3DS

3DS プログラミングマニュアル

FONT ライブラリ編

2015-11-05

Version 1.1

目次

コード

表

図

1. はじめに

本ドキュメントは、CTR-SDK に付属する FONT ライブラリを利用して、文字の描画をアプリケーションで行うためのプログラミング手順などについて説明したものです。

「2. 用語」では、FONT ライブラリ独自の用語を説明しています。本ドキュメントに記述されている用語について知ることができます。

「3. フォントの管理」では、フォントリソースを管理するクラスを説明しています。FONT ライブラリにおいて、フォントのリソースがどのように管理されているのかを知ることができます。

「4. 文字の表示」では、文字を表示するための方法を説明しています。FONT ライブラリを利用して文字を描画するための実装方法を知ることができます。

「5. フォントの作成」では、ctr_FontConverter を利用してフォントリソースを作成する方法を説明しています。アプリケーションで使用するフォントリソースを作成したい場合に参照してください。

2. 用語

FONT ライブラリで使用する用語を定義します。ここでの定義は独自のものであり、一般的な意味と異なることがあります。

2.1. 文字に関連する用語

文字に関連する用語をまとめたものを下表に示します。

表 2-1. 文字に関連する用語

用語	説明
文字	描画の対象となる図形。視覚的に認識できる図形に加え、その周囲の空白領域も含みます。
文字コード	文字をコンピュータで扱うために、それぞれの文字に割り当てられる数値。
文字セット	文字と文字コードのペアの集合。
文字列エンコーディング	文字コードを変換・配置して文字列をバイト列として表すための方法。
ISO 8859-1	ASCII 文字にいくつかの欧州文字や記号を追加した文字セットで、Latin-1 とも呼ばれています。 使われる文字コードの範囲は 0x00～0xFF ですが、0x00～0x1F と 0x80～0x9F には、表示することのできない制御文字が割り当てられています。
CP1252（Code Page 1252）	Windows のために Microsoft 社によって定義された欧州文字セット。ISO 8859-1 のうち、制御文字に割り当てられている 0x80～0x9F の範囲を表示可能な文字に置き換えたものですので、ISO 8859-1 で表示可能な文字はすべて CP1252 で表示することができます。
JIS X 0201	JIS 規格で定められている日本語文字セット。いわゆる全角文字で、ひらがな・カタカナ・漢字等を含んでいます。
JIS X 0208	JIS 規格で定められている日本語文字セット。いわゆる半角文字で、ASCII と半角カタカナを含んでいます。
ShiftJIS	日本語環境での標準的な文字列エンコーディング。JIS X 0201 と JIS X 0208 に含まれている文字を使用することができます。
Unicode	全世界の文字を、単一の文字セットで表現することを目的とした文字体系。文字セットおよび文字列エンコーディング双方の定義を含んでいます。
UTF16	Unicode で定義されている文字列エンコーディング。すべての文字が 2 バイトで表現されているため、ほかの文字列エンコーディングのほとんどが持っている、ASCII との互換性を持っていません。 エンディアンの違いで、UTF16-BE と UTF16-LE のように表記することがあります。
UTF8	Unicode で定義されている文字列エンコーディング。文字によって 1 文字に対応するバイト数が異なります。ASCII 文字が 1 バイトで表現されるため、UTF16 とは異なり、ASCII との互換性を持っています。

2.2. グリフに関連する用語

視覚的に認識できる文字の形を「グリフ」と定義し、この節では、そのグリフの表示に関する用語を説明します。

下図は、グリフの表示に関連するパラメータを示したものです。黄色い領域がグリフ、グリフと緑色の領域を合わせたものが文字です。黒字は FONT ライブラリが提供するグリフが持っているパラメータ、青字は FONT ライブラリが提供するフォントが持っているパラメータ、赤字はグリフを文字列として表示する際にアプリケーション側で設定するパラメータです。

図 2-1. グリフ

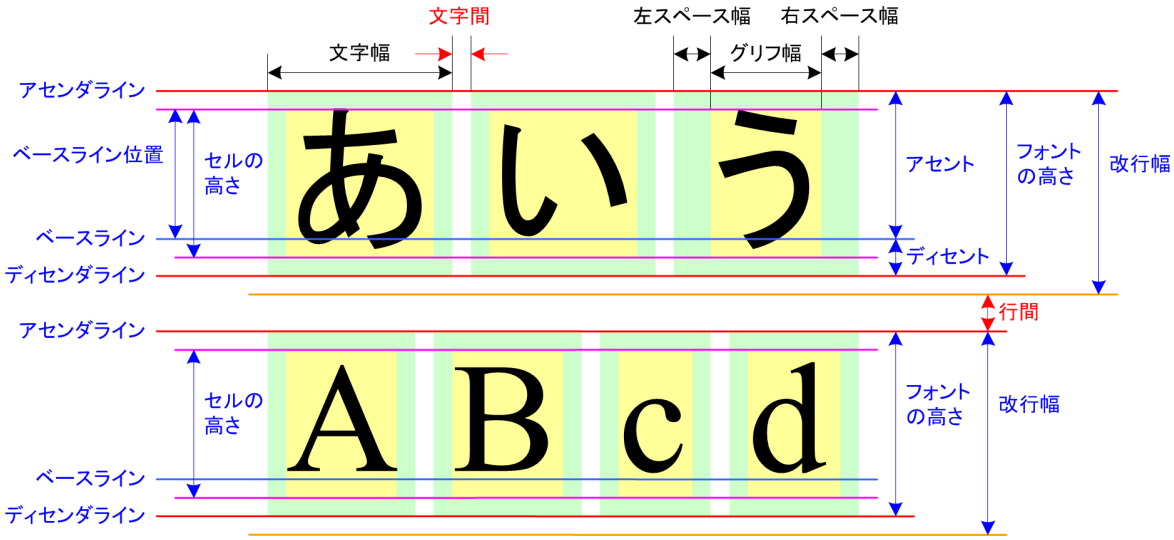


表 2-2. グリフに関連する用語

用語	説明
グリフイメージ	画像としての面を特に強調して、グリフを指す言葉。セルと呼ばれることもあります。
セルの高さ	グリフイメージの高さ。フォント全体で共通です。
グリフ幅	グリフに外接する矩形の幅。図中では黄色い領域の幅です。
左スペース幅	グリフと直前の文字との間に空ける空白の幅。
右スペース幅	グリフと直後の文字との間に空ける空白の幅。
文字幅	文字の幅。グリフ幅と左右のスペース幅を足し合わせたものです。
文字幅情報	文字の幅に関する情報。左スペース幅、右スペース幅、グリフ幅、文字幅の 4 つの情報をまとめたものです。
ベースライン	文字を横に並べて配置する(=文字列を描画する)際に、各グリフの上下方向の配置の基準となる位置。
アセンダライン	文字列が占めるべき領域の上端となるライン。
ディセンダライン	文字列が占めるべき領域の下端となるライン。
アセント	ベースラインとアセンダラインの距離。
ディセント	ベースラインとディセンダラインの距離。
フォントの高さ	アセンダラインとディセンダラインの距離。
フォントの幅	タブ幅の基準となる、文字によらない文字幅。

2.3. フォントのリソースに関する用語

フォントデータなど、フォントのリソースに関連する用語をまとめたものを下表に示します。

表 2-3. フォントのリソースに関連する用語

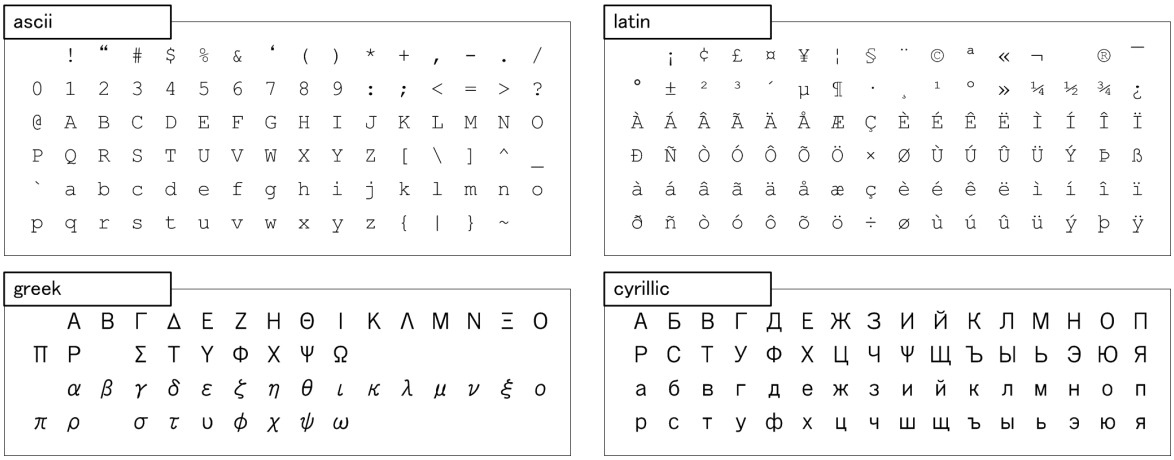
用語	説明
bcfnt (Binary Ctr FoNT)	FONT ライブラリで使用する通常のフォントデータ。また、そのフォントデータを格納したファイルの拡張子。
bcfna (Binary Ctr FoNt Archived)	FONT ライブラリで使用する圧縮されたフォントデータ。また、そのフォントデータを格納したファイルの拡張子。 グリフの集合単位で抽出・展開して使用することができます。表示するフォントを言語ごとにグループで分ける場合や、一部のフォントだけ書体を切り替えたい場合などに利用します。
フォントリソース	FONT ライブラリで使用する通常のフォントデータ。bcfnt と同義です。
アーカイブフォント	FONT ライブラリで使用する圧縮されたフォントデータ。bcfna と同義です。
ctr_FontConverter	bcfnt および bcfna を作成するための Windows ツール。GUI 版と CUI 版が用意されています。
代替文字	フォントに含まれていない文字の情報が要求された場合に、代わりに使用される文字。

2.3.1. グリフグループ

グループ分けされたグリフの集合を「グリフグループ」と定義します。また、グリフグループの集合を「グループセット」と呼びます。

FONT ライブラリでは、グループセット内の任意のグループを組み合わせて、フォントを構築する機能を提供しています。例えば「ascii」、「latin」、「greek」、「cyrillic」の 4 つのグループからなるグループセットから、ラテン語圏向けには「ascii」と「latin」の組み合わせ、ロシア語圏向けには「ascii」と「cyrillic」の組み合わせというように、表示に必要なグリフが含まれるグループだけでフォントを構築することができます。

図 2-2. グリフグループの例

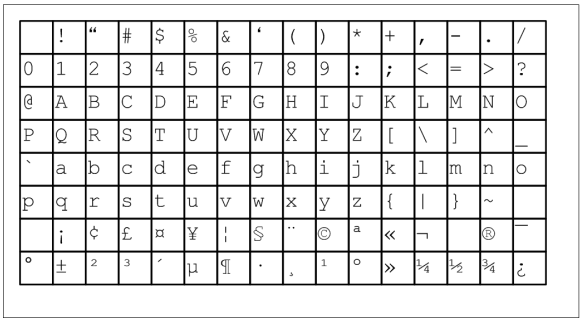


2.3.2. シート

グリフイメージが描かれているテクスチャを「シート」と定義します。テクスチャ 1 枚がシート 1 枚に対応し、1 枚のシートには、複数のグリフイメージが並べられています。

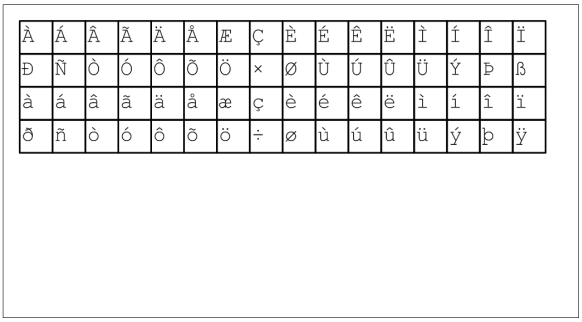
図 2-3. シートの例

シート 1



	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿

シート 2



À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

2.4. その他の用語

そのほかの、FONT ライブラリに関連する用語をまとめたものを下表に示します。

表 2-4. その他の用語

用語	説明
タグ文字列	描画に関する命令を組み込んだ文字列。 FONT ライブラリではタグ文字列を処理する枠組みを提供します。文字列に埋め込まれたタグに対する実際の処理はアプリケーションで用意しなければなりません。

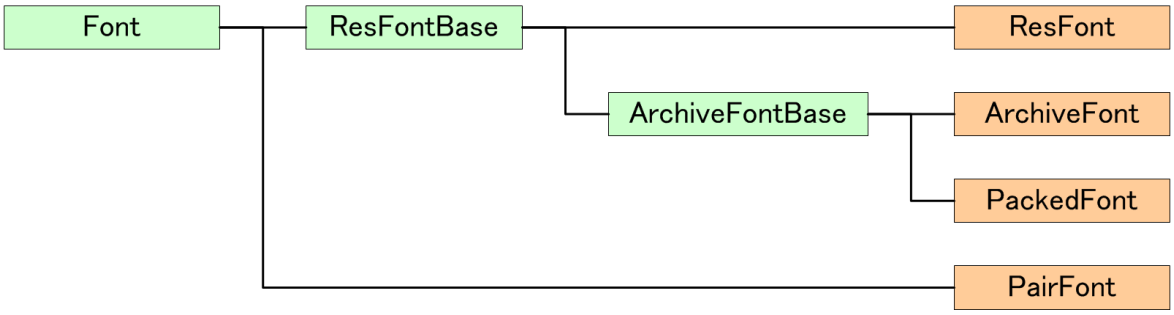
3. フォントの管理

この章ではフォントの管理について説明し、アプリケーションでの文字描画については以降の章で説明します。

FONT ライブラリでは、基礎となる Font クラスをベースに、メモリ上にロードされたフォントリソースを操作する ResFont クラス、圧縮されているフォントリソースを操作する ArchiveFont クラスと PackedFont クラス、2 つのフォントを 1 つのフォントとして扱う PairFont クラスを用意しています。なお、フォントリソースの文字コードには、CTR-SDK のライブラリが文字列を扱う際に使用している、UTF16-LE を使用することを推奨します。

下図は、FONT ライブラリのクラスの階層図(左側が継承元)を示したものです。アプリケーションでは、図中で右端にあるクラスの中から、フォントリソースの利用形態によってインスタンスを作成するクラスを選択します。

図 3-1. クラス階層図(フォントの管理部分)



3.1. Font クラス

フォントリソースを扱うクラスのベースとなるクラスですが、このクラス自身のインスタンスを作成することはありません。文字の描画に FONT ライブラリのクラスを利用する場合、テクスチャ(グリフイメージが描画されているシート)のロードがメインメモリを直接参照する設定で行われるため、アプリケーションはデバイスメモリ上にフォント(イメージ)用のバッファを用意しなければなりません。このメモリ配置の制限については、このクラスを継承するそれぞれのクラスで説明します。

下表は、Font クラスで定義されているメンバ関数の一覧です。パラメータを取得する場合は、基本的にピクセル単位の値を返すことに注意してください。

表 3-1. Font クラスのメンバ関数一覧

カテゴリ	関数	説明
フォント情報取得	`GetWidth()`	フォントの幅を返します。タブ幅の解決などに使用します。
	`GetHeight()`	フォントの高さを返します。
	`GetAscent()`	アセントを返します。文字の上端をそろえるために使用します。
	`GetDescent()`	ディセントを返します。文字の下端をそろえるために使用します。
	`GetMaxCharWidth()`	含まれている文字の中で最大の文字幅を返します。
	`GetType()`	フォントの種別を返します。

	GetTextureFormat()	シートの(テクスチャ)フォーマットを返します。
	GetLineFeed()	改行幅を返します。
	GetDefaultCharWidths()	デフォルトの文字幅情報を返します。フォントにない文字の文字幅などに使用します。
フォント情報設定	SetLineFeed()	改行幅を再設定します。
	SetDefaultCharWidths()	デフォルトの文字幅情報を再設定します。
	SetAlternateChar()	代替文字を設定します。
文字情報取得	GetCharWidth()	文字の文字幅を返します。
	GetCharWidths()	文字の文字幅情報を返します。
	GetGlyph()	グリフの情報。指定された文字のグリフがフォントに含まれていなければ、代替文字のグリフを返します。
	HasGlyph()	指定された文字のグリフがフォントに含まれているかどうかを返します。
文字列 エンコーディング	GetCharacterCode()	文字コードを返します。
	GetCharStrmReader()	FONT ライブラリで文字を描画する際に、CharWriter クラスが利用します。
シート情報取得	GetBaselinePos()	ベースラインを返します。
	GetCellHeight()	セルの高さを返します。
	GetCellWidth()	セルの幅を返します。
テクスチャ補間	EnableLinearFilter()	FONT ライブラリで文字を描画する際、拡大縮小時に線形補間を行うかどうかを設定します。
	IsLinearFilterEnableAtSmall()	FONT ライブラリで文字を描画する際、縮小時に線形補間を行うかどうかを返します。
	IsLinearFilterEnableAtLarge()	FONT ライブラリで文字を描画する際、拡大時に線形補間を行うかどうかを返します。
	GetTextureWrapFilterValue()	FONT ライブラリで文字を描画する際の、ラッピングモードとフィルタ設定を返します。

3.2. ResFont クラス

フォントリソースを操作するためのクラスです。グループ化には対応していませんので、すべてのシートがメモリにロードされます。そのため、フォントリソース全体がメモリ上にロードされていなければなりません。

メモリ上にロードされたフォントリソース(bcfnt ファイル)は SetResource() で ResFont クラスに関連付けます。文字の描画に FONT ライブラリのクラスを利用する場合、フォントリソースに含まれているシートと同じ数のテクスチャオブジェクトが glGenTextures() で作成されます。作成されたテクスチャオブジェクトはテクスチャイメージに GPU が直接アクセスする設定のため、フォントリソースはデバイスメモリ上に 128 Byte のアライメント(nn::font::GlyphDataAlignment で定義)で読み込まなければなりません。関連付けの解除は RemoveResource() で行います。関連付けを解除するまで、フォントリソースを読み込んだバッファは解放しないでください。

文字の描画に FONT ライブラリのクラスを利用する場合、GetDrawBufferSize() で取得したサイズの描画用バッファを SetDrawBuffer() で設定する必要がありますが、描画用バッファはデバイスメモリ上に確保する必要はありません。描画用バッファは nn::font::ResFont::BUFFER_ALIGNMENT (4 Byte) でアライメントする必要があります。

3.2.1. 内蔵フォント

3DS では、本体保存メモリ内に内蔵フォントと呼ばれるフォントリソースが格納されています。アプリケーションは PL ライブラリを使用して、この内蔵フォントを共有のフォントリソースとして利用することができます。

nn::pl::InitializeSharedFont() の呼び出しで 3DS 本体のリージョンに対応した内蔵フォントのロードが開始され、nn::pl::GetSharedFontLoadState() が nn::pl::SHARED_FONT_LOAD_STATE_LOADED を返した時点でロードが完了します。この手順でロードされたフォントリソースは、固定アドレス(アプリケーション管轄外の物理アドレス)にロードされるため、アプリケーションのメモリを消費しません。

3DS 本体のリージョンに対応してロードされた内蔵フォントの種類は、nn::pl::GetSharedFontType() で取得することができます。この関数で取得した種類ではない内蔵フォントを使用する場合は、nn::pl::MountSharedFont() で内蔵フォントのアーカイブをマウントし、フォントリソースのファイルをアプリケーションでロードしなければなりません。ファイルは LZ77 形式で圧縮されていますので、CX ライブラリを利用して解凍してください。また、内蔵フォントの種類ごとにマウントする必要があることと、FONT ライブラリで文字を描画する場合は解凍したフォントリソースをデバイスメモリ上に置かなければならないことに注意してください。フォントリソースのロードが完了したときは、nn::pl::UnmountSharedFont() でアーカイブをアンマウントしてください。

> **補足:** 内蔵フォントのロードについては、「3DS プログラミングマニュアル – システム編」を参照してください。

3.3. ArchiveFont クラス

アーカイブフォントを操作するためのクラスです。フォントファイル内のグリフはグループで分けられており、指定したグループ(複数可)のグリフが含まれているシートが解凍されてメモリにロードされます。グループの指定には、すべてのグリフをロードする LOAD_GLYPH_ALL か、ロードするグループ名を ″,″ 区切りでつなげた文字列を用います。

フォントの構築方法には、すでにメモリ上にロードされているアーカイブフォントのデータから構築する方法と、データを順次ロードすることでロード時に必要なメモリを節約する方法とがあります。前者は Construct()、後者は InitStreamingConstruct() と StreamingConsturuct() を使用します。フォントのデータは圧縮されていますが、フォントイメージは解凍されてメモリ上に展開されます。フォントの構築に必要なバッファ(展開先のメモリ)のサイズは GetRequireBufferSize() で取得します。文字の描画に FONT ライブラリのクラスを利用する場合は、バッファをデバイスメモリ上に 128 Byte アライメントで確保する必要があります。

フォントの構築後は、アーカイブフォントのロードに使用したメモリを解放することができますが、フォントを破棄するまで、フォントの構築に使用したバッファは解放することができません。フォントの破棄は Destroy() で行います。その際、解放すべきバッファのポインタが返されます。

3.4. PackedFont クラス

ArchiveFont クラスと同様に、アーカイブフォントを操作するためのクラスですが、圧縮されたままのフォントリソースをメモリに持ち、必要なときに解凍して使用することができます。文字に対応するシートを解凍する処理のために負荷はかかりますが、キャッシュが実装されているため、ある程度の負荷は軽減されます。また、キャッシュに残す割合を調整することや、シートをロックすることでキャッシュ落ちを防ぐことができます。

フォントの構築方法は ArchiveFont クラスと同じです。バッファのサイズを計算する際に、キャッシュに残すシートの数や割合を指定します。キャッシュを管理する関数としては、キャッシュ可能なシート数を取得する GetNumCache()、あらかじめシートをキャッシュに読み込む PreloadSheet()、シートをロックする LockSheet()、シート単位でロックを解除する UnlockSheet()、すべてのシートのロックを解除する UnlockSheetAll() が用意されています。

フォントの構築後は、アーカイブフォントのロードに使用したメモリを解放することができますが、フォントを破棄するまで、フォントの構築に使用したバッファは解放することができません。フォントの破棄は Destroy() で行います。その際、解放すべきバッファのポインタが返されます。

3.5. PairFont クラス

Font クラスを継承したクラスで、同じ文字コード、同じテクスチャフォーマットの 2 つのフォントを 1 つのフォントとして利用するためのクラスです。ベースラインの位置が異なるフォントを組み合わせた場合、文字の表示位置に問題が発生する可能性があることに注意してください。

PairFont クラスのインスタンス構築時に指定するフォントの順番で、先に指定したものをプライマリフォント、次に指定したものをセカンダリフォントと呼びます。このクラスでパラメータを取得すると、下表の条件に従って、プライマリフォントかセカンダリフォントのパラメータを取得することになります。

表 3-2. PairFont クラスのパラメータ

パラメータ	パラメータ取得の条件
フォントの幅	両者のうちで最大の値。
フォントの高さ	
最大文字幅	
アセント	フォントの高さの値が大きいフォントの値。 フォントの高さが同じならばプライマリフォントの値を取得します。
ディセント	
ベースライン	
セルの高さ	
改行幅	
セルの幅	フォントの幅の値が大きいフォントの値。 フォントの幅が同じならばプライマリフォントの値を取得します。
デフォルト文字幅	
グリフの取得	プライマリフォント優先。 プライマリフォントになければセカンダリフォントから取得し、指定の文字がどちらにもなければプライマリフォントの代替文字を取得します。ただし、インスタンス構築後に代替文字を設定したときは、その文字がプライマリフォントになく、セカンダリフォントにあった場合はセカンダリフォントの代替文字を取得します。
代替文字	プライマリフォント優先。 ただし、インスタンス構築後に代替文字を設定したときは、その文字がプライマリフォントになく、セカンダリフォントにあった場合はセカンダリフォントの代替文字を取得します。

4. 文字の表示

文字を表示するには、フォントリソースから取得したグリフの情報をもとに表示する方法と、FONT ライブラリで用意されている文字描画クラスを利用する方法とがあります。

4.1. グリフの情報をもとに表示する

Font クラスの GetGlyph() は、グリフの情報を Glyph 構造体で返します。この構造体には、グリフを文字として表示するために必要な情報が格納されており、以下のように定義されています。

コード 4-1. Glyph 構造体

```
struct Glyph
{
    const void* pTexture;
    struct CharWidths
    {
        s8  left;
        u8  glyphWidth;
        s8  charWidth;
    } widths;
    u8      height;
    u16     texWidth;
    u16     texHeight;
    u16     cellX;
    u16     cellY;
    u8      isSheetUpdated;
    TexFmt  texFormat;
    const TextureObject* pTextureObject;
};
```

各メンバに格納されている値は以下のとおりです。幅や高さの値はピクセル単位で格納されています。

表 4-1. Glyph 構造体のメンバ

メンバ	説明
pTexture	シート(テクスチャイメージ)へのポインタ。
widths.left	左スペース幅。
widths.glyphWidth	グリフ幅。
widths.charWidth	文字幅。 右スペース幅は(文字幅 – グリフ幅 – 左スペース幅)で求めることができます。
height	セル(グリフイメージ)の高さ。
texWidth	シートの幅。
texHeight	シートの高さ。
cellX	シートの左上隅を原点としたときの、セル左上隅の X 座標。
cellY	シートの左上隅を原点としたときの、セル左上隅の Y 座標。

isSheetUpdated	シート更新フラグ。FONT ライブラリが参照します。
texFormat	シートの(テクスチャ)フォーマット。
pTextureObject	テクスチャオブジェクトなどの情報。FONT ライブラリが参照します。

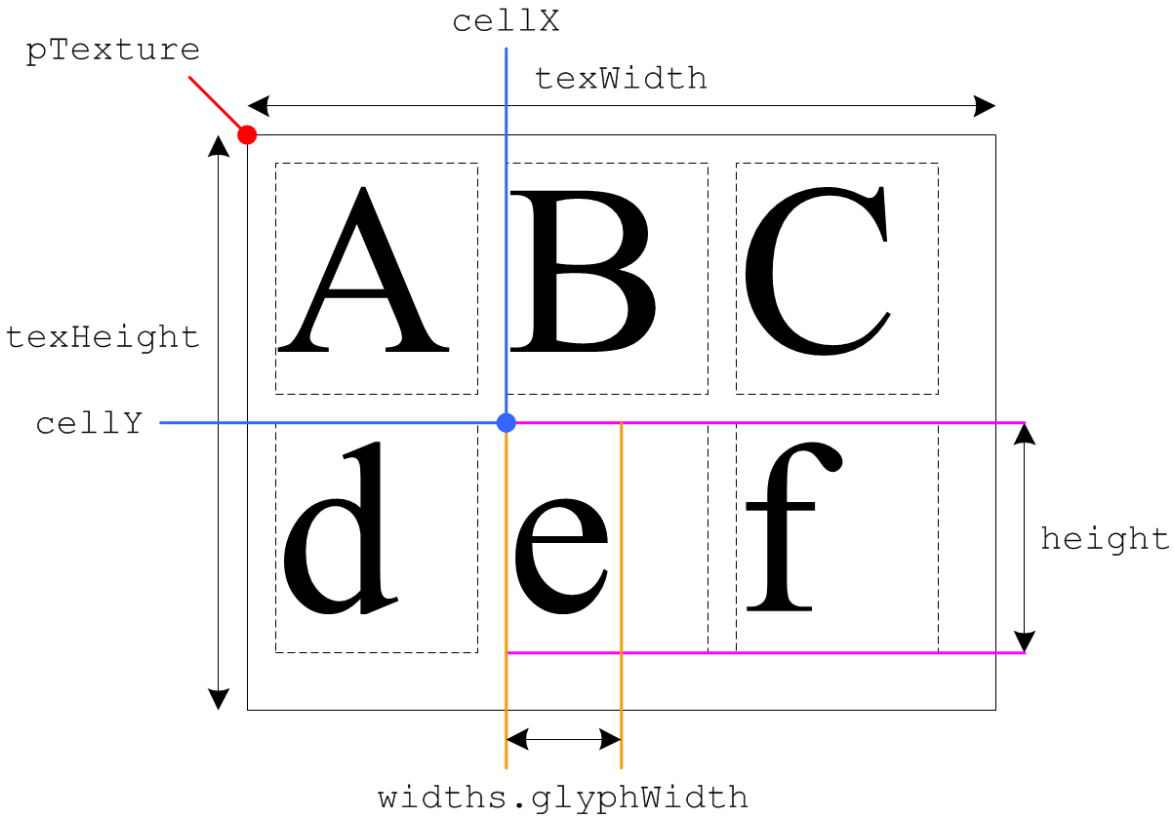
シートのフォーマットの定義と glTexImage2D() の引数に指定すべき値の対応は以下のとおりです。

表 4-2. シートのフォーマットの定義と glTexImage2D() の引数に指定すべき値

フォーマット	format	type
FONT_SHEET_FORMAT_A4	GL_ALPHA_NATIVE_DMP	GL_UNSIGNED_4BITS_DMP
FONT_SHEET_FORMAT_A8	GL_ALPHA_NATIVE_DMP	GL_UNSIGNED_BYTE
FONT_SHEET_FORMAT_LA4	GL_LUMINANCE_ALPHA_NATIVE_DMP	GL_UNSIGNED_BYTE_4_4_DMP
FONT_SHEET_FORMAT_LA8	GL_LUMINANCE_ALPHA_NATIVE_DMP	GL_UNSIGNED_BYTE
FONT_SHEET_FORMAT_RGBA4	GL_RGBA_NATIVE_DMP	GL_UNSIGNED_SHORT_4_4_4_4
FONT_SHEET_FORMAT_RGB5A1	GL_RGBA_NATIVE_DMP	GL_UNSIGNED_SHORT_5_5_5_1
FONT_SHEET_FORMAT_RGBA8	GL_RGBA_NATIVE_DMP	GL_UNSIGNED_BYTE
FONT_SHEET_FORMAT_RGB565	GL_RGB_NATIVE_DMP	GL_UNSIGNED_SHORT_5_6_5
FONT_SHEET_FORMAT_RGB8	GL_RGB_NATIVE_DMP	GL_UNSIGNED_BYTE

表示するグリフイメージの、シート内での位置を特定するために必要なメンバは以下のとおりです。

図 4-1. グリフイメージの位置を特定するために必要な Glyph 構造体のメンバ



3DS のテクスチャ座標は左下が原点であるのに対し、セル(グリフイメージ)の座標が左上を原点としていることに注意が必要です。そのため、テクスチャ座標の計算は以下のサンプルコードのように実装します。

コード 4-2. テクスチャ座標の計算

```
Glyph glyph;
myFont.GetGlyph(&glyph, character);

const f32 texLeft   = 1.0f * glyph.cellX / glyph.texWidth;
const f32 texRight  = 1.0f * (glyph.cellX + glyph.widths.glyphWidth)
                      / glyph.texWidth;
// グリフイメージの座標とテクスチャ座標の原点の違いに注意すること
const f32 texTop    = 1.0f * (glyph.texHeight - glyph.cellY) / glyph.texHeight;
const f32 texBottom = 1.0f * (glyph.texHeight - (glyph.cellY + glyph.height))
                      / glyph.texHeight;
```

4.2. FONT ライブラリのクラスを利用する

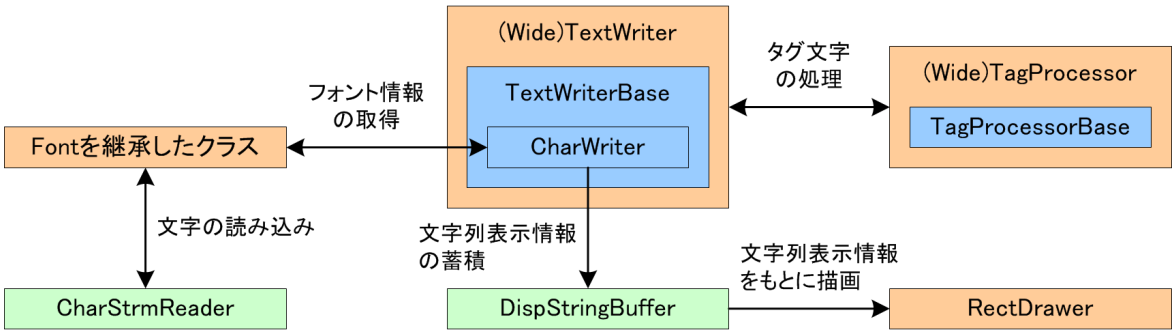
FONT ライブラリでは、文字(列)を描画するためのクラスを用意しています。また、単純な文字の描画だけでなく、以下のような機能を有しています。

- 文字色の変更
- 文字ごとの縦または横方向のグラデーション
- 文字の拡大縮小
- 強制的な等幅フォント表示
- 文字間、行間、タブ幅の処理
- 左右寄せ

- 文字列の自動改行
- フォーマット文字列による描画(printf 相当)
- タグ文字列の処理(カスタマイズ可能)

FONT ライブラリによる文字の描画で使用するクラスは複数あり、以下のように連携して処理を行っています。その中で、アプリケーションでインスタンスを生成し、描画のための設定を行うクラスは、下図で橙色の四角形に記述されているクラスです。ちなみに、青色が継承元のクラス、緑色が暗黙的に使用されるクラスです。

図 4-2. FONT ライブラリのクラス構成(描画部分)



FONT ライブラリのクラスを利用して文字の描画を行う場合は、3D コマンド(PICA レジスタ情報)を直接生成する RectDrawer クラスの使用を推奨します。

表 4-3. 描画に使用するクラスで行われる処理

クラス	行われる処理
CharWriter	文字単体の描画を行います。TextWriter を使用しない場合は、このクラスを継承して、文字列の成形などを行ってください。
TextWriter	文字列の描画を行います。文字列を成形するための基本的な機能を備えています。
TagProcessor	タグ文字(0x0000～0x001F の制御文字)の処理を行います。改行(0x000A)とタブ文字(0x0009)以外にも対応したい場合は、継承したクラスをカスタマイズしてください。
RectDrawer	TextWriter(CharWriter)クラスが蓄積した、文字列表示情報から 3D コマンドを生成します。専用のシェーダプログラムに合わせた 3D コマンドを生成します。

4.2.1. アプリケーションでの実装手順

ここでは、FONT ライブラリのクラスを利用して文字列を画面に表示するまでに必要な手順を、サンプルデモに収録されている ResFont デモのソースを交えて説明します。

4.2.1.1. フォントリソースのロードと構築

文字の描画に使用するフォントリソースのロードおよび構築を最初に行います。サンプルデモでは、`InitFont()` に ResFont クラスのインスタンスへのポインタとフォントリソースのファイル名(拡張子は bcfnt)を渡しています。

コード 4-3. フォントリソースのロードと構築

```
//------------------------------------------------------------------------
//! @brief      ResