5.4.5.4	graphics_quality_flags_t	描画品質を表すデフォルト可能フラグ
5.4.5.5	window_surfaces_t	フレームバッファーと画面表示
5.4.5.6	window_surfaces_config_t	window_surfaces_t の設定
5.4.5.7	layer_attributes_t	R_WINDOW_SURFACES_AccessLayerAttributes 関数のパラメーター
5.4.5.8	access_t	設定値を設定する（Write）か、取得する（Read）か、などの操作
5.4.5.9	byte_per_pixel_t	１ピクセルあたりのバイト数
5.4.5.10	pixel_format_t	ピクセルフォーマット
5.4.5.11	frame_buffer_delegate_t	アプリケーションが定義するデータ
5.4.5.12	v_sync_t	表示画面の V-Sync 信号の同期
5.4.5.13	vram_ex_stack_t	外部 RAM に配置したオフスクリーン バッファーを取得できるスタックの型のサンプル
5.4.5.14	graphics_image_properties_t	画像のプロパティの型
5.4.5.15	graphics_composite_operation_t	合成操作の種類
5.4.5.16	graphics_status_t	グラフィックス描画 コンテキストを保存する領域
5.4.5.17	graphics_matrix_float_t	行列の要素
5.4.5.18	repetition_t	パターンの繰り返し方の型
5.4.5.19	r8g8b8a8_t	R,G,B,A の順になっているピクセルフォーマット
5.4.5.20	animation_timing_function_t	アニメーションの動き（ベジェ関数）
5.4.5.21	graphics_jpeg_decoder_t	JPEG デコーダーの種類

5.4.5.2 frame_buffer_t

概　要	描画対象フレームバッファー
ヘッダー	RGA.h, frame_buffer.h
説　明	

メンバー変数

uint8_t* buffer_address[ ]	フレームバッファーの先頭アドレスの配列。キャッシュ領域の論理アドレス（一般変数と同じ）。[条件] 32 の倍数を指定してください。
int_fast32_t buffer_count	buffer_address の配列要素数。1 ならシングルバッファー。2 ならダブルバッファー。[設定範囲] 1～2。
int_fast32_t show_buffer_index	表示しているバッファー番号。ソース画像にするバッファー番号。buffer_address の配列番号。
int_fast32_t draw_buffer_index	描画しているバッファー番号。buffer_address の配列番号。
int_fast32_t　width	フレームバッファーの幅（ピクセル）。[設定範囲]　図 1.2 を参照。
byte_per_pixel_t byte_per_pixel	１ピクセルあたりのバイト数。参考：表 1.2。
int_fast32_t　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。[条件] 32 の倍数。
int_fast32_t　height	フレームバッファーの高さ（ピクセル）。[設定範囲] 図 1.2 を参照。
pixel_format_t pixel_format	ピクセルフォーマット。参考：表 1.2。
frame_buffer_delegate_t* delegate	ユーザー定義変数。RGA はアクセスしません。

サンプル：

```
static uint8_t  gs_frame_buffer_memory[2][800][480][4];
frame_buffer_t  frame;
frame.buffer_address[0] = gs_frame_buffer_memory [0];
frame.buffer_address[1] = gs_frame_buffer_memory [1];
frame.buffer_count = 2;
frame.show_buffer_index = 0;
frame.draw_buffer_index = 1;
frame.width = 800;
frame.byte_per_pixel = 4;
frame.stride = 800 * 4;
frame.height = 480
frame.pixel_format = PIXEL_FORMAT_ARGB8888;
frame.delegate = NULL;
```

5.4.5.3 graphics_config_t

概　要	graphics_t の設定
ヘッダー	RGA.h
説　明	デフォルト値をカスタマイズするときや、必須の設定をまとめたいときは、「(5.8.1.2）R_GRAPHICS_OnInitialize_FuncType」を使ってください。

メンバー変数

bit_flags_fast32_t　flags	フラグド構造体パラメーター（必須）参考：(5.11.6)。 F_GRAPHICS_FRAME_BUFFER F_GRAPHICS_WORK_BUFFER_ADDRESS F_GRAPHICS_WORK_BUFFER_SIZE F_GRAPHICS_MAX_HEIGHT_OF_FRAME_BUFFER F_GRAPHICS_QUALITY_FLAGS F_GRAPHICS_BACKGROUND_COLOR F_GRAPHICS_IS_FAST_MANUAL_FLUSH

	F_GRAPHICS_WORK_BUFFER_B_ADDRESS F_GRAPHICS_WORK_BUFFER_B_SIZE F_GRAPHICS_JPEG_DECODER F_GRAPHICS_INTERNAL_EVENT_VALUE F_GRAPHICS_LOCK_OBJECT
frame_buffer_t* frame_buffer	描画対象のフレームバッファー。（必須） frame_buffer_t 型構造体のアドレスを設定してください。
void* work_buffer_address	ワークバッファーの先頭アドレス。非キャッシュ領域のアドレスを設定してください。以下の場合に必須。 ● Raw 形式画像を描画するとき ● ワークバッファーB を使用するとき
size_t　work_buffer_size	内部で使うワークバッファーのサイズ（バイト） （work_buffer_address と同時に指定） 参考：(5.7.14.3)R_RGA_CalcWorkBufferSize
int_fast32_t max_height_of_frame_buffer	描画対象になるフレームバッファーのうち最大の高さ。 （work_buffer_address と同時に指定）
graphics_quality_flags_t quality_flags	描画品質。参考：(5.4.5.4)
r8g8b8a8_t background_color	背景色。参考：(5.7.1.22))
bool_t is_fast_manual_flush	高速手動フラッシュモード(5.11.4)かどうか
void* work_buffer_b_address	ワークバッファーB の先頭アドレス。以下の場合に必須。 ● フレームバッファーが 16 ビットカラーのときに、JPEG の左端の X 座標が 4 で割り切れないとき ● フレームバッファーが 32 ビットカラーのときに、JPEG の左端の X 座標が 2 で割り切れないとき ● JPEG 画像を行列変換して描画するとき
size_t work_buffer_b_size	ワークバッファーB のサイズ（バイト） （work_buffer_b_address と同時に指定） 参考：(5.7.14.4)R_RGA_CalcWorkBufferB_Size
graphics_jpeg_decoder_t jpeg_decoder	使用する JPEG デコーダー
bit_flags32_t internal_event_value	内部で使用するスレッド付属イベントの値。 省略時は、R_OSPL_UNUSED_FLAG。 アプリケーションは、ここで指定した値のイベントを待つことはできません。 OSPL のアプリケーションノートを参照。
BSP_CFG_USER_LOCKING_TYPE*　lock_object	RGA の使用権を管理するロック オブジェクト。 R_GRAPHICS_Initialize 関数～R_GRAPHICS_Finalize 関数の間でロックします。 OSPL のアプリケーションノートを参照。

5.4.5.4 graphics_quality_flags_t

描画品質を表すデフォルト可能フラグです。 「(5.11.7)デフォルト可能フラグ」を参照してください。

```
#include <RGA.h>
```

定数名	値	内容
GRAPHICS_RENDERING_QUALITY_ANTIALIASED	0x00001	境界線にアンチエイリアス有効
GRAPHICS_RENDERING_QUALITY_NONANTIALIASED	0x10000	境界線にアンチエイリアス無効
GRAPHICS_IMAGE_QUALITY_ANTIALIASED	0x00002	画像内に補間フィルター有効
GRAPHICS_IMAGE_QUALITY_NONANTIALIASED	0x20000	画像内に補間フィルター無効

5.4.5.5 window_surfaces_t

概　要	フレームバッファーと画面表示
ヘッダー	RGA.h, window_surfaces.h
説　明	メンバー関数に相当する関数：(5.7.2.2)
メンバー変数	アクセス禁止

5.4.5.6 window_surfaces_config_t

概　要　window_surfaces_t の設定

ヘッダー　RGA.h, window_surfaces.h

説　明

メンバー変数

bit_flags_fast32_t　flags	フラグド構造体パラメーター（必須）参考：(5.11.6)。 F_WINDOW_SURFACES_PIXEL_FORMAT F_WINDOW_SURFACES_LAYER_COUNT F_WINDOW_SURFACES_BUFFER_HEIGHT F_WINDOW_SURFACES_BACKGROUND_COLOR
char*　title;	予約
int_fast32_t　width	予約
int_fast32_t　height	予約
pixel_format_t pixel_format	表示ウィンドウのピクセルフォーマット。参考 (5.4.5.10)。 [条件] SH7269 のサンプルで指定できる値は、以下の通り。 ● PIXEL_FORMAT_ARGB8888 ● PIXEL_FORMAT_XRGB8888 ● PIXEL_FORMAT_RGB565（デフォルト） ● PIXEL_FORMAT_ARGB4444 ● PIXEL_FORMAT_ARGB1555 ● PIXEL_FORMAT_YCbCr422 ● PIXEL_FORMAT_CLUT1 ● PIXEL_FORMAT_CLUT4 ● PIXEL_FORMAT_CLUT8
pixel_format_t overlay_pixel_format	予約
int_fast32_t layer_count	作成するレイヤー数。デフォルト＝1。 [条件] SH7269 のサンプルで指定できる値は、1 のみ。
r8g8b8a8_t background_color	背景色。参考 (5.4.5.19)。 デフォルトは、R_RGA_Get_R8G8B8A8(255, 255, 255, 0)。
int_fast32_t buffer_height	フレームバッファーの高さ。デフォルトは、画面の高さ。

5.4.5.7 layer_attributes_t

概　要　R_WINDOW_SURFACES_AccessLayerAttributes 関数(5.7.9.10)のパラメーター
ヘッダー　RGA.h, window_surfaces.h
説　明　参考：(5.11.5)サンプル画面制御のレイヤー構造

メンバー変数

access_t　access	設定または取得
bit_flags_fast32_t　flags	フラグド構造体パラメーター（必須）参考：(5.11.6)。 F_LAYER_ID F_LAYER_LAYER_COLOR F_LAYER_CLUT F_LAYER_CLUT_COUNT
int_fast32_t　id	アクセスする対象となるレイヤー番号。 -1 は背景。（必須）参考：(5.11.5)
int_fast32_t　priority	予約
bool_t　is_show	予約
bool_t　is_color_key	予約
r8g8b8a8_t　color_key	予約
r8g8b8a8_t　layer_color	レイヤー全体の色。ID=-1 に対してのみ可能。参考：(5.4.5.19)。
int_fast32_t　x	予約
int_fast32_t　y	予約
int_fast32_t　width	予約
int_fast32_t　height	予約
int_fast32_t　offset_x	予約
int_fast32_t　offset_y	予約
uint32_t*　CLUT	CLUT(Color Look Up Table、パレット）に設定する色の配列。（オプション） graphics_image_properties_t (5.4.5.14) の CLUT を指定してください。 ピクセルフォーマットは、ARGB8888 です。 複数の CLUT 形式の画像を同時に表示しようとして、CLUT の色を上書きすると、上書された CLUT を持つ画像は正しく表示できません。
int_fast32_t　CLUT_count	CLUT に設定する CLUT の要素数（CLUT 変数と同時に指定） graphics_image_properties_t (5.4.5.14) の CLUT_count を指定してください。 [設定範囲] 最大は、CLUT8=256、CLUT4=16、CLUT1=2。

5.4.5.8 access_t

値を設定する（Write）か、取得する（Read）ことを示す定数。

```
#include "RGA.h" /* or window_surfaces.h" */
```

定数名	値	内容
ACCESS_READ	1	値を取得する
ACCESS_WRITE	2	値を設定する

5.4.5.9 byte_per_pixel_t

```
#include <RGA.h>
```

```
typedef int  byte_per_pixel_t;
```

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	

関連する関数：5.7.9.2

5.4.5.10 pixel_format_t

ピクセルフォーマットの型。

参考：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先のピクセルフォーマットの組み合わせ

```
#include  "RGA.h"  /* or "frame_buffer.h" */
```

定数名	値	値	内容
PIXEL_FORMAT_UNKNOWN	0	0x00	不明
PIXEL_FORMAT_ARGB8888	1	0x01	ARGB8888
PIXEL_FORMAT_RGB565	3	0x03	RGB565
PIXEL_FORMAT_ARGB4444	5	0x05	ARGB4444
PIXEL_FORMAT_A1	13	0x0D	1 ビット アルファ（予約）
PIXEL_FORMAT_A4	14	0x0E	4 ビット アルファ（予約）
PIXEL_FORMAT_A8	11	0x0B	8 ビット アルファ（予約）
PIXEL_FORMAT_RGB888	15	0x0F	RGB888（予約）
PIXEL_FORMAT_R8G8B8A8	6 | (1 << 4)	0x16	R8G8B8A8
PIXEL_FORMAT_XRGB8888	0 | (1 << 6)	0x40	XRGB8888
PIXEL_FORMAT_ARGB1555	4 | (1 << 6)	0x44	ARGB1555
PIXEL_FORMAT_YCbCr422	2 | (1 << 16)	0x10002	YCbCr422
PIXEL_FORMAT_YUV422	2 | (1 << 16)	0x10002	YUV422
PIXEL_FORMAT_YUV422_GRAY_SCALE_IS_0x80	2 | (1 << 16)	0x10002	YCbCr422 Cb, Cr = 0x80 でグレーになる
PIXEL_FORMAT_JPEG	12 | (2 << 8)	0x20C	JPEG
PIXEL_FORMAT_PNG	12 | (3 << 8)	0x30C	PNG （予約）
PIXEL_FORMAT_GIF	12 | (4 << 8)	0x40C	GIF （予約）
PIXEL_FORMAT_CLUT1	12 | (1 << 12)	0x100C	1 ビット CLUT
PIXEL_FORMAT_CLUT4	12 | (4 << 12)	0x400C	4 ビット CLUT
PIXEL_FORMAT_CLUT8	12 | (8 << 12)	0x800C	8 ビット CLUT

5.4.5.11 frame_buffer_delegate_t

frame_buffer_t の delegate メンバー変数から参照されるオブジェクトのクラスです。

frame_buffer_t を使うライブラリ、または、アプリケーションが定義することができます。

デフォルトでは、次のように定義されます。 よって、frame_buffer_t::delegate は、void* 型になります。

```
#include  <RGA.h>
```

```
typedef  void  frame_buffer_delegate_t;
```

型を変更するには、frame_buffer_delegate_t を定義しているヘッダー ファイルを #include する前に、型を定義して、#define frame_buffer_delegate_t frame_buffer_delegate_t すると、型を変更できます。

```
typedef  MyFrameBufferClass       frame_buffer_delegate_t;
#define  frame_buffer_delegate_t  frame_buffer_delegate_t

#include  <RGA.h>  /* default define frame_buffer_delegate_t */
```

5.4.5.12 v_sync_t

概　要	表示画面の V-Sync 信号に同期する機能のコンテキスト
ヘッダー	RGA.h, vsync.h
説　明	メンバー関数に相当する関数：(5.7.11)
メンバー変数	アクセス禁止

5.4.5.13 vram_ex_stack_t

概　要	外部 RAM に配置したオフスクリーン バッファーを取得できるスタック
ヘッダー	RGA.h, window_surfaces.h
説　明	メンバー関数に相当する関数：(5.7.12) SH7269 では、外部 RAM に配置されたフレームバッファーを直接表示コントローラーVDC4 で表示させず、外部 RAM に配置されたフレームバッファーを内蔵 RAM に配置されたフレームバッファーに転送して、表示させてください。
メンバー変数	アクセス禁止

5.4.5.14 graphics_image_properties_t

概　要	画像のプロパティ	
ヘッダー	RGA.h	
説　明	関連する関数：(5.7.2.6) R_GRAPHICS_IMAGE_GetProperties ソース画像をプログラムの中で生成するときは、(5.7.2.5) R_GRAPHICS_IMAGE_InitByShareFrameBuffer を呼び出してください。 [設定範囲] (5.7.2.5) を参照。	
メンバー変数	int_fast32_t　width	画像の幅（ピクセル）
	int_fast32_t　height	画像の高さ（ピクセル）
	uint8_t*　data	画像を構成するピクセルの配列の先頭アドレス。 R8G8B8A8 形式以外のときは、NULL
	void*　pixels	画像を構成するピクセルの配列の先頭アドレス。 R8G8B8A8 形式以外のときでも有効
	pixel_format_t pixelFormat	ピクセルフォーマット
	uint32_t*　CLUT	CLUT(Color Look Up Table、パレット）。色の配列。 CLUT を持たないときは、NULL。layer_attributes_t (5.4.5.7) に指定してください。
	int_fast32_t CLUT_count	CLUT の配列要素数。layer_attributes_t (5.4.5.7) に指定してください。

5.4.5.15 graphics_composite_operation_t

アルファブレンドするときの演算方法の型。

```
#include  "RGA.h"
```

定数名	値	内容
GRAPHICS_SOURCE_OVER	1	一般的な、アルファブレンドの演算（SRC over DST）を行います。 Premultiplied alpha の式かどうかは、描画する画像のヘッダーに含まれるフラグや、描画対象のピクセルフォーマットにアルファがあるかどうかによります。（デフォルト）
GRAPHICS_DESTINATION_OUT	7	逆アルファ マスク (Inverted alpha mask)を描画するときに使います。 描画対象（デスティネーション）のアルファ成分だけ、下記の式によって変化します。 a_dst = a_dst * (1 - a_src) アルファ成分を持たないフレームバッファーに描画するときは、GRAPHICS_DESTINATION_OUT を設定することができません。 最終的な描画対象が XRGB8888 のときは、ARGB8888 のバックバッファーに、逆アルファ マスクを描画してください。 R_GRAPHICS_SetGlobalAlpha 関数または globalAlpha プロパティは、255 を設定してください。 A8, A4, A1 形式のソース画像には対応していません。
GRAPHICS_COPY	0	アルファブレンドの演算を行わず、ソース画像のデータをそのまま描画します。その分だけ高速になる可能性があります。 高速化のために、バックバッファーにアルファ付き PNG 画像をあらかじめ伸張しておくときに使えば、伸張する前にバックバッファーをクリアしておく必要がなくなります。

5.4.5.16 graphics_status_t

概　要	グラフィックス描画コンテキストを保存する領域の型
ヘッダー	RGA.h
説　明	メンバー関数に相当する関数： ● (5.7.1.8) R_GRAPHICS_Save 関数 ● (5.7.1.9) R_GRAPHICS_Restore 関数
メンバー変数	アクセス禁止

5.4.5.17 graphics_matrix_float_t

行列の要素の型です。

```
#include  <RGA.h>
typedef  float  graphics_matrix_float_t;
```

5.4.5.18 repetition_t

パターンの繰り返し方法を指定する定数。

```
#include  "RGA.h"
```

定数名	値	内容
GRAPHICS_REPEAT	1	繰り返す

5.4.5.19 r8g8b8a8_t

概　要	バイトの並びが Red, Green, Blue, Alpha の順になっているピクセルフォーマット
ヘッダー	RGA.h
説　明	[設定範囲] red, green, blue, alpha のそれぞれが、0（黒、透明）～255（明るい、不透明） の値をとります。

メンバー変数

uint8_t　u.red	赤
uint8_t　u.green	緑
uint8_t　u.blue	青
uint8_t　u.alpha	アルファ、不透明度

5.4.5.20 animation_timing_function_t

概　要	アニメーションの動きを表す（ベジェ関数を表す）オブジェクト
ヘッダー	RGA.h
説　明	メンバー関数に相当する関数：(5.7.13)

メンバー変数

アクセス禁止

5.4.5.21 graphics_jpeg_decoder_t

JPEG デコーダーの種類を表す定数。

```
#include "RGA.h"
```

定数名	値	内容
GRAPHICS_JPEG_DECODER_NONE	0	JPEG デコーダーを使用しない。RGA 以外から JPEG ハードウェア デコーダーを使うことができるようになります。
GRAPHICS_JPEG_DECODER_HARD	1	JPEG ハードウェア デコーダー。（デフォルト）

5.4.6 文字列の書式

章	シンボル	内容
5.4.6.1	#rrggbb, #rgb	CSS Color の書式

5.4.6.1 #rrggbb, #rgb

CSS Color の書式を使って、色を 16 進数で表します。

対象：fillStyle (5.6.1.3)

例："#FFFF00" … 赤成分 255、緑成分 255、青成分 0、アルファ成分 1.0

例："#FF0" … 赤成分 255、緑成分 255、青成分 0、アルファ成分 1.0

5.4.7 移植層の型

5.4.7.1 一覧

型名	内容
NCGvoid	RGPNCG で扱う void 型の抽象
NCGenum	RGPNCG で扱う 列挙型の抽象 [4]
NCGboolean	RGPNCG で扱う _Bool 型の抽象
NCGbitfield	RGPNCG で扱う ビットフラグの抽象 [4]
NCGchar	RGPNCG で扱う 文字の型の抽象
NCGint8	RGPNCG で扱う int8_t 型の抽象 [4]
NCGint16	RGPNCG で扱う int16_t 型の抽象 [4]
NCGint32	RGPNCG で扱う int32_t 型の抽象
NCGint64	RGPNCG で扱う int64_t 型の抽象 [4]
NCGuint8	RGPNCG で扱う uint8_t 型の抽象 [4]
NCGuint16	RGPNCG で扱う uint16_t 型の抽象 [4]
NCGuint32	RGPNCG で扱う uint32_t 型の抽象
NCGuint64	RGPNCG で扱う uint64_t 型の抽象 [4]
NCGfloat32	RGPNCG で扱う IEEE 754 の単精度浮動小数の抽象 [4]
NCGfloat64	RGPNCG で扱う IEEE 754 の倍精度浮動小数の抽象 [4]
NCGsizei	RGPNCG で扱う サイズを格納する符号なし整数型 [4]
NCGclampf	RGPNCG で扱う 0.0～1.0 の範囲に限定した浮動小数の抽象 [4]
NCGfixed	RGPNCG で扱う 固定小数の抽象 [4]

[4] RGA では未使用

5.4.8 クラスの状態遷移

図 5-5 に graphics_t クラスの状態遷移図を示します。

表 5.6 に R_～_Finalize 関数を持つ各クラスの状態遷移を伴う関数一覧を示します。状態遷移を伴わない関数は、「通常」状態でのみ使用することができます。

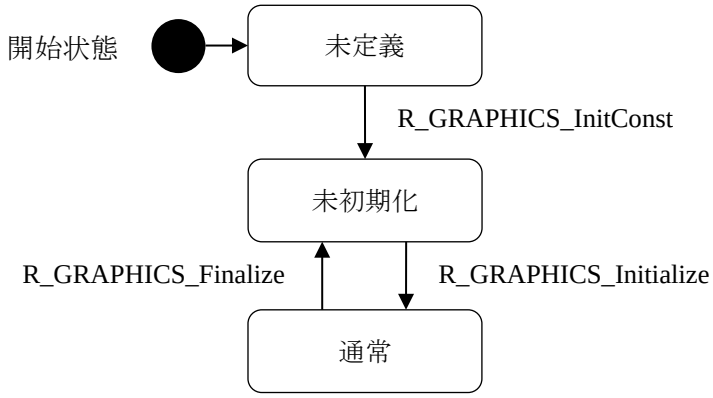


図 5-5 graphics_t クラスの状態遷移図

表 5.6 R_～_Finalize 関数を持つ各クラスの状態遷移を伴う関数一覧

クラス	未定義→未初期化	未初期化→通常	通常→未初期化
graphics_t	R_GRAPHICS_InitConst	R_GRAPHICS_Initialize	R_GRAPHICS_Finalize

図 5-6 に graphics_pattern_t クラスの状態遷移図を示します。

表 5.7 に R_～_Finalize 関数を持たないクラスの状態遷移を伴う関数一覧を示します。状態遷移を伴わない関数は、「通常」状態でのみ使用することができます。

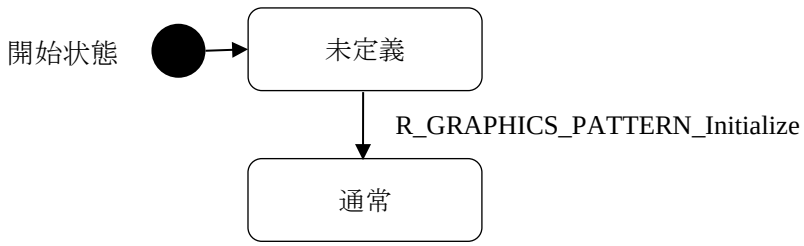


図 5-6 graphics_pattern_t クラスの状態遷移図

表 5.7 R_～_Finalize 関数を持たないクラスの状態遷移を伴う関数一覧

クラス	未定義→通常
graphics_image _t	R_GRAPHICS_IMAGE_InitR8G8B8A8 R_GRAPHICS_IMAGE_InitSameSizeR8G8B8A8 R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8 R_GRAPHICS_IMAGE_InitByShareFrameBuffer ※ImagePackager が生成したオブジェクトは、開始状態が「通常」状態です。
graphics_pattern_t	R_GRAPHICS_PATTERN_Initialize

5.5 変数一覧

表 5.8 にグローバル変数を、表 5.9 に static 型変数を、表 5.10 に const 型変数を示します。

表 5.8 グローバル変数

型	変数名	内容	使用関数
なし			

表 5.9 static 型変数

型	変数名	内容	使用関数
なし			

表 5.10 const 型変数

型	変数名	内容	使用関数
なし			

5.6 プロパティ

5.6.1 Canvas2D_ContextClass のプロパティ

5.6.1.1 一覧

章	プロパティ名	概要
5.6.1.2	c_LanguageContext	C 言語 API に使えるコンテキスト
5.6.1.3	fillStyle	塗りつぶし方。
5.6.1.4	globalAlpha	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）。
5.6.1.5	globalCompositeOperation	アルファブレンドするときの演算方法。

5.6.1.2 c_LanguageContext

```
graphics_t*  Canvas2D_ContextClass::c_LanguageContext;  /* get only */
```

C 言語 API に使えるコンテキスト。

C 言語 API で提供されていて、C++API で提供されていない機能を使うときに、本プロパティを参照してください。

5.6.1.3 fillStyle

```
char* Canvas2D_ContextClass::fillStyle;  /* set only */ /* CSS color */
Canvas2D_PatternClass  Canvas2D_ContextClass::fillStyle;  /* set only */
Canvas2D_GradientClass  Canvas2D_ContextClass::fillStyle;  /* set only */
```

塗りつぶし方法を指定します。以下に羅列するいずれかの型の値を持つプロパティです。

char* 型の場合、CSS Color で表現される単色塗りつぶし色を指定します。

Canvas2D_PatternClass 型の場合、パターンを指定します。

char*型を指定した場合、単色塗りつぶしになります。参考：#rrggbb, #rgb(5.4.6.1)、rgb(5.7.15.2)、rgba(5.7.15.3)

初期値は、不透明な黒です。

パターンを描画するときは、fillRect メソッドを使ってください。

5.6.1.4 globalAlpha

```
float32_t  Canvas2D_ContextClass::globalAlpha;  /* get,set */
```

すべての描画に対してかける単一のアルファ値（非透明度）を保持します。

デフォルト値は、1.0 です。

0.0 より小さい値を設定した場合、0.0 になります。

1.0 より大きい値を設定した場合、1.0 になります。

次の描画関数に影響します。

→ fillRect (Canvas2D_ContextClass) などの図形の塗りつぶし

→ fillRect (Canvas2D_ContextClass) のパターン描画

→ strokeRect (Canvas2D_ContextClass) などの図形の境界線の描画

→ drawImage (Canvas2D_ContextClass) などの画像の描画

次の描画関数には影響しません。

→ clearRect (Canvas2D_ContextClass)

5.6.1.5 globalCompositeOperation

```
char*  Canvas2D_ContextClass::globalCompositeOperation;  /* get,set */
```

アルファブレンドするときの演算方法を保持します。参考：(5.4.5.15) graphics_composite_operation_t

5.6.2 Canvas2D_ImageClass のプロパティ

5.6.2.1 一覧

章	プロパティ名	概要
5.6.2.2	src	画像データ。
5.6.2.3	width	画像の幅。
5.6.2.4	height	画像の高さ。
5.6.2.5	data	ピクセルの色成分が入った配列。

5.6.2.2 src

```
GraphicsImageClass*  Canvas2D_ImageClass::src;  /* set only */
```

ImagePackager から生成された Raw 形式画像データ。

5.6.2.3 width

```
int_t  Canvas2D_ImageClass::width;  /* get only */
```

画像の幅を保持します。

5.6.2.4 height

```
int_t  Canvas2D_ImageClass::height;  /* get only */
```

画像の高さを保持します。

5.6.2.5 data

```
uint8_t*  Canvas2D_ImageClass::data;  /* get only */
```

ピクセルの色成分が入った配列です。

左上が先頭。

R,G,B,A の順番。

R,G,B,A は、それぞれ 0～255。

5.7 関数／メソッド

5.7.1 graphics_t クラスのメンバー関数に相当する関数

5.7.1.1 一覧

章	関数名	概要
5.7.1.2	R_GRAPHICS_InitConst	定数部分を初期化します
5.7.1.3	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。
5.7.1.4	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
5.7.1.5	R_GRAPHICS_SetFrameBuffer	描画対象を変更します。
5.7.1.6	R_GRAPHICS_GetFrameBuffer	描画対象を取得します。
5.7.1.7	R_GRAPHICS_Finish	描画が完了するまで待ちます。
5.7.1.8	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。
5.7.1.9	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。
5.7.1.10	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。
5.7.1.11	R_GRAPHICS_SetMatrix_2x3	行列の各要素を設定します。(2x3)
5.7.1.12	R_GRAPHICS_SetMatrix_3x3	行列の各要素を設定します。 (3x3)
5.7.1.13	R_GRAPHICS_GetMatrix_3x3	行列の各要素を取得します。
5.7.1.14	R_GRAPHICS_TranslateMatrixI	現在の行列から移動させます。（整数型指定）
5.7.1.15	R_GRAPHICS_TranslateMatrix	現在の行列から移動させます。（浮動小数型指定）
5.7.1.16	R_GRAPHICS_ScaleMatrix	現在の行列から拡大縮小させます。
5.7.1.17	R_GRAPHICS_RotateMatrixDegree	現在の行列から回転させます。回転の中心は (0,0)
5.7.1.18	R_GRAPHICS_ShearMatrix	現在の行列からせん断変形させます。
5.7.1.19	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。
5.7.1.20	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。
5.7.1.21	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。
5.7.1.22	R_GRAPHICS_SetBackgroundColor	背景色を設定します。
5.7.1.23	R_GRAPHICS_GetBackgroundColor	背景色を取得します。
5.7.1.24	R_GRAPHICS_GetClearColor	R_GRAPHICS_Clear するときの色を取得します。
5.7.1.25	R_GRAPHICS_Clear	長方形の領域をクリアします。
5.7.1.26	R_GRAPHICS_DrawImage	画像を描画します。
5.7.1.27	R_GRAPHICS_DrawImageResized	画像を長方形の中に拡大縮小して描画します。
5.7.1.28	R_GRAPHICS_DrawImageChild	画像の一部を描画します。
5.7.1.29	R_GRAPHICS_FillRect	長方形を塗りつぶします。
5.7.1.30	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。
5.7.1.31	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。
5.7.1.32	R_GRAPHICS_BeginPath	デフォルト パスの内容を空にリセットします。
5.7.1.33	R_GRAPHICS_Rect	デフォルト パスに、長方形を追加します。
5.7.1.34	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。

5.7.1.35	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。
5.7.1.36	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。
5.7.1.37	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。
5.7.1.38	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。
5.7.1.39	R_GRAPHICS_STATIC_SetOnInitialize	graphics_config_t のデフォルト値を設定するコールバック関数を登録します。
5.7.1.40	R_GRAPHICS_STATIC_SetOnFinalize	R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数を登録します。
5.7.1.41	R_GRAPHICS_SetQualityFlags	描画品質を設定します。
5.7.1.42	R_GRAPHICS_GetQualityFlags	現在の描画品質を取得します。
5.7.1.43	R_GRAPHICS_SetStrokeColor	現在の境界線の色を設定します。
5.7.1.44	R_GRAPHICS_StrokeRect	長方形の辺を描画します。
5.7.1.45	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。
5.7.1.46	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。
5.7.1.47	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。

5.7.1.2 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：(5.11.8)内部変数を定数で初期化する関数について（*_initConst 関数）
引　数	GraphicsClass* self　　グラフィックス描画コンテキスト
リターン値	なし

5.7.1.3 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-GPVG)や JPEG コーデックユニット (JCU) を初期化します。 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ーに描画するときは、R_GRAPHICS_SetFrameBuffer を使用してフレームバッファーを切り替えてください。	
引　数	graphics_t* self	グラフィックス描画コンテキスト
	graphics_config_t* config	(5.4.5.3)を参照してください。
リターン値	エラーコード、正常=0、参考:(6.2),(5.11.2)	

5.7.1.4 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

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、参考:(5.11.9)(6.2)

5.7.1.5 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

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ー

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.6 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

graphics_t* self	グラフィックス描画コンテキスト
frame_buffer_t** out_frame_buffer	（出力）描画対象のフレームバッファー

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.7 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　高速手動フラッシュ モード(5.11.4)のときに、ハードウェア レンダリングのグラフィックス ライブラリの API を呼び出した後で、CPU からフレームバッファーの内容を、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、R_GRAPHICS_Finish 関数が呼ばれ、CPU キャッシュの管理も行います。

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、参考:(6.2)

5.7.1.8 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

引　数

graphics_t* self	グラフィックス描画コンテキスト
graphics_status_t* out_status	（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、参考:(6.2)

5.7.1.9 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

graphics_t* self	グラフィックス描画コンテキスト
graphics_status_t* status	コンテキストの設定値
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 または e、0=成功かつ e=0、参考:(5.11.9)(6.2)

5.7.1.10 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

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、参考:(6.2)

5.7.1.11 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　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_t* self	グラフィックス描画コンテキスト
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	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、参考:(6.2)

5.7.1.12 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	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_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t * matrix	3 行 3 列の行列（配列） (Matrix[0] Matrix[3] Matrix[6] Matrix[1] Matrix[4] Matrix[7] Matrix[2] Matrix[5] Matrix[8])
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.13 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		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t * out_matrix	（出力）3 行 3 列の行列（配列） (Matrix[0] Matrix[3] Matrix[6] Matrix[1] Matrix[4] Matrix[7] Matrix[2] Matrix[5] Matrix[8])
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.14 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_fast32_t　tx,　int_fast32_t　ty);
補　足	$M=M\cdot\begin{pmatrix}1 & 0 & tx\\ 0 & 1 & ty\\ 0 & 0 & 1\end{pmatrix}$ 【制限事項】画像の全体がフレームバッファーより外に移動する行列で画像を描画すると、RGA の動作が停止することがあります。

引　数		
	graphics_t* self	グラフィックス描画コンテキスト
	int_fast32_t　tx, int_fast32_t　ty	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、参考:(6.2)	

5.7.1.15 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

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

【制限事項】画像の全体がフレームバッファーより外に移動する行列で画像を描画すると、RGA の動作が停止することがあります。

引　数		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t tx graphics_matrix_float_t ty	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、参考:(6.2)	

5.7.1.16 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		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t tx graphics_matrix_float_t ty	倍率。 拡大縮小の中心は原点
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.17 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

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

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		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t angle	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、参考:(6.2)	

5.7.1.18 R_GRAPHICS_ShearMatrix

概　要　現在行列(M)をせん断変形します。

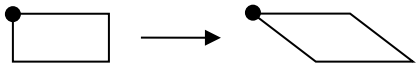
ヘッダー　RGA.h

宣　言
```
errnum_t　R_GRAPHICS_ShearMatrix( graphics_t* self,
        graphics_matrix_float_t shx, graphics_matrix_float_t shy );
```

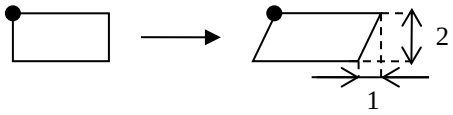
補　足

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

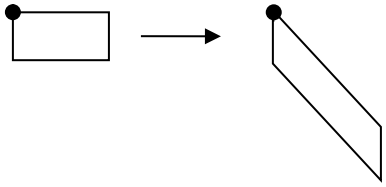
(shx, shy) = (1.0, 0.0) の場合、垂直の辺が 45 度に傾く平行四辺形になります。ただし、原点が長方形の左上になければ、移動を伴ってしまうので注意してください。



(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		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_matrix_float_t shx graphics_matrix_float_t shy	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、参考:(6.2)	

5.7.1.19 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

graphics_t* self	グラフィックス描画コンテキスト
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	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、参考:(6.2)

5.7.1.20 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

graphics_t* self	グラフィックス描画コンテキスト
graphics_matrix_float_t * matrix	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、参考:(6.2)

5.7.1.21 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
```
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_lLeft_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 );
```

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
graphics_matrix_float_t destination*	変換後の４点の座標
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、参考:(6.2)

5.7.1.22 R_GRAPHICS_SetBackgroundColor

概　要　背景色を設定します。

ヘッダー　RGA.h

宣　言　errnum_t　R_GRAPHICS_SetBackgroundColor(graphics_t* self, r8g8b8a8_t color);

補　足　アルファなしのフレームバッファー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。デフォルトは、白です。
アルファ付きのフレームバッファー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ーに描くクリア色は、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。背景色のデフォルトは、白です。

引　数

graphics_t* self	グラフィックス描画コンテキスト
r8g8b8a8_t color	背景色の取得は 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、参考:(6.2)

5.7.1.23 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

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

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.24 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		
	graphics_t* self	グラフィックス描画コンテキスト
	r8g8b8a8_t* out_color	（出力）　クリア色
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.25 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, int_fast32_t　width, int_fast32_t　height	長方形の領域
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.26 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。参照「(5.11.3)画像の形式の識別」。 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。 YUV422 形式に描画するときは、行列で変換した後の X 座標の値が偶数でなければ、エラーになります。 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ーにアルファ成分が含まれていないときは、フレームバッファーに描かれる 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ー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, ARGB1555 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, YUV422 CLUT 形式のフレームバッファーに描画するときは、フレームバッファー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 (5.4.5.14) の CLUT を表示コントローラーに設定してください。 高速手動フラッシュモード(5.11.4)の場合、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、フラッシュが必要です。ただし、ROM 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
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.27 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。

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
	int_fast32_t　width, int_fast32_t　height	描画先の幅と高さ
リターン値	エラーコード、正常=0、参考:(6.2)	

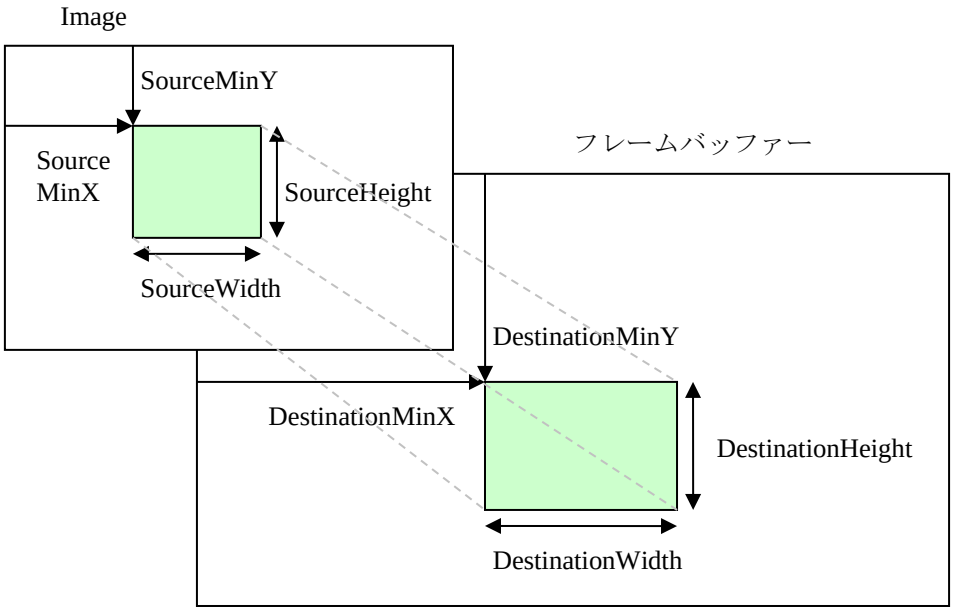
5.7.1.28 R_GRAPHICS_DrawImageChild

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

その他、R_GRAPHICS_DrawImage 関数の説明を参照。

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
int_fast32_t　source_width int_fast32_t　source_height	画像の中の、幅と高さ
int_fast32_t　destination_min_x int_fast32_t　destination_min_y	描画先の X, Y 座標の最小値
int_fast32_t　destination_width int_fast32_t　destination_height	描画先の幅と高さ

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.29 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
	int_fast32_t　width, int_fast32_t　height	長方形の幅と高さ
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.30 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、参考:(6.2)	

5.7.1.31 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 を使ってください。

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、参考:(6.2)	

5.7.1.32 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、参考:(6.2)	

5.7.1.33 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		
	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
	int_fast32_t　width, int_fast32_t　height	長方形の幅と高さ
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.34 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ーの一部のときに、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		
	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、参考:(6.2)	

5.7.1.35 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		
	graphics_t* self	グラフィックス描画コンテキスト
	uint8_t alpha_value	アルファ値。 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、参考:(6.2)	

5.7.1.36 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		
	graphics_t* self	グラフィックス描画コンテキスト
	uint8_t* out_alpha_value	（出力） アルファ値。 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、参考:(6.2)	

5.7.1.37 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		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_composite_operation_t composite_operation	演算方法
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.38 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(GraphicsClass* self, graphics_composite_operation_t*　out_composite_operation);
補　足	

引　数		
	graphics_t* self	グラフィックス描画コンテキスト
	graphics_composite_operation_t* out_composite_operation	（出力）演算方法
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.39 R_GRAPHICS_STATIC_SetOnInitialize

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を登録します。
ヘッダー	RGA.h
宣　言	errnum_t　R_GRAPHICS_STATIC_SetOnInitialize(R_GRAPHICS_OnInitialize_FuncType callback_function);
補　足	本関数を呼ばなかったときにコールバックされる関数は、 R_GRAPHICS_STATIC_OnInitializeDefault 関数です。

引　数		
	R_GRAPHICS_OnInitialize_FuncType callback_function	コールバック関数、または、NULL
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.1.40 R_GRAPHICS_STATIC_SetOnFinalize

概　要　R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数を登録します。

ヘッダー　RGA.h

宣　言
```
errnum_t　R_GRAPHICS_STATIC_SetOnFinalize(
    R_GRAPHICS_OnFinalize_FuncType callback_function );
```

補　足　本関数を呼ばなかったときにコールバックされる関数は、R_GRAPHICS_STATIC_OnFinalizeDefault 関数です。

引　数

R_GRAPHICS_OnFinalize_FuncType callback_function	コールバック関数、または、NULL

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.41 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

graphics_t* self,	グラフィックス描画コンテキスト
graphics_quality_flags_t qualities	(5.4.5.4)を参照してください

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.42 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	（出力）(5.4.5.4)を参照してください

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.1.43 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、参考:(6.2)

5.7.1.44 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、参考:(6.2)	

5.7.1.45 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	本関数は、高速手動フラッシュモード(5.11.4)のときに、呼び出す必要があります。

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、参考:(6.2)	

5.7.1.46 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	本関数は、高速手動フラッシュモード(5.11.4)のときに、呼び出す必要があります。 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、参考:(6.2)	

5.7.1.47 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	本関数は、高速手動フラッシュモード(5.11.4)のときに、呼び出す必要があります。 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、参考:(6.2)	

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

5.7.2.1 一覧

章	関数名	概要
5.7.2.2	R_GRAPHICS_IMAGE_InitR8G8B8A8	r8g8b8a8_t のイメージ オブジェクトを初期化します。
5.7.2.3	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。
5.7.2.4	R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8	表示中のフレームバッファー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。
5.7.2.5	R_GRAPHICS_IMAGE_InitByShareFrameBuffer	フレームバッファーの内容をイメージとして初期化します。
5.7.2.6	R_GRAPHICS_IMAGE_GetProperties	画像のプロパティを取得します

5.7.2.2 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。

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、参考:(6.2)

5.7.2.3 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。

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、参考:(6.2)

5.7.2.4 R_GRAPHICS_IMAGE_InitCopyFrameBufferR8G8B8A8

概　要　表示中のフレームバッファー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。

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_t* context	コピー元のフレームバッファーを描画対象にしたコンテキスト
int_fast32_t　min_x, int_fast32_t　min_y	取得する範囲の X, Y 座標の最小値 （フレームバッファー座標）
int_fast32_t　width, int_fast32_t　height	取得する範囲の幅と高さ

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.2.5 R_GRAPHICS_IMAGE_InitByShareFrameBuffer

概　要　フレームバッファー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。

引　数

graphics_image _t* self	画像
frame_buffer_t* frame_buffer	イメージの内容が入っているフレームバッファー

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.2.6 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	高速手動フラッシュ モード(5.11.4)の場合、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		
	graphics_image _t* self	画像
	graphics_image_properties_t* out_properties	（出力）画像のプロパティ (5.4.5.14)
リターン値	エラーコード、正常=0、参考:(6.2)	

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

5.7.3.1 一覧

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

5.7.3.2 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、参考:(6.2)

5.7.4 C++オブジェクトの生成に関する関数

5.7.4.1 一覧

章	関数名	概要
5.7.4.2	R_RGA_New_Canvas2D_ContextClass	Canvas2D_ContextClass のオブジェクトを生成します。
5.7.4.3	R_RGA_New_Canvas2D_ImageClass	Canvas2D_ImageClass のオブジェクトを生成します。

5.7.4.2 R_RGA_New_Canvas2D_ContextClass

概　要	Canvas2D_ContextClass のオブジェクトを生成します。	
ヘッダー	RGA.h	
宣　言	Canvas2D_ContextClass R_RGA_New_Canvas2D_ContextClass(Canvas2D_ContextConfigClass& in_out_Config); Canvas2D_ContextClass　R_RGA_New_Canvas2D_ContextClass(frame_buffer_t* in_frame_buffer);	
補　足	内部変数を初期化します。 オブジェクトを使用しなくなったら、destroy メソッドを呼び出してください。	
引　数	Canvas2D_ContextConfigClass& in_out_Config	(5.4.4.6)を参照してください。
	frame_buffer_t* in_frame_buffer	(5.4.5.2)を参照してください。
リターン値	Canvas2D コンテキスト オブジェクト、エラーのときは、undefined	

5.7.4.3 R_RGA_New_Canvas2D_ImageClass

概　要	Canvas2D_ImageClass のオブジェクトを生成します。	
ヘッダー	RGA.h	
宣　言	Canvas2D_ImageClass　R_RGA_New_Canvas2D_ImageClass();	
補　足	Canvas2D の new Image()に相当します。	
引　数	なし	
リターン値	Canvas2D_ImageClass のオブジェクト、エラーのときは、undefined	

5.7.5 Canvas2D_ContextClass のメンバー関数

5.7.5.1 一覧

章	関数名	概要
5.7.5.2	destroy	Canvas2D_ContextClass のオブジェクトを削除します。
5.7.5.3	clearError	Canvas2D_ContextClass のオブジェクトの中にあるエラーをクリアします。
5.7.5.4	clearRect	長方形の領域をクリアします。
5.7.5.5	save	コンテキストの設定値を内部スタックに退避します。
5.7.5.6	restore	コンテキストの設定値を、内部スタックから、コンテキストに戻します。
5.7.5.7	drawImage	画像を描画します。
5.7.5.8	createImageData	r8g8b8a8_t のイメージ オブジェクトを生成します。
5.7.5.9	getImageData	表示中のフレームバッファーの一部をコピーした、イメージ オブジェクトを生成します。
5.7.5.10	putImageData	画像を描画します。
5.7.5.11	fillRect	長方形の領域を塗りつぶします。
5.7.5.12	createPattern	パターン オブジェクトを生成します。
5.7.5.13	beginPath	デフォルト パスの内容を空にリセットします。
5.7.5.14	rect	デフォルト パスに、長方形を追加します。
5.7.5.15	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。
5.7.5.16	setTransform	行列の各要素を設定します。
5.7.5.17	translate	現在の行列を平行移動させます。
5.7.5.18	scale	現在の行列を拡大縮小させます。
5.7.5.19	rotate	現在の行列を回転させます。回転の中心は (0,0)
5.7.5.20	transform	現在の行列に対して指定した 2 行 3 列の行列を掛けます。

5.7.5.2 destroy (Canvas2D_ContextClass)

概　要	Canvas2D_ContextClass のオブジェクトを削除します。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::destroy();
補　足	

引　数		
	なし	
リターン値	なし	

5.7.5.3 clearError (Canvas2D_ContextClass)

概　要	Canvas2D_ContextClass のオブジェクトの中にあるエラーをクリアします。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::clearError();
補　足	Canvas2D には、本インターフェースはありません。

引　数		
	なし	
リターン値	なし	

5.7.5.4 clearRect (Canvas2D_ContextClass)

概　要　長方形の領域をクリアします。

ヘッダー　RGA.h

宣　言
```
void　Canvas2D_ContextClass::clearRect(
                    int_t MinX, int_t MinY, int_t Width, int_t Height );
```

補　足　クリア色の設定は、R_GRAPHICS_SetBackgroundColor()を使用してください。その第 1 引数には、c_LanguageContext (Canvas2D_ContextClass) プロパティを指定してください。
クリアするときの色は、R_GRAPHICS_GetClearColor 関数で取得できます。
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、クリッピングの影響を受けます。

引　数		
	int_t MinX, int_t MinY	長方形の X, Y 座標の最小値
	int_t Width, int_t Height	長方形の幅と高さ
リターン値	なし	

5.7.5.5 save (Canvas2D_ContextClass)

概　要　コンテキストの設定値を内部スタックに退避します。

ヘッダー　RGA.h

宣　言
```
void　Canvas2D_ContextClass::save();
```

補　足　内部スタック用に、内部で、ヒープ領域を確保します。

引　数		
	なし	
リターン値	なし	

5.7.5.6 restore (Canvas2D_ContextClass)

概　要　コンテキストの設定値を、内部スタックから、コンテキストに戻します。

ヘッダー　RGA.h

宣　言
```
void　Canvas2D_ContextClass::restore();
```

補　足　内部で、内部スタックとして使っていたヒープ領域を開放します。

引　数		
	なし	
リターン値	エラーコード、正常=0	

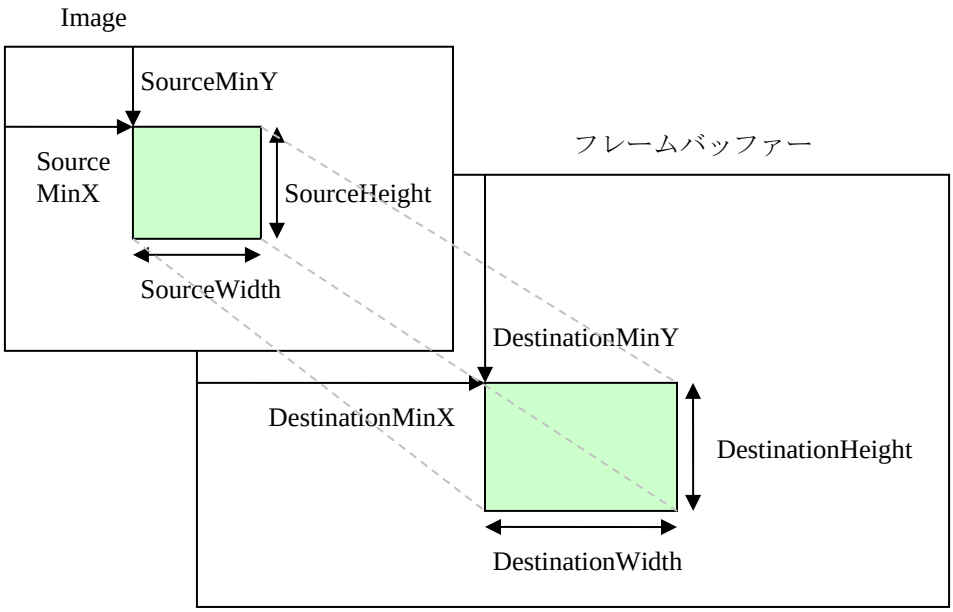
5.7.5.7 drawImage (Canvas2D_ContextClass)

概　要　画像を描画します。

ヘッダー　RGA.h

宣　言
```
void　Canvas2D_ContextClass::drawImage(
      GraphicsImageClass* Image,　int_t MinX,　int_t MinY );
void　Canvas2D_ContextClass::drawImage(
      GraphicsImageClass* Image,
      int_t MinX,　int_t MinY,　int_t Width,　int_t Height );
void　Canvas2D_ContextClass::drawImage(
      GraphicsImageClass* Image,
      int_t SourceMinX,　　int_t SourceMinY,
      int_t SourceWidth,　　int_t SourceHeight,
      int_t DestinationMinX,　int_t DestinationMinY,
      int_t DestinationWidth,　int_t DestinationHeight );
```

補　足　引数に直接 JPEG データなどを指定できます。参照「(5.11.3)画像の形式の識別」。
Width, Height を指定しなかったときは、描画先の幅と高さは、画像の幅と高さと同じになります。
SourceMinX ～ DestinationHeight を指定したときは、画像の一部を描画します。



SourceWidth ≠DestinationWidth または SourceHeight ≠DestinationHeight のときは、拡大縮小します。
Canvas2D_ContextClass::GlobalAlpha の影響を受けます。
現在の行列の影響を受けます。
YUV422 形式に描画するときは、MinX、または、MinY を行列で変換した後の値が偶数でなければ、エラーになります。
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ーにアルファ成分が含まれていないときは、フレームバッファーに描かれる 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ー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, ARGB1555
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, YUV422

引　数	GraphicsImageClass* Image	画像
	int_t MinX,　int_t MinY	描画先の X, Y 座標の最小値
	int_t Width,　int_t Height	描画先の幅と高さ
	int_t SourceMinX, int_t SourceMinY	画像の中の、X, Y 座標の最小値
	int_t SourceWidth, int_t SourceHeight	画像の中の、幅と高さ
	int_t DestinationMinX, int_t DestinationMinY	描画先の X, Y 座標の最小値
	int_t DestinationWidth, int_t DestinationHeight	描画先の幅と高さ
リターン値	なし	

5.7.5.8 createImageData (Canvas2D_ContextClass)

概　要　r8g8b8a8_t のイメージ オブジェクトを生成します。

ヘッダー　RGA.h

宣　言
```
Canvas2D_ImageClass　Canvas2D_ContextClass::createImageData(
    int_t Width, int_t Height );
Canvas2D_ImageClass　Canvas2D_ContextClass::createImageData(
    Canvas2D_ImageClass ImageReferencedWidthHeight );
```

補　足　内部でヒープ領域からメモリー領域を確保します。
内部で new 演算子を使っています。

引　数

int_t Width, int_t Height	イメージの幅と高さ
Canvas2D_ImageClass ImageReferencedWidthHeight	幅と高さを参照するイメージ オブジェクト

リターン値　生成した Image オブジェクト、エラー、またはメモリー不足＝undefined

5.7.5.9 getImageData (Canvas2D_ContextClass)

概　要　表示中のフレームバッファーの一部をコピーした、イメージ オブジェクトを生成します。

ヘッダー　RGA.h

宣　言
```
Canvas2D_ImageClass　Canvas2D_ContextClass::getImageData(
    int_t MinX,　int_t MinY,　int_t Width,　int_t Height );
```

補　足

引　数

int_t MinX,　int_t MinY	取得する範囲の X, Y 座標の最小値 （フレームバッファー座標）
int_t Width,　int_t Height	取得する範囲の幅と高さ

リターン値　生成した Image オブジェクト、エラー＝undefined

5.7.5.10 putImageData (Canvas2D_ContextClass)

概　要　画像を描画します。

ヘッダー　RGA.h

宣　言
```
void　Canvas2D_ContextClass::putImageData(
    Canvas2D_ImageClass ImageData,　int_t MinX,　int_t MinY );
void　Canvas2D_ContextClass::putImageData(
    Canvas2D_ImageClass ImageData,　int_t MinX,　int_t MinY,
    int_t DirtyX,　int_t DirtyY,　int_t DirtyWidth,　int_t DirtyHeight );
```

補　足　DirtyX, DirtyY, DirtyWidth, DirtyHeight を指定すると、Image オブジェクトの画像の一部だけ描画します。 拡大縮小はしません。

引　数

Canvas2D_ImageClass ImageData	描画する画像が入った Image オブジェクト
int_t MinX,　int_t MinY	描画先の X, Y 座標の最小値 （キャンバス座標）
int_t DirtyX,　int_t DirtyY	Image の中の最小 X, Y 座標 （Image 座標）
int_t DirtyWidth, int_t DirtyHeight	Image の中のサイズ ＝ 描画するサイズ

リターン値　なし

5.7.5.11 fillRect (Canvas2D_ContextClass)

概　要　長方形の領域を塗りつぶします。

ヘッダー　RGA.h

宣　言　void　Canvas2D_ContextClass::fillRect(
int_t MinX, int_t MinY, int_t Width, int_t Height);

補　足　現在の行列と現在の塗りつぶしのペイントの影響を受けます。
境界線は描画しません。
globalAlpha プロパティの影響を受けます。

引　数		
	int_t MinX, int_t MinY	長方形の X, Y 座標の最小値
	int_t Width, int_t Height	長方形の幅と高さ
リターン値	なし	

5.7.5.12 createPattern (Canvas2D_ContextClass)

概　要　パターン オブジェクトを生成します。

ヘッダー　RGA.h

宣　言　Canvas2D_PatternClass
Canvas2D_ContextClass::createPattern(GraphicsImageClass* Image, char* Repetition);

補　足　パターン オブジェクトは、fillStyle プロパティに設定してください。

引　数		
	GraphicsImageClass* Image	パターンの構成要素である画像
	char* Repetition	繰り返しの設定。 "repeat"を指定してください
リターン値	生成したパターン オブジェクト	

5.7.5.13 beginPath (Canvas2D_ContextClass)

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　void　Canvas2D_ContextClass::beginPath();

補　足

引　数		
	なし	
リターン値	なし	

5.7.5.14 rect (Canvas2D_ContextClass)

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　void　Canvas2D_ContextClass::rect(
int_t MinX, int_t MinY, int_t Width, int_t Height);

補　足

引　数		
	int_t MinX, int_t MinY	長方形の X, Y 座標の最小値
	int_t Width, int_t Height	長方形の幅と高さ
リターン値	なし	

5.7.5.15 clip (Canvas2D_ContextClass)

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	void　Canvas2D_ContextClass::clip();
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ーの一部のときに、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、restore を呼び出します。

引　数	なし	
リターン値	なし	

5.7.5.16 setTransform (Canvas2D_ContextClass)

概　要	現在行列の各要素を設定します。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::setTransform(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	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	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	なし	

5.7.5.17 translate (Canvas2D_ContextClass)

概　要	現在行列(M)を平行移動させます。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::translate(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	graphics_matrix_float_t tx, graphics_matrix_float_t ty	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	なし	

5.7.5.18 scale (Canvas2D_ContextClass)

概　要	現在行列(M)を拡大縮小させます。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::translate(graphics_matrix_float_t sx, graphics_matrix_float_t sy);
補　足	$M=M\cdot\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$

引　数		
	graphics_matrix_float_t tx, graphics_matrix_float_t ty	倍率。 拡大縮小の中心は原点
リターン値	なし	

5.7.5.19 rotate (Canvas2D_ContextClass)

概　要	現在行列(M)を回転させます。回転の中心座標は (0,0)です。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::rotate(graphics_matrix_float_t angle);
補　足	$M=M\cdot\begin{pmatrix} \cos(angle) & -\sin(angle) & 0 \\ \sin(angle) & \cos(angle) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

引　数		
	graphics_matrix_float_t angle	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	なし	

5.7.5.20 transform (Canvas2D_ContextClass)

概　要	現在行列(M)に対して指定した 2 行 3 列の行列を掛けます。
ヘッダー	RGA.h
宣　言	void　Canvas2D_ContextClass::transform(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	$M=M\cdot\begin{pmatrix} sx & kx & tx \\ ky & sy & ty \\ 0 & 0 & 1 \end{pmatrix}$

引　数		
	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	掛ける行列。 2 行 3 列の行列
リターン値	なし	

5.7.6 Canvas2D_ImageClass に関連する関数

5.7.6.1 一覧

章	関数名	概要
5.7.6.2	destroy	Canvas2D_ImageClass のオブジェクトを削除します。

5.7.6.2 destroy (Canvas2D_ImageClass)

概　要	Canvas2D_ImageClass のオブジェクトを削除します。	
ヘッダー	RGA.h	
宣　言	void　Canvas2D_ImageClass::destroy();	
補　足		
引　数	なし	
リターン値	なし	

5.7.7 Canvas2D_PatternClass に関連する関数

5.7.7.1 一覧

章	関数名	概要
5.7.7.2	destroy	Canvas2D_PatternClass のオブジェクトを削除します。

5.7.7.2 destroy (Canvas2D_PatternClass)

概　要	Canvas2D_PatternClass のオブジェクトを削除します。
ヘッダー	RGA.h
宣　言	void　Canvas2D_PatternClass::destroy();
補　足	

引　数	なし	
リターン値	なし	

5.7.8 WindowSurfacesClass のメンバー関数

5.7.8.1 一覧

章	関数名	概要
5.7.8.2	initialize	WindowSurfacesClass のオブジェクトを初期化します。
5.7.8.3	destroy	WindowSurfacesClass のオブジェクトを削除します。
5.7.8.4	get_layer_frame_buffer	フレームバッファーの情報を取得します。
5.7.8.5	swap_buffers	指定したレイヤーのバッファーをスワップして、描画していた内容を表示します。
5.7.8.6	alloc_offscreen_stack	オフスクリーン バッファーを VRAM のスタックから確保します
5.7.8.7	free_offscreen_stack	オフスクリーン バッファーを解放し、VRAM のスタックに返却します
5.7.8.8	do_message_loop	メッセージ ループに入ります。
5.7.8.9	access_layer_attributes	指定した表示レイヤーの属性にアクセスします。

5.7.8.2 initialize (WindowSurfacesClass)

概　要　WindowSurfacesClass のオブジェクトを削除します。

ヘッダー　RGA.h

宣　言　errnum_t　WindowSurfacesClass::initialize(WindowSurfacesConfigClass& in_out_config);

補　足　参考：(5.7.9.3) R_WINDOW_SURFACES_Initialize

引　数

WindowSurfacesConfigClass& in_out_config	各種パラメーター。参照：(5.4.4.7)

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.8.3 destroy (WindowSurfacesClass)

概　要　WindowSurfacesClass のオブジェクトを削除します。

ヘッダー　RGA.h

宣　言　void　WindowSurfacesClass::destroy();

補　足　参考：(5.7.9.4) R_WINDOW_SURFACES_Finalize

引　数

なし	

リターン値　なし

5.7.8.4 get_layer_frame_buffer (WindowSurfacesClass)

概　要　フレームバッファーの情報を取得します。

ヘッダー　RGA.h

宣　言　errnum_t　WindowSurfacesClass::get_layer_frame_buffer(int_fast32_t　layer_num, frame_buffer_t** out_frame_buffer);

補　足　参考：(5.7.9.5) R_WINDOW_SURFACES_GetLayerFrameBuffer

引　数

int_fast32_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） フレームバッファー構造体

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.8.5 swap_buffers (WindowSurfacesClass)

概　要	指定したレイヤーのバッファーをスワップして、描画していた内容を表示します。
ヘッダー	RGA.h
宣　言	errnum_t　WindowSurfacesClass::swap_buffers(int_fast32_t　layer_num, Canvas2D_ContextClass&　context);
補　足	参考：(5.7.9.7) R_WINDOW_SURFACES_SwapBuffers

引　数		
	int_fast32_t　layer_num	レイヤー番号。 0 が最も奥。 +1 が 1 つ手前。
	Canvas2D_ContextClass& context	描画を行ったグラフィックス コンテキスト
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.8.6 alloc_offscreen_stack (WindowSurfacesClass)

概　要	オフスクリーン バッファーを VRAM のスタックから確保します
ヘッダー	RGA.h
宣　言	errnum_t　WindowSurfacesClass::alloc_offscreen_stack(frame_buffer_t* in_out_frame_buffer);
補　足	参考：(5.7.9.11) R_WINDOW_SURFACES_AllocOffscreenStack

引　数		
	frame_buffer_t* in_out_frame_buffer	（入出力） オフスクリーン バッファー
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.8.7 free_offscreen_stack (WindowSurfacesClass)

概　要	オフスクリーン バッファーを解放し、VRAM のスタックに返却します
ヘッダー	RGA.h
宣　言	errnum_t　WindowSurfacesClass::free_offscreen_stack(const frame_buffer_t* frame_buffer);
補　足	参考：(5.7.9.12) R_WINDOW_SURFACES_FreeOffscreenStack

	frame_buffer_t** out_frame_buffer	（入出力）解放するオフスクリーン バッファー
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.8.8 do_message_loop (WindowSurfacesClass)

概　要	メッセージ ループに入ります。
ヘッダー	RGA.h
宣　言	errnum_t　WindowSurfacesClass::do_message_loop();
補　足	参考：(5.7.9.9) R_WINDOW_SURFACES_DoMessageLoop

引　数		
	なし	
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.8.9 access_layer_attributes (WindowSurfacesClass)

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
宣　言	errnum_t　WindowSurfacesClass::access_layer_attributes(LayerAttributesClass& in_out_Attributes);
補　足	参考：(5.7.9.10) R_WINDOW_SURFACES_AccessLayerAttributes

引　数		
	LayerAttributesClass& in_out_Attributes	（入出力） レイヤーの属性(5.4.4.8) 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、参考:(6.2)	

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

5.7.9.1 一覧

章	関数名	概要
5.7.9.2	R_WINDOW_SURFACES_InitConst	内部変数を定数で初期化します。
5.7.9.3	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。
5.7.9.4	R_WINDOW_SURFACES_Finalize	表示デバイスの終了処理をします。
5.7.9.5	R_WINDOW_SURFACES_GetLayerFrameBuffer	指定したレイヤーのフレームバッファー構造体へのポインターを取得します。
5.7.9.6	R_WINDOW_SURFACES_GetLayerCount	レイヤーの数を取得します。
5.7.9.7	R_WINDOW_SURFACES_SwapBuffers	指定したレイヤーのバッファーをスワップして、描画していた内容を表示します。
5.7.9.8	R_WINDOW_SURFACES_WaitForVSync	画面の V-Sync 信号を待ちます。
5.7.9.9	R_WINDOW_SURFACES_DoMessageLoop	メッセージ ループに入ります。
5.7.9.10	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。
5.7.9.11	R_WINDOW_SURFACES_AllocOffscreenStack	オフスクリーン バッファーを VRAM のスタックから確保します
5.7.9.12	R_WINDOW_SURFACES_FreeOffscreenStack	オフスクリーン バッファーを解放し、VRAM のスタックに返却します

5.7.9.2 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_SampleLib.h

宣　言　void　R_WINDOW_SURFACES_InitConst(window_surfaces_t* self);

補　足

引　数

window_surfaces_t* self	フレームバッファーと画面表示

リターン値　なし

5.7.9.3 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_SampleLib.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	フレームバッファーと画面表示
window_surfaces_config_t* in_out_config	(5.4.5.6)を参照してください。

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.9.4 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_SampleLib.h
宣　言	errnum_t　R_WINDOW_SURFACES_Finalize (window_surfaces_t* self, errnum_t e);
補　足	

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
	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、参考:(5.11.9)(6.2)	

5.7.9.5 R_WINDOW_SURFACES_GetLayerFrameBuffer

概　要	指定したレイヤーのフレームバッファー構造体へのポインターを取得します。
ヘッダー	RGA_SampleLib.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	参考：(5.11.5)サンプル画面制御のレイヤー構造 フレームバッファー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	フレームバッファーと画面表示
	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） フレームバッファー構造体
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.9.6 R_WINDOW_SURFACES_GetLayerCount

概　要　レイヤーの数を取得します。

ヘッダー　RGA_SampleLib.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	フレームバッファーと画面表示
	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、参考:(6.2)	

5.7.9.7 R_WINDOW_SURFACES_SwapBuffers

概　要　指定したレイヤーのバッファーをスワップして、描画していた内容を表示します。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_WINDOW_SURFACES_SwapBuffers(window_surfaces_t* self, int_t layer_num, graphics_t graphics);

補　足　シングルバッファーのときは、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。

Graphics = NULL を指定した場合、バッファーをスワップする前に、描画の完了を待たなくなります。

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
	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、参考:(6.2)	

5.7.9.8 R_WINDOW_SURFACES_WaitForVSync

概　要　画面の V-Sync 信号を待ちます。

ヘッダー　RGA.h, window_surfaces.h

宣　言　errnum_t　R_WINDOW_SURFACES_WaitForVSync(window_surfaces_t*　self, int_fast32_t　swap_interval,　bool_t　is_1_v_sync_at_minimum);

補　足　前回スワップしたときの V-Sync のカウンター値に、swap_interval 引数に指定した値を足したカウンター値になるまで待ちます。

is_1_v_sync_at_minimum = false のときは、SwapInterval を足した V-Sync のカウンター値に、すでになっていたらすぐに本関数から返ります。

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
	int_fast32_t　swap_interval	表示画面のチャンネル番号
	bool_t is_1_v_sync_at_minimum	前回待ちから抜けたときの V-Sync のカウンター値に足す値。0 以上
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.9.9 R_WINDOW_SURFACES_DoMessageLoop

概　要　メッセージ ループに入ります。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_WINDOW_SURFACES_DoMessageLoop(window_surfaces_t* self);

補　足　アプリケーションが終了したら、本関数から返ります。

終了方法は、サブ クラスの仕様を参照してください。

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.9.10 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_SampleLib.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	フレームバッファーと画面表示
	layer_attributes_t* in_out_attributes	（入出力）　レイヤーの属性(5.4.5.7)
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.9.11 R_WINDOW_SURFACES_AllocOffscreenStack

概　要	オフスクリーン バッファーを VRAM のスタックから確保します。
ヘッダー	RGA_SampleLib.h
宣　言	errnum_t　R_WINDOW_SURFACES_AllocOffscreenStack(window_surfaces_t* self,　frame_bufer_t* in_out_frame_buffer);
補　足	入力する frame_bufer_t のメンバー変数は、stride、height、buffer_count です。 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。 メモリー不足になったときは、E_FEW_ARRAY エラーが返ります。 R_WINDOW_SURFACES_Finalize が呼ばれたら、オフスクリーン バッファーも解放されます。

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
	frame_bufer_t* in_out_frame_buffer	（入出力）　オフスクリーン バッファー
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.9.12 R_WINDOW_SURFACES_FreeOffscreenStack

概　要	オフスクリーン バッファーを解放し、VRAM のスタックに返却します。
ヘッダー	RGA_SampleLib.h
宣　言	errnum_t　R_WINDOW_SURFACES_FreeOffscreenStack(window_surfaces_t* self,　frame_bufer_t* frame_buffer);
補　足	入力する frame_bufer_t のメンバー変数は、buffer_count、buffer_address です。 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。 R_WINDOW_SURFACES_AllocOffscreenStack を呼び出す前にエラーが発生して、終了処理で本関数が呼び出されたときに何もしないようにするには、*frame_buffer の変数を frame_buffer->buffer_count = 0 で初期化します。

引　数		
	window_surfaces_t* self	フレームバッファーと画面表示
	frame_bufer_t* frame_buffer	（入出力）解放するオフスクリーン バッファー
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.10 byte_per_pixel_t クラスに関連する関数

5.7.10.1 一覧

章	関数名	概要
5.7.10.2	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。
5.7.10.3	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。
5.7.10.4	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。

5.7.10.2 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）	

5.7.10.3 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	

5.7.10.4 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	

5.7.11 v_sync_t クラスに関連する関数

5.7.11.1 一覧

章	関数名	概要
5.7.11.2	R_V_SYNC_InitConst	内部変数を定数で初期化します。
5.7.11.3	R_V_SYNC_Initialize	V-Sync 割込みにアタッチします。
5.7.11.4	R_V_SYNC_Finalize	V-Sync 割込みからデタッチします。
5.7.11.5	R_V_SYNC_WaitForInterrupt	V-Sync 割込みが入るまで待ちます。

5.7.11.2 R_V_SYNC_InitConst

概　要　内部変数を定数で初期化します。

ヘッダー　RGA_SampleLib.h

宣　言　void　R_V_SYNC_InitConst(v_sync_t* self);

補　足

引　数	v_sync_t* self	V-Sync 割込み待ちのコンテキスト
リターン値	なし	

5.7.11.3 R_V_SYNC_Initialize

概　要　V-Sync 割込みにアタッチします。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_V_SYNC_Initialize(v_sync_t* self);

補　足　本関数の内部から、ユーザー定義関数 NCGDU_Attach_ISR をコールバックします。
すでにアタッチ済みのときは、エラーになります。
アプリケーションで V-Sync 割込みを管理しているときは、v_sync_t クラスを使うことができません。
本関数は、window_surfaces_t クラスの内部で使っています。そのため、window_surfaces_t クラスを使うときは、v_sync_t クラスを使うことはできません。

引　数	v_sync_t* self	V-Sync 割込み待ちのコンテキスト
リターン値	エラーコード、正常=0	

5.7.11.4 R_V_SYNC_Finalize

概　要　V-Sync 割込みからデタッチします。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_V_SYNC_Finalize(v_sync_t* self, errnum_t e);

補　足

引　数	v_sync_t* self	V-Sync 割込み待ちのコンテキスト
リターン値	エラーコード または e、0=成功かつ e=0	

5.7.11.5 R_V_SYNC_WaitForInterrupt

概　要	V-Sync 割込みが入るまで待ちます。
ヘッダー	RGA_SampleLib.h
宣　言	errnum_t　R_V_SYNC_WaitForInterrupt(v_sync_t* self, int_t swap_interval, bool_t is_1_v_sync_at_minimum);
補　足	前回スワップしてから、swap_interval に指定した回数まで V-Sync が入るまで待ちます。 is_1_v_sync_at_minimum = false のときは、swap_interval に指定した回数まで、すでに V-Sync が入っていたらすぐに本関数から返ります。

引　数		
	v_sync_t* self	V-Sync 割込み待ちのコンテキスト
	int_t swap_interval	フレームバッファーをスワップするまでの V-Sync の数
	bool_t is_1_v_sync_at_minimum	少なくとも 1 回は V-Sync を待つかどうか
リターン値	エラーコード、正常=0、参考:(6.2)	

5.7.12 vram_ex_stack_t クラスに関連する関数

5.7.12.1 一覧

章	関数名	概要
5.7.12.2	R_VRAM_EX_STACK_Initialize	初期化します
5.7.12.3	R_VRAM_EX_STACK_Alloc	外部 RAM からオフスクリーン バッファーを取得します
5.7.12.4	R_VRAM_EX_STACK_Free	外部 RAM にオフスクリーン バッファーを返却します

5.7.12.2 R_VRAM_EX_STACK_Initialize

概　要　初期化します。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_VRAM_EX_STACK_Initialize(vram_ex_stack_t* self, void* null_config);

補　足　再初期化を行うと、今まで確保されていたオフスクリーン バッファーはすべて返却されます。

引　数

vram_ex_stack_t* self	外部 RAM に配置された VRAM スタック領域
void* null_config	予約領域。NULL を指定してください。

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.12.3 R_VRAM_EX_STACK_Alloc

概　要　外部 RAM からオフスクリーン バッファーを取得します。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_VRAM_EX_STACK_Alloc(vram_ex_stack_t* self, frame_bufer_t* in_out_frame_buffer);

補　足　入力する frame_bufer_t のメンバー変数は、stride、height、buffer_count です。
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。
メモリー不足になったときは、E_FEW_ARRAY エラーが返ります。
R_VRAM_EX_STACK_Initialize が呼ばれたら、オフスクリーン バッファーも解放されます。

引　数

vram_ex_stack_t* self	外部 RAM に配置された VRAM スタック領域
frame_bufer_t* in_out_frame_buffer	（入出力） オフスクリーン バッファー

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.12.4 R_VRAM_EX_STACK_Free

概　要　外部 RAM にオフスクリーン バッファーを返却します。

ヘッダー　RGA_SampleLib.h

宣　言　errnum_t　R_VRAM_EX_STACK_Free(vram_ex_stack_t* self, frame_bufer_t* frame_buffer);

補　足　入力する frame_bufer_t のメンバー変数は、buffer_count、buffer_address です。
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。

引　数

vram_ex_stack_t* self	外部 RAM に配置された VRAM スタック領域
frame_bufer_t* frame_buffer	解放するオフスクリーン バッファー

リターン値　エラーコード、正常=0、参考:(6.2)

5.7.13 animation_timing_function_t クラスに関連する関数

5.7.13.1 一覧

章	関数名	概要
5.7.13.2	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
5.7.13.3	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

5.7.13.2 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

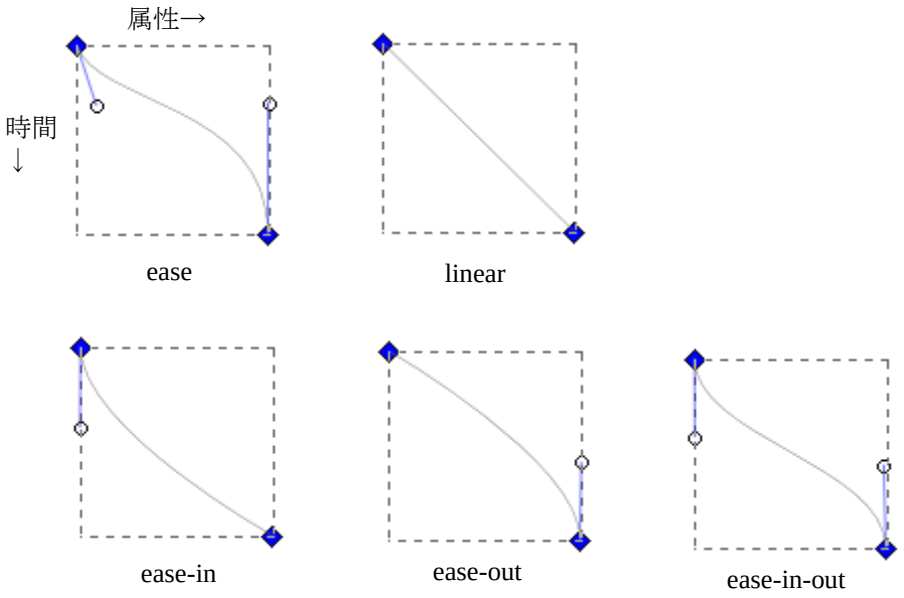
引　数

char* timing_name	アニメーション タイミングの名前。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、参考:(6.2)

5.7.13.3 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	

5.7.14 その他の関数

5.7.14.1 一覧

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

5.7.14.2 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 0～255
リターン値	R8G8B8A8 カラーの値	

5.7.14.3 R_RGA_CalcWorkBufferSize

概　要	ワークバッファー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_CalcWorkBufferSize(int_t MaxHeightOfFrameBuffer);	
補　足	将来、パラメーターが変わる可能性があります。 R_RGA_CalcWorkBufferSize は #define マクロです。 ワークバッファーに必要なサイズ： ディスプレイ リストの最大長 + 64 * （フレームバッファーの最大の高さ、８の倍数） *4*2 [Byte] ディスプレイ リストの最大長 = 128 参考：(5.4.5.3)graphics_config_t	
引　数	int_t MaxHeightOfFrameBuffer	描画先になるフレームバッファーの最大の高さ
リターン値	ワークバッファーに必要なサイズ（バイト）	

5.7.14.4 R_RGA_CalcWorkBufferB_Size

概　要　ワークバッファー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。
ワークバッファーB に必要なサイズ：ceil_16(MaxWidthOfJPEG) * ceil_16(MaxHeightOfJPEG) * MaxBytePerPixelOfFrameBuffer
ただし、ceil_16 は、16 の倍数に切り上げ。

JPEG コーデックユニット (JCU) が直接出力できる下記の条件をすべて満たすとき、ワークバッファーB は不要です。

- JPEG 画像の左上の位置が、描画先のアドレスが 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。つまり、JPEG 画像が YCbCr422 のとき、16 ピクセル×8 ラインの倍数、YCbCr420 のとき、16 ピクセル×16 ラインの倍数のとき
- 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。参考：(5.4.5.3)graphics_config_t

引　数

int_t MaxWidthOfJPEG	JPEG 画像の最大の幅
int_t MaxHeightOfJPEG	JPEG 画像の最大の高さ
int_t MaxBytePerPixelOfFrameBuffer	描画先になるフレームバッファー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　ワークバッファーB に必要なサイズ（バイト）

5.7.15 文字列内の関数

5.7.15.1 一覧

章	関数名	概要
5.7.15.2	rgb	CSS Color 形式で指定した色を返します。
5.7.15.3	rgba	CSS Color 形式で指定した色を返します。アルファ付き。

5.7.15.2 rgb

概　要	CSS Color 形式で指定した色を返します。	
ヘッダー	RGA.h	
宣　言	r8g8b8a8_t　rgb(int_t　red_max255, int_t　green_max255, int_t　blue_max255);	
補　足	例："rgb(255, 255, 0)" 対象：fillStyle (5.6.1.3) アルファ成分は、最大値（1.0）になります。	
引　数	int_t　red_max255	赤色成分 0～255
	int_t　green_max255	緑色成分 0～255
	int_t　blue_max255	青色成分 0～255
リターン値	R8G8B8A8 カラーの値	

5.7.15.3 rgba

概　要	CSS Color 形式で指定した色を返します。	
ヘッダー	RGA.h	
宣　言	r8g8b8a8_t　rgba(int_t　red_max255, int_t　green_max255, int_t　blue_max255, float32_t　alpha_max1);	
補　足	例："rgba(255, 255, 0, 0.5)" 対象：fillStyle (5.6.1.3)	
引　数	int_t　red_max255	赤色成分 0～255
	int_t　green_max255	緑色成分 0～255
	int_t　blue_max255	青色成分 0～255
	float32_t　alpha_max1	アルファ成分 0.0～1.0
リターン値	R8G8B8A8 カラーの値	

5.8 移植層の関数

本モジュールから、コールバックされる OS やボードを移植するときに変更する関数です。パッケージにはサンプルが含まれていますが、ユーザーが変更することができます。

5.8.1 RGA のデフォルト設定に関する関数

章	関数名	概要
5.8.1.1	R_GRAPHICS_STATIC_OnInitializeDefault	graphics_config_t のデフォルト値を設定するコールバック関数のデフォルト。
5.8.1.2	R_GRAPHICS_OnInitialize_FuncType	graphics_config_t のデフォルト値を設定するコールバック関数の型。
5.8.1.3	R_GRAPHICS_STATIC_OnFinalizeDefault	R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数のデフォルト。
5.8.1.4	R_GRAPHICS_OnFinalize_FuncType	R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数の型。

5.8.1.1 R_GRAPHICS_STATIC_OnInitializeDefault

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のデフォルト。

ヘッダー　RGA.h

宣　言　errnum_t　R_GRAPHICS_STATIC_OnInitializeDefault(graphics_t* self,
graphics_config_t* in_out_config, void** out_default_object);

補　足　R_GRAPHICS_OnInitialize_FuncType 関数型のデフォルトです。
参考：(5.8.1.2) R_GRAPHICS_OnInitialize_FuncType

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	(5.4.5.3)を参照してください。
void** out_default_object	（出力）　本関数の内部で確保したメモリーなど

リターン値　エラーコード、正常=0、参考:(6.2)

5.8.1.2 R_GRAPHICS_OnInitialize_FuncType

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の型。

ヘッダー　RGA.h

宣　言　errnum_t　R_GRAPHICS_OnInitialize_FuncType(graphics_t* self,
graphics_config_t* in_out_config, void** out_default_object);

補　足　本関数型のコールバック関数の登録は、R_GRAPHICS_STATIC_SetOnInitialize()を使用してください。
本関数型のコールバック関数は、R_GRAPHICS_Initialize() の中からコールバックされます。
out_default_object は、R_GRAPHICS_OnInitialize_FuncType()の内部で開放するためだけに用いられます。 R_GRAPHICS_OnInitialize_FuncType()をコールバックさせないときは、 out_default_object を設定しなくてもかまいません。

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	(5.4.5.3)を参照してください。
void** out_default_object	（出力）　本関数の内部で確保したメモリーなど

リターン値　エラーコード、正常=0、参考:(6.2)

5.8.1.3 R_GRAPHICS_STATIC_OnFinalizeDefault

概　要	R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数のデフォルト。
ヘッダー	RGA.h
宣　言	errnum_t　R_GRAPHICS_STATIC_OnFinalizeDefault(GraphicsClass* self,　void* default_object, errnum_t e);
補　足	R_GRAPHICS_OnFinalize_FuncType 関数型のデフォルトです。 参考：(5.8.1.4) R_GRAPHICS_OnFinalize_FuncType

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* default_object	（出力）　本関数の内部で確保したメモリーなど
	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、参考:(5.11.9)(6.2)	

5.8.1.4 R_GRAPHICS_OnFinalize_FuncType

概　要	R_GRAPHICS_OnInitialize_FuncType の関数の内部で確保したメモリーなどを開放する関数の型。
ヘッダー	RGA.h
宣　言	errnum_t　R_GRAPHICS_OnFinalize_FuncType(GraphicsClass* self, 　　void* default_object, errnum_t e);
補　足	本関数型のコールバック関数の登録は、R_GRAPHICS_STATIC_SetOnFinalize()を使用してください。 本関数型のコールバック関数は、R_GRAPHICS_Finalize()　関数の中の最後からコールバックされます。 default_object　は、R_GRAPHICS_OnInitialize_FuncType()関数の *out_default_object　に出力した値が入っています。

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* default_object	（出力）　本関数の内部で確保したメモリーなど
	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、参考:(5.11.9)(6.2)	

5.8.2 キャッシュに関する関数

章	関数名	概要
5.8.2.1	NCGSYS_WriteBackAndInvalidate	キャッシュから RAM にライトバック

5.8.2.1 NCGSYS_WriteBackAndInvalidate

概　要　キャッシュに入っているデータを、RAM にライトバック＆無効化する。

ヘッダー　RGA_Callback.h

宣　言　errnum_t　NCGSYS_WriteBackAndInvalidate(void* start, void* end);

補　足

引　数

void* start	開始仮想アドレス
void* end	終了仮想アドレス　（終了の次ではない）

リターン値　エラーコード、正常=0、参考:(6.2)

5.8.3 表示コントローラー割込みに関する関数

章	関数名	概要
5.8.3.1	NCGDU_Attach_ISR	表示コントローラー割込みを割込み処理関数にアタッチする
5.8.3.2	NCGDU_Detach_ISR	表示コントローラー割込みから割込み処理関数をデタッチする

5.8.3.1 NCGDU_Attach_ISR

概　要	表示コントローラー割込みを割込み処理関数にアタッチする。
ヘッダー	ncg_du_isr.h
宣　言	NCGint32　NCGDU_Attach_ISR(NCGISRfp pfnInterrupt);
補　足	RGA からコールバックしたときに引数に渡る割込み処理関数は、ビデオディスプレイコントローラ 4 (VDC4) ドライバーの VDC4_RegistCallbackFunc 関数に直接渡すことができます。 引数に渡された割込み処理関数をアタッチすることとは別に、本関数の内部、または、デバイスを使い始める前に、ビデオディスプレイコントローラ 4 (VDC4) ドライバーの割込みハンドラーを登録してください。(5.9.2)

引　数		
	NCGISRfp pfnInterrupt	アタッチする割込み処理関数
リターン値	エラーコード、正常=NCG_no_err エラーコートの例：NCG_err_isr_management_failed	

5.8.3.2 NCGDU_Detach_ISR

概　要	表示コントローラー割込みから割込み処理関数をデタッチする。
ヘッダー	ncg_du_isr.h
宣　言	NCGint32　NCGDU_Detach_ISR(NCGISRfp pfnInterrupt);
補　足	

引　数		
	NCGISRfp pfnInterrupt	デタッチする割込み処理関数
リターン値	エラーコード、正常=NCG_no_err エラーコートの例：NCG_err_isr_management_failed	

5.8.4 OSPL による移植層の関数

RGA は、以下の OSPL API 関数を使用しています。メモーマップや OS を変更したときは、OSPL API 関数の内容を変更するか変更済みのものに置き換えてください。 詳しくは、SH7268/SH7269 グループ OS 移植層（OSPL）サンプルプログラム（R01AN2339JJ）のドキュメントを参照してください。

- メモリーマップ依存の関数
  - R_OSPL_ToCachedAddress
  - R_OSPL_ToUncachedAddress
  - R_OSPL_ToPhysicalAddress
- OS 依存の関数
  - R_OSPL_THREAD_GetCurrentId
  - R_OSPL_EVENT_Set
  - R_OSPL_EVENT_Clear
  - R_OSPL_EVENT_Wait
  - R_OSPL_MEMORY_Flush
  - R_OSPL_Delay
- コンパイラーまたは OS 依存の関数
  - INLINE
  - STATIC_INLINE
  - R_OSPL_SECTION
  - R_OSPL_ALIGNMENT
  - R_OSPL_EnableAllInterrupt
  - R_OSPL_DisableAllInterrupt
  - R_OSPL_MEMORY_Barrier
- ハードウェア依存の関数
  - R_OSPL_FTIMER_InitializeIfNot
  - R_OSPL_FTIMER_Get
- 移植が不要な関数
  - （略）

5.8.5 RGPNCG による移植層の関数

RGA は、以下の RGPNCG 関数を使用しています。RGPNCG は、OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) に関わる処理の移植層です。OS を変更したときは、RGPNCG 関数の内容を変更するか変更済みのものに置き換えてください。NCGSYS_*State 関数の中で OSPL のイベントフラグを使うときは、NCGSYS_*State 関数の移植は不要です。

- OS 依存の関数

章	関数名	概要
5.8.5.1	NCGSYS_CreateState	RGPNCG のイベントフラグを生成します。
5.8.5.2	NCGSYS_DestroyState	RGPNCG のイベントフラグを削除します。
5.8.5.3	NCGSYS_SetState	RGPNCG のイベントフラグの値を変更します。
5.8.5.4	NCGSYS_GetState	RGPNCG のイベントフラグの値を取得します。
5.8.5.5	NCGSYS_WaitState	RGPNCG のイベントフラグを待ちます。
5.8.5.6	NCGSYS_SetStateEventValue	使用する OS のイベントフラグの値を設定します。
5.8.5.7	NCGSYS_GetLastCreatedState	直前に生成したイベントフラグを返します。
5.8.5.8	NCGSYS_SetNextStateEventValue	次に生成するイベントフラグの値を設定します。
5.8.5.9	NCGVG_Attach_ISR	R-GPVG の割込みコールバック関数を登録します。
5.8.5.10	NCGVG_Detach_ISR	R-GPVG の割込みコールバック関数を登録解除します。
5.8.5.11	NCGVGISRfp	R-GPVG の割込みコールバック関数の型

- 低消費電力モードのポリシーによる関数

章	関数名	概要
5.8.5.12	NCGVG_Init	R-GPVG の使用を開始します。
5.8.5.13	NCGVG_DeInit	R-GPVG の使用を終了します。

- 移植が不要な関数
  - NCGSYS_Abort
  - NCGSYS_CPUVAddrToSysPAddr
  - NCGSYS_ReadReg
  - NCGSYS_WriteReg

5.8.5.1 NCGSYS_CreateState

概　要	RGPNCG のイベントフラグを生成します。	
ヘッダー	ncg_state.h	
宣　言	NCGint32　NCGSYS_CreateState(NCGvoid** ppObj, NCGuint32 ui32StateID);	
補　足	OS に応じて内容を変更してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。	
引　数	NCGvoid** ppObj	（出力）イベントフラグ オブジェクト
	NCGuint32 ui32StateID	使われていません
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.2 NCGSYS_DestroyState

概　要	RGPNCG のイベントフラグを削除します。
ヘッダー	ncg_state.h
宣　言	NCGint32　NCGSYS_DestroyState(NCGvoid *pObj);
補　足	OS に応じて内容を変更してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。

引　数		
	NCGvoid* pObj	イベントフラグ オブジェクト
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.3 NCGSYS_SetState

概　要	RGPNCG のイベントフラグの値を変更します。
ヘッダー	ncg_state.h
宣　言	NCGint32　NCGSYS_SetState(NCGvoid* pObj, NCGuint32 ui32State, NCGuint32 ui32Flags);
補　足	OS に応じて内容を変更してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。

引　数		
	NCGvoid* pObj	イベントフラグ オブジェクト
	NCGuint32 ui32State	ui32Flags 引数によって変わります
	NCGuint32 ui32Flags	NCGSYS_STATE_SET_SET のとき： イベントフラグの全ビットを ui32State 引数の値に変更します NCGSYS_STATE_SET_OR のとき： イベントフラグと論理和をします。 ui32State 引数に指定した値が 1 になっている桁のイベントフラグのビットを 1 にします。 NCGSYS_STATE_SET_AND のとき： イベントフラグと論理積をします。 ui32State 引数に指定した値が 0 になっている桁のイベントフラグのビットを 0 にします。
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.4 NCGSYS_GetState

概　要	RGPNCG のイベントフラグの値を取得します。
ヘッダー	ncg_state.h
宣　言	NCGuint32　NCGSYS_GetState(NCGvoid* pObj, NCGuint32 ui32Flags);
補　足	OS に応じて内容を変更してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。

引　数		
	NCGvoid* pObj	イベントフラグ オブジェクト
	NCGuint32 ui32Flags	使われていません
リターン値	イベントフラグの値	

5.8.5.5 NCGSYS_WaitState

概　要	RGPNCG のイベントフラグの値が指定した値になるまで待ちます。
ヘッダー	ncg_state.h
宣　言	NCGint32　NCGSYS_WaitState(NCGvoid* pObj, NCGuint32 ui32State, NCGuint32 ui32Flags, NCGuint32 ui32Timeout);
補　足	OS に応じて内容を変更してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。 ステートのクリアはされません。 ２つのビットが立っているとき、片方のビットを待つために本関数を呼び出した後で、もう片方のビットを待つために本関数を呼び出すこともできます。 １回のビットを立てた後、同じビットに対して本関数を何度も呼ぶことができます。

引　数		
	NCGvoid* pObj	イベントフラグ オブジェクト
	NCGuint32 ui32State	１になるのを待っているイベントフラグのビットを１にした値
	NCGuint32 ui32Flags	NCGSYS_STATE_WAIT_AND のとき： ui32State 引数に 1 を指定したビットがすべて 1 になるまで待ちます。 NCGSYS_STATE_WAIT_OR のとき： ui32State 引数に 1 を指定したビットのどれかが 1 になるまで待ちます。
	NCGuint32 ui32Timeout	NCG_TIMEOUT_INFINITE が渡されます
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.6 NCGSYS_SetStateEventValue

概　要	使用する OS のスレッド付属イベントフラグの値を設定します。
ヘッダー	ncg_state.h
宣　言	NCGvoid　NCGSYS_SetStateEventValue (NCGvoid* pObj,　NCGuint32 ui32EventValue);
補　足	本関数が定義されていないときは、定義してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。 NCGSYS_SetState 関数に指定するイベントフラグは、NCG 内部の変数です。 NCGSYS_SetStateEventValue 関数に指定するスレッド付属イベントフラグは、OS のオブジェクトです。

引　数		
	NCGvoid* pObj	NCG のイベントフラグ
	NCGuint32 ui32EventValue	使用する OS のスレッド付属イベントの値
リターン値	なし	

5.8.5.7 NCGSYS_GetLastCreatedState

概　要	直前に生成したイベントフラグを返します。
ヘッダー	ncg_state.h
宣　言	NCGvoid*　NCGSYS_GetLastCreatedState(void);
補　足	本関数が定義されていないときは、定義してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。 RGA では、NCGSYS_CreateState 関数で生成したイベントフラグを返します。 NCGSYS_CreateState 関数が呼び出されてから、NCGSYS_GetLastCreatedState 関数が呼び出される間は、他のスレッドによって割り込まれることはありません。

引　数		
	なし	
リターン値	直前に生成したイベントフラグ	

5.8.5.8 NCGSYS_SetNextStateEventValue

概　要	次に生成するイベントフラグの値を設定します。
ヘッダー	ncg_state.h
宣　言	NCGvoid　NCGSYS_SetNextStateEventValue (NCGuint32 ui32EventValue);
補　足	本関数が定義されていないときは、定義してください。 OSPL のスレッド付属イベントを使うときは、定義の変更は不要です。 本関数を呼び出した後の NCGSYS_CreateState 関数呼び出しで生成したイベントフラグが内部で使用するスレッド付属イベントの値を設定します。 R_OSPL_UNUSED_FLAG を指定することができます。 通知先のスレッドは、R_GRAPHICS_Initialize 関数を呼び出したスレッドです。RGA では、graphics_config_t::internal_event_value の値が渡されます。RGA では、NCGSYS_SetNextStateEventValue 関数が呼び出されてから、NCGSYS_CreateState 関数が呼び出される間は、他のスレッドによって割り込まれることはありません。

引　数	NCGuint32 ui32EventValue	使用する OS のスレッド付属イベントの値
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.9 NCGVG_Attach_ISR

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の割込みコールバック関数を登録します。
ヘッダー	ncg_vg_isr.h
宣　言	NCGint32　NCGVG_Attach_ISR(NCGVGISRfp pfnInterrupt);
補　足	OS に応じて内容を変更してください。

引　数	NCGVGISRfp pfnInterrupt	登録する割込みコールバック関数
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.10 NCGVG_Detach_ISR

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の割込みコールバック関数を登録解除します。
ヘッダー	ncg_vg_isr.h
宣　言	NCGint32　NCGVG_Detach_ISR(NCGVGISRfp pfnInterrupt);
補　足	OS に応じて内容を変更してください。

引　数	NCGVGISRfp pfnInterrupt	登録解除する割込みコールバック関数
リターン値	エラーコード、正常=NCG_no_err	

5.8.5.11 NCGVGISRfp

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の割込みコールバック関数の型。
ヘッダー	ncg_defs.h
宣　言	NCGuint32 (*NCGVGISRfp)(void);
補　足	本関数の最後から、R_GRAPHICS_OnInterrupting 関数を呼び出してください。 付属の RGPNCG では、NCGVGISRfp 型の関数は、NCGVG_RGPVG_ISR 関数です。

引　数	なし	
リターン値	なし	

5.8.5.12 NCGVG_Init

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の使用を開始します。	
ヘッダー	ncg_vg.h	
宣　言	NCGvoid　NCGVG_Init(PNCGVGINFO pVGInfo);	
補　足	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の低消費電力モードが停止状態なら本関数の内部で動作状態に変更してください。	
引　数	PNCGVGINFO pVGInfo	使われていません
リターン値	なし	

5.8.5.13 NCGVG_DeInit

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の使用を終了します。	
ヘッダー	ncg_vg.h	
宣　言	NCGvoid　NCGVG_DeInit(PNCGVGINFO pVGInfo);	
補　足	必要なら本関数の内部で OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG)の低消費電力モードを停止状態に変更してください。	
引　数	PNCGVGINFO pVGInfo	使われていません
リターン値	なし	

5.9 割込みハンドラーの登録

割込みベクターに静的に登録してコンパイルするか、本ライブラリの関数を呼び出す前に動的に登録してください。

5.9.1 OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) の割込み

5.9.1.1 NCGVG_RGPVG_ISR

概　要	OpenVG™用ルネサスグラフィックスプロセッサ (R-GPVG) の割込みハンドラー。	
ヘッダー	（なし）	
宣　言	void　NCGVG_RGPVG_ISR(void);	
補　足	SH7269 の OpenVG™用ルネサスグラフィックスプロセッサ（R-GPVG）のすべての割込み INT3 (184), INT2 (185), INT1 (186), INT0 (187)から、本関数が呼ばれるようにしてください。	
引　数	なし	
リターン値	なし	

5.9.2 ビデオディスプレイコントローラ 4 (VDC4) の割込み

「VDC4 Driver User’s Manual」を参考に割込みハンドラーを登録してください。

参考：(5.8.3.1)NCGDU_Attach_ISR

5.9.3 JPEG コーデックユニット (JCU) の割込み

「JPEG Codec Unit Driver User's Manual」を参考に割込みハンドラーを登録してください。

参考：(5.8.3.1)NCGDU_Attach_ISR

5.10 セクション

メモリーマップ セクション情報に下記のセクションを登録してください。 登録しなかったときは、次のようなリンカーエラーが発生します。

```
L1120 (W) Section address is not assigned to "P_RGA"
```

ROM から RAM へマップするセクションに関するリンカーオプションを設定してください。

RAM に配置した各セクションの初期化コードを初期設定プログラムに追加してください。

場所	セクション名
ROM	P_RGA, C_RGA, D_RGA P_RGAH, C_RGAH, D_RGAH P_JCU, C_JCU, D_JCU P_VDC, C_VDC, D_VDC P_OSPL, C_OSPL, D_OSPL
RAM キャッシュ領域	B_RGA, R_RGA（D_RGA が初期値となるセクション） B_RGAH, R_RGAH（D_RGAH が初期値となるセクション） B_JCU, R_JCU（D_JCU が初期値となるセクション） B_VDC, R_VDC（D_VDC が初期値となるセクション） B_OSPL, R_OSPL（D_OSPL が初期値となるセクション）
RAM 非キャッシュ領域	B_RGAH_Work

5.11 補足

5.11.1 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++ API	RGA - C 言語 API	H/W
1 Conformance requirements			
CanvasRenderingContext2D interface.	Canvas2D_ContextClass	graphics_t	－
getContext()	R_RGA_New_Canvas2D_ContextClass	R_GRAPHICS_Initialize	－
context.canvas	-		
CSS currentColor	-		
2 The canvas state			
.save ()	.save()	R_GRAPHICS_Save	－
.restore()	.restore()	R_GRAPHICS_Restore	－
3 Transformations			
.scale(x, y)	.scale(x, y)	R_GRAPHICS_ScaleMatrix	○
.rotate(angle)	.rotate(angle)	R_GRAPHICS_RotateMatrixRadian	○
.translate(x, y)	.translate(x, y)	R_GRAPHICS_TranslateMatrix	○
.transform(a, b, c, d, e, f)	.transform(a, b, c, d, e, f)	R_GRAPHICS_TransformMatrix	○
.setTransform(a, b, c, d, e, f)	.setTransform(a, b, c, d, e, f)	R_GRAPHICS_SetMatrix_2x3	○
4 Line styles			
.lineWidth	-		
.lineCap, .lineJoin, .miterLimit	-		
5 Text styles			
	-		
6 Building paths			
.moveTo()	-		
.closePath()	-		
.lineTo()	-		
.quadraticCurveTo()	-		
.bezierCurveTo()	-		
.arcTo()	-		
.arc()	-		
.rect()	.rect()	R_GRAPHICS_Rect	○
7 Fill and stroke styles			
.fillStyle 単色	.fillStyle	R_GRAPHICS_SetFillColor	○
.fillStyle グラデーション	-		
.fillStyle パターン	.fillStyle	R_GRAPHICS_SetFillPattern	○

.strokeStyle 単色	-		
.strokeStyle グラデーション	-		
.strokeStyle パターン	-		
.createLinearGradient()	-		
.createRadialGradient()	-		
CanvasGradient.addColorStop()	-		
.createPattern()	.createPattern()	R_GRAPHICS_PATTERN_Initialize	―
8 The current default path			
.beginPath()	.beginPath()	R_GRAPHICS_BeginPath	―
.fill()	-		
.stroke()	-		
.drawSystemFocusRing()	-		
.drawCustomFocusRing()	-		
.scrollPathIntoView()	-		
.clip()	.clip() （1 つの長方形のみ）	R_GRAPHICS_Clip （1 つの長方形のみ）	○
.isPointInPath()	-		
9 Drawing rectangles to the canvas			
.clearRect()	.clearRect()	R_GRAPHICS_Clear	○
.fillRect()	.fillRect()	R_GRAPHICS_FillRect	○
.strokeRect()	-		
10 Drawing text to the canvas			
	-		
11 Drawing images to the canvas			
.drawImage(dx, dy)	.drawImage()	R_GRAPHICS_DrawImage	○
.drawImage(dx, dy, dw, dh)	.drawImage()	R_GRAPHICS_DrawImageResized	○
.drawImage(sx, sy, sw, sh, dx, dy, dw, dh)	.drawImage()	R_GRAPHICS_DrawImageChild	○
12 Pixel manipulation			
.createImageData(Width, Height)	.createImageData()	R_GRAPHICS_IMAGE_-InitR8G8B8A8	―
.createImageData(ImageData)	.createImageData()	R_GRAPHICS_IMAGE_-InitSameSizeR8G8B8A8	―
ImageData.width	ImageData.width	R_GRAPHICS_IMAGE_GetProperties	―
ImageData.height	ImageData.height	R_GRAPHICS_IMAGE_GetProperties	―
ImageData.data	ImageData.data	R_GRAPHICS_IMAGE_GetProperties	―

.getImageData()	.getImageData()	R_GRAPHICS_IMAGE_-InitCopyFrameBufferR8G8B8A8	×
.putImageData()	.putImageData()	R_GRAPHICS_DrawImage R_GRAPHICS_DrawImageChild	×
13 Compositing			
.globalAlpha	.globalAlpha	R_GRAPHICS_SetGlobalAlpha	○
.globalCompositeOperation	.globalCompositeOperation	R_GRAPHICS_-SetGlobalCompositeOperation	○
14 Shadows			
	-		

5.11.2 初期化関数の内部の動き

RGA の初期化に失敗するときは、下記のコールツリーとコメントを参考に対処していただくようお願いします。

```
R_GRAPHICS_Initialize
    R_GRAPHICS_STATIC_OnInitializeDefault  or  R_GRAPHICS_STATIC_SetOnInitialize で登録した関数
        // graphics_config_t 構造体のデフォルト設定。 ワークバッファーの設定
    R_OSPL_ToUncachedAddress  // ワークバッファーの非キャッシュ領域のアドレスに変換、または、チェック
    R_OSPL_ToPhysicalAddress  // ワークバッファーの物理アドレスに変換、または、チェック
    NCGSYS_WriteReg （以下、この関数によるレジスターアクセスが何度も呼び出されるが省略）
        // もし、R-GPVG にクロックが供給されていなければ、ここで例外が発生する
        // STBCR8 レジスタの bit4 を 0 に設定すると供給を開始する
    NCGSYS_CreateState
    NCGSYS_SetState( 0x33, NCGSYS_STATE_SET_OR )
        R_OSPL_EVENT_Set
    NCGVG_Attach_ISR
    NCGSYS_SetState( ~0x02, NCGSYS_STATE_SET_AND )
        R_OSPL_EVENT_Clear
    // 割込み RGPVG INT1 (186)。NCGSYS_WaitState と前後する可能性あり
        NCGVG_RGPVG_ISR
            NCGVG_Attach_ISR に指定された関数
                NCGSYS_SetState( 0x02, NCGSYS_STATE_WAIT_OR | NCGSYS_STATE_CALL_INTERRUPT )
                    R_OSPL_EVENT_Set
        // もし割込みが入らなければ、割込みがマスクされているか、
        // ワークバッファーを非キャッシュ領域に配置していないか、R_OSPL_ToUncachedAddress で
        // 非キャッシュ領域に変換していない可能性があります。
    NCGSYS_WaitState( 0x02, NCGSYS_STATE_WAIT_OR )
        R_OSPL_EVENT_Wait  // 割込みから呼ばれる R_OSPL_EVENT_Set を待つ
    NCGSYS_SetState( ~0x02, NCGSYS_STATE_SET_AND )
        R_OSPL_EVENT_Clear
    NCGSYS_WaitState( 0x02, NCGSYS_STATE_WAIT_OR )
        R_OSPL_EVENT_Wait  // 割込みから呼ばれる R_OSPL_EVENT_Set を待つ
    // 割込み RGPVG INT1 (186)。NCGSYS_WaitState と前後する可能性あり
    // ２回目の割込みが入らない場合、SH 版 OS レスでは、R_BSP_InterruptWrite 関数に指定した関数の
    // #pragma interrupt の有無が間違っている可能性があります。 詳しくは、
    // OSPL の R_BSP_InterruptWrite 関数のドキュメントを参照。
        NCGVG_RGPVG_ISR
            NCGVG_Attach_ISR に指定された関数
                NCGSYS_SetState( 0x02, NCGSYS_STATE_WAIT_OR | NCGSYS_STATE_CALL_INTERRUPT )
                    R_OSPL_EVENT_Set
    NCGSYS_GetLastCreatedState
```

5.11.3 画像の形式の識別

(5.7.1.26)R_GRAPHICS_DrawImage、(5.7.1.27)R_GRAPHICS_DrawImageResized、(5.7.1.28)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
Raw	その他	graphics_image_t 型の構造体 ＋ Raw データ

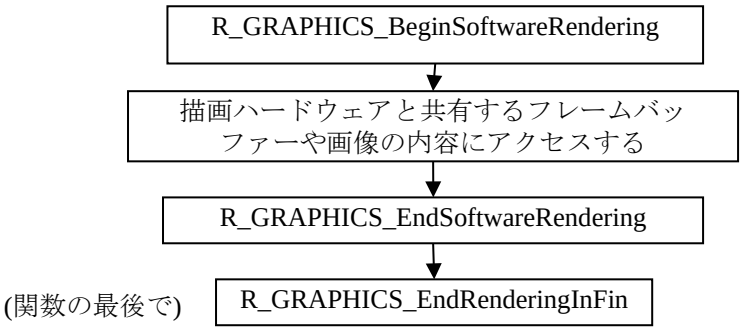
5.11.4 フラッシュモード

自動フラッシュモードのときは、描画した内容を表示するときなどに R_GRAPHICS_Finish または R_WINDOW_SURFACES_SwapBuffers を呼び出すだけで構いませんが、高速手動フラッシュモードのときは、キャッシュのフラッシュ操作が必要になります。

デフォルトは、自動フラッシュモードです。 この場合、本章の内容に注意する必要はありません。

高速手動フラッシュモードにするときは、graphics_config_t::is_fast_manual_flush を true に設定してください。 このモードでは、グラフィックス ライブラリの内部で、不要なフラッシュ操作を行わなくなるため、高速に動作する可能性があります。

フラッシュ操作を効率よく行なうには、描画ハードウェアと共有するフレームバッファーや画像の内容にアクセスする処理を、R_GRAPHICS_BeginSoftwareRendering(5.7.1.45) ～ R_GRAPPHICS_EndSoftwareRendering(5.7.1.46)で囲むようにしてください。



囲む必要がある処理の具体例を示します。

- グラフィックス ライブラリの外から、直接フレームバッファーをリードまたはライトするとき。ただし、フレームバッファーを、非キャッシュ領域からアクセスするときは、囲む必要はありません。
- R_GRAPHICS_DrawImage などに指定する画像データや、R_GRAPHICS_IMAGE_GetAddress から得られた画像データの内容に、リードまたはライトするとき。ただし、ROM に用意された画像データの内容に、リードまたはライトするときは、囲む必要はありません。
- 上記のリードおよびライトが、ファイルをリード ライトする API などの内部で行なわれているとき。

これらの関数を呼び出すと、必要に応じて、フレームバッファーの内容や画像データを、CPU キャッシュから物理メモリーにフラッシュしたり、ハードウェアによる描画を完了させたりします。

もし、これらの関数を使いたくないときは、描画ハードウェアと共有するフレームバッファーや画像の内容にアクセスする前に、R_GRAPHICS_Finish を呼び出して、アクセスした後に、CPU のキャッシュをライトバックしてください。

サンプル

```
  e= R_GRAPHICS_BeginSoftwareRendering( graphics ); IF(e){goto fin;}

  /* Access to frame buffer directly */
  IF ( error ) { e=ERROR_CODE; goto fin; }

  e= R_GRAPHICS_EndSoftwareRendering( graphics ); IF(e){goto fin;}

fin:
  e= R_GRAPHICS_EndRenderingInFin( graphics, e );
```

5.11.5 サンプル画面制御のレイヤー構造

(5.4.5.5)window_surfaces_t がサポートしているレイヤー構造と、ビデオディスプレイコントローラ 4 (VDC4) のプレーンの対応関係を示します。

(5.4.5.7)layer_attributes_t の id	ビデオディスプレイコントローラ 4 (VDC4) のプレーン名
（未使用）	グラフィックス（３）
0	YUV422 以外：グラフィックス（２） YUV422：グラフィックス（１）
-1 （背景色専用）	グラフィックス（１）

5.11.6 フラグド構造体パラメーター

構造体の中の Flags メンバー変数をビット フィールドとして使い、ビットが 1 であれば、対応するメンバー変数を有効にするというコーディング パターンです。ビットが 0 ならば、メンバー変数の値は、省略されたものとして、デフォルトの値が設定されるか設定を変更しません。バージョンアップしたら構造体のメンバーが増えた場合でも、バイナリ互換にできます。

```
FuncA_ConfigClass  config;

config.Flags  = F_FuncA_Param1 | F_FuncA_Param2;
config.Param1 = 10;
config.Param2 = 2;
FuncA( &config );
```

Flags |= F_FuncA_Param3 が無いため、config.Param3 はデフォルト値。

5.11.7 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）

5.11.8 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、destory メンバー関数を呼び出しても例外は発生しません。

5.11.9 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	対象となるオブジェクトは、削除することができます。 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。 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。 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。 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。
リセット状態	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。
元の状態	元の状態に戻ったオブジェクトは、削除できません。 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。

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。

5.11.10 C++言語と JavaScript のオブジェクトの互換性について

Canvas 2D のオブジェクトを使った JavaScript のコードは、本ライブラリが提供する C++言語のクラスを使ってそのまま動くようにすることが可能です。本ライブラリが提供するクラスの変数を使ったコードは、下記のようになり、JavaScript のオブジェクトを参照する変数を使ったコードと同様の記述になります。

- 代入演算をすると、１つのオブジェクトに対して２つの変数からアクセスできるようになります。つまり、C++言語のオブジェクトを指すポインターの値をコピーしたような動きになります。

```
object_1_reference = object_1;
```

- メンバーにアクセスするときは、-> 演算子（ハイフン＋不等号）ではなく、ピリオドを記述してください。

```
object_1.attr = 1;
```

- JavaScript 同様に自動的にデストラクターが起動することはないため、destroy メンバー関数を明示的に呼び出してオブジェクトを削除してください。これは、C++言語の delete 演算子に相当します。

```
object_1.destroy();
```

本ライブラリが提供する C++言語と JavaScript の互換性は、オブジェクトに対するコードのみです。C++言語では変数宣言が必要になるなど、すべての JavaScript のコードがそのまま動くわけではありません。

エラーが発生すると、R_OSPL_GetErrNum 関数の返り値が 0 以外になります。例外は発生しません。エラーから復帰するときは、R_OSPL_CLEAR_ERROR 関数を呼び出してください。エラー状態のときは、メソッドを呼び出しても、内部で何も処理を行ないません。ただし、エラー状態でも処理が必要な save, restore メソッドは、エラー状態でも処理を行います。

エラーがあったかどうかをチェックするコードのサンプルコード：

```
if ( R_OSPL_GetErrNum() != 0 ) { ... }
```

エラーをクリアするコードのサンプルコード：

```
R_OSPL_CLEAR_ERROR();
```

オブジェクトを生成する関数の内部でエラーが発生したら、R_OSPL_GetErrNum 関数の返り値が 0 以外になると同時に、undefined オブジェクトが取得されます。オブジェクトを参照するハンドルの変数と、== 演算子を使って undefined オブジェクトが取得されたかどうかを判定できます。

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。 画像ファイルをまとめることに関する設定は、XML ファイルに記述します。XML ファイルに記述するファイル名および拡張子は大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。内部で必要な 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. armcc¥common¥src¥samples¥RGA 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) 2012-2016 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) 2012-2016 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  armcc¥common¥src¥samples¥RGA¥Sample_Common¥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_SH7269.c" language="C"
symbol="g_RGA_Sample_BinaryImage"
  source_template="${ImagePackagerLib}¥SourceTemplate.xml#default"
  super_class="${ImagePackagerLib}¥SuperClass.xml#default">
<OutputHeader path="BinaryImage_SH7269.h"
include_define="BINARYIMAGE_SH7269_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：

<table>
<tr><th>オフセット</th><th>サイズ</th><th>説明</th></tr>
<tr><td>0x00</td><td>4byte</td><td>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。</td></tr>
<tr><td>0x04</td><td>4byte</td><td>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。</td></tr>
<tr><td>0x08</td><td>4byte</td><td>（予約領域）</td></tr>
<tr><td>0x0C</td><td>2byte</td><td>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。</td></tr>
<tr><td>0x0E</td><td>2byte</td><td>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。</td></tr>
<tr><td>0x10</td><td>1byte</td><td>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</td></tr>
<tr><td>0x11</td><td>7byte</td><td>（予約領域）</td></tr>
<tr><td>*</td><td></td><td>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、@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は、先頭アドレスと１行あたりのバイト数が、下記の値でアラインメントされている必要があります。
<table>
<tr><th colspan="3">画像データの先頭アドレスのアラインメント（バイト）</th><th colspan="3">１行あたりのバイト数のアラインメント（バイト）</th></tr>
<tr><td>RGB</td><td>YCbCr</td><td>CLUT</td><td>RGB</td><td>YCbCr</td><td>CLUT</td></tr>
<tr><td>32</td><td>4</td><td>1</td><td>32</td><td>4</td><td>1</td></tr>
</table>
</td></tr>
</table>

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"。
@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" … C_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 32` 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。 デフォルトは、@super_class が指す先の @raw_stride_alignment です。 例： `raw_stride_alignment="32"` [設定範囲] 1 以上

@raw_stride_alignment_4	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。 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。 例： `raw_stride_alignment_4="YUV422"`
@alpha_raw_image_width	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。 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。
@table_format	テーブルのフォーマット。 オプション。 ● "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も埋め込みます。 ● "Enbed"：画像やファイルをそのまま順次埋めていきます。 デフォルトは、次の通り。 ● @language="Binary" なら @table_format="Offset" ● @language="C" なら @table_format="Embed"
@source_template	@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。 /ImagePackager/SourceTemplate/@id の値を指定します。 他の XML ファイルの値を参照することもできます。 ${ImagePackagerLib} を指定すると、ImagePackagerLib.vbs があるフォルダーのパスに置き換わります。 例： `source_template="${ImagePackagerLib}¥SourceTemplate.xml#default"`
@super_class	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。 /ImagePackager/SuperClass/@id の値を指定します。 他の XML ファイルの値を参照することもできます。 ${ImagePackagerLib} を指定すると、ImagePackagerLib.vbs があるフォルダーのパスに置き換わります。 例： `super_class="${ImagePackagerLib}¥SuperClass.xml#default"`

/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。

@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 フォルダーのパスや、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できます。
@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。 graphics_image_t* を指定してください
@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

出力する画像ファイルのフォーマット。 必須。
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。
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。
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。

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）

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。

@premultiplied_alpha

RGB 成分を Alpha 成分で乗算済みにするかどうか。 オプション

- "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

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。

@type

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[]。

@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。 uint8_t[] 型
@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
@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。 整数か下記の特殊形が指定できます。 ● 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 file_path には、画像ファイルのパスを指定してください。

/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。 ● ${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） ● ${BinaryData} : バイナリデータ
SourceWithSection/text()	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。 SourceWithSection タグが省略されたときは、セクション指定があるときでも Source タグの内容が使用されます。
Header/text()	ヘッダー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。

- ${include_define} : 二重インクルード防止のマクロ名。/ImagePackager/OutputBinary/OutputHeader/@include_define の値
- ${DeclareBinaryImageSymbol} : バイナリデータのシンボルの宣言。DeclareVariable/text() または DeclareAddress/text() の内容と、@raw_image_alignment_symbol などによる #define 定義。
- ${Variables} : 変数に相当する #define の一覧
- ${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）
- ${StartAddress} : バイナリの先頭アドレス
- ${LastAddress} : バイナリの最終アドレス

HeaderWithSection/text() 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。
HeaderWithSection タグが省略されたときは、セクション指定があるときでも Header タグの内容が使用されます。

DeclareVariable/text() @language="C" のときの ${DeclareBinaryImageSymbol} に入れるテンプレート。
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。

- ${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）

DeclareAddress/text() @language="Binary" のときの ${DeclareBinaryImageSymbol} に入れるテンプレート。
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。

- ${Section} : セクション名。/ImagePackager/OutputBinary/@section の値
- ${Symbol} : 変数名。/ImagePackager/OutputBinary/@symbol の値
- ${StartAddress} : バイナリの先頭アドレス
- ${LastAddress} : 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。

6.1.8.1 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ないときは、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。 仕様に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>
```

6.1.8.2 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、下記のようになります。 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、下記のようになります。 タグ名と属性名の間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 / ではないときは、相対パスになります。

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

6.1.8.3 # 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 エラー情報検索 SearchErrorInformation

デバッグ版のライブラリの中でエラーが発生すると、R_DEBUG_BREAK_IF_ERROR 関数の中から、エラーが発生した場所が表示されます。SH7269 版では、シリアル出力に下記のように出力されます。もし、シリアルが使えなくても LED などを使ってエラーが発生した場所の情報を得る方法もあります（後記）。

出力：

```
<ERROR error_ID="1" file="C:¥folder¥Library.c(2095)"/>
```

デバッグ版に切り替えるには、リンクするライブラリを lib¥SH7269¥Release¥RGA.lib から lib¥SH7629¥Debug¥RGA.lib に変更し、R_OSPL_NDEBUG の#define 定義を非定義にしてください。

RGA_Tools.vbs をダブルクリックして、SearchErrorInformation コマンドを実行すると、ライブラリの中のファイル名と行番号から、エラーコードよりも詳細なエラーの情報を得ることができます。ただし、ライブラリになっていないプロジェクトに入っているソースファイルについては、検索データベースに入っていません。

RGA_Tools.vbs をダブルクリックした後のウィンドウ：

```
RGA Tools - Copyright(c) 2012-2016 Renesas Electronics Corporation
 1. 画像フォーマット変換 [RunImagePackager]
 2. エラー情報検索 [SearchErrorInformation]
番号またはコマンド >2
------------------------------------------------------------------
  ((( [SearchErrorInformation_sth] )))
エラーが発生したソースファイルのパスまたはファイル名 >Library.c(2095)
SoftFillRectangle_YUV422() : Rectangle.Left must be even
SoftFillRectangle_YUV422() : Rectangle.Left は偶数であること
```

エラーが発生した位置でブレークさせるときは、R_OSPL_SET_BREAK_ERROR_ID 関数をプログラムの最初で呼び出してください。引数は、上記 ERROR タグの error_ID 属性の値を指定してください。

```
R_OSPL_SET_BREAK_ERROR_ID( 1 );
```

SH7269 では、RGA¥Sample_SH7269¥src¥driver¥ospl¥porting¥DebugBreak.c を HEW にドラッグ＆ドロップして開き、R_DebugBreak 関数の先頭の行の左にある E(Event)の列をダブルクリックして、ハードウェア ブレークポイントを張ってください。

プログラムを再起動すると、エラーが発生した場所でブレークします。HEW では、[表示 > コード > スタックトレース] を選ぶと呼び出し元の関数が分かります。ただし、淡色表示になって選べないときは、すでにどこかに表示されています。

もし、シリアルが使えなくても LED、または GPIO とオシロスコープが使えれば、シリアル出力される情報と同等の情報を得ることができます。その方法は、R_DebugBreak 関数の中から LED を制御する関数と一定時間待つ関数（R_OSPL_Delay など）を呼び出し、モールス信号のように R_DebugBreak 関数の引数を表示することです。たとえば、次のように表示します。

- 開始は、１秒点灯、１秒消灯
- １なら 0.5 秒点灯 0.5 秒消灯の表示
- ０なら 0.2 秒点灯 0.8 秒消灯の表示
- シフト演算で次の桁の１／０を表示していき、２進数を表示する

6.3 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。 データのみのバイナリでは、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) 2012-2016 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 >
```

6.4 画像ファイル作成 RawToBmp

フレームバッファーにある 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) 2012-2016 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	フレームバッファーにあった 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 1.2。
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. ソフトウェア改訂内容

本章では主な改訂内容を示します。すべての改訂については、改訂記録を参照してください。

7.1 電源投入直後から描画できないことがある現象を改善

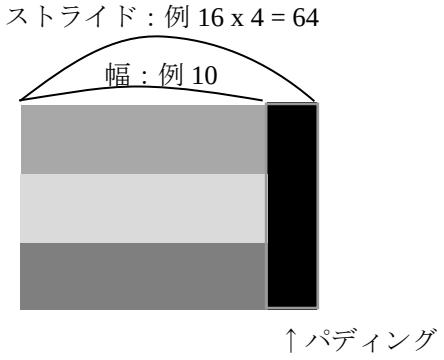
RGA 2.10 以前をご使用の場合、チップの個体によっては、電源投入直後から描画できないことがある現象がありました。 RGA 2.12 ではその現象を改善しています。

7.2 拡大して画像を描画する際の制限事項

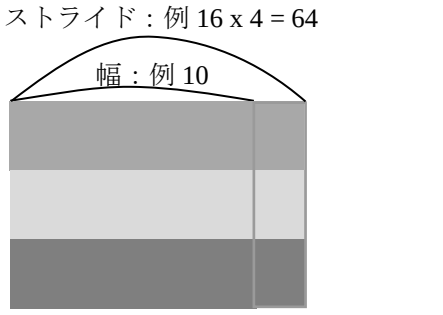
下記の条件では使用しないでください。

下記の 4 条件が同時に成立するとき：

- graphics_quality_flags_t に GRAPHICS_IMAGE_QUALITY_ANTIALIASED を設定
- 参照元画像のバイト幅（i.e. 横幅 Pixel 数 x 色深度）≠ 32 の倍数。
  例えば、幅:100、カラーフォーマット:ARGB4444 の場合、幅（Byte）は 100 x 2 = 200 Byte
  → 32 の倍数ではないので本条件が成立
- 参照元画像を拡大して描画
  例えば、1 x 128 の参照元画像を 256 x 128 のサイズに拡大して描画する場合、本条件が成立
- パディング部分が、それぞれの行の右端の色の値で埋められていない



それぞれの行の右端の色の
値で埋められていない



それぞれの行の右端の色の
値で埋められている

7.3 透明なピクセルを持つ画像を描画する際の制限事項

下記の条件では使用しないでください。

以下のいずれかの RGB のフォーマットの参照元（ソース）画像を、

- ARGB8888 かつ A=0
- ARGB1555 かつ A=0
- ARGB4444 かつ A=0

以下のいずれかの RGB フォーマットの描画先（デスティネーション）へ描画する。

- ARGB1555
- ARGB4444

- RGB565

参照元、描画先のフォーマットのどの組み合わせでも使用しないでください。

7.4 完全不透過のピクセルを持つ画像を描画する際の制限事項

下記の条件では使用しないでください。

以下のいずれかの RGB のフォーマットの参照元（ソース）画像を、

- XRGB8888
- ARGB8888 かつ A=255
- ARGB1555 かつ A=1
- ARGB4444 かつ A=15
- RGB565

以下のいずれかの RGB フォーマットの描画先（デスティネーション）へ描画する。

- ARGB1555
- ARGB4444
- RGB565

参照元、描画先のフォーマットのどの組み合わせでも使用しないでください。

8. サンプルコード

サンプルコードは、RGA フォルダーに入っています。

SH7269 で動作させるときは、RGA¥Sample_SH7269 に入っている HEW のプロジェクト ファイルが使えます。

PC で動作させるときは、RGA¥Sample_PC に入っている Visual Studio 2008 のプロジェクト ファイルが使えます。

操作手順は、別紙「RGA チュートリアル」を参照してください。

9. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

SH7268/SH7269 グループユーザーズマニュアル ハードウェア編 Rev.1.00
（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。）

ユーザーズマニュアル：開発環境

SuperH™ RISC engine C/C++コンパイラー、アセンブラ、最適化リンケージエディタコンパイラパッケージ V.9.04 ユーザーズマニュアル Rev.1.01
（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。）

HTML Canvas 2D Context

W3C Candidate Recommendation 29 March 2012

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://japan.renesas.com

お問合せ先

http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容
2.12	2018.01.26	● 修正：5.11.2 章：以下の関数名の誤記修正 誤：NCGSYS_SetStateEventValue 正：NCGSYS_GetLastCreatedState ● 新規：7 章：「ソフトウェア改定改訂内容」の章を新規作成 以下は、コードの変更内容 ● 同梱している JCU のバージョンを更新：1.03→1.04 ● 修正：チップの個体によっては、電源投入直後から描画できないことがある現象を改善（7.1 章） ● 修正：描画が停止する現象を修正 修正内容は、移植層 RGPNCG に割込み禁止による排他制御を追加
2.10	2016.02.29	● 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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理

   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

   【注意】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. リザーブアドレスのアクセス禁止

   【注意】リザーブアドレスのアクセスを禁止します。

   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。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。

4. クロックについて

   【注意】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。

5. 製品間の相違について

   【注意】型名の異なる製品に変更する場合は、事前に問題ないことをご確認下さい。

   同じグループのマイコンでも型名が違うと、内部メモリー、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

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みます。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)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/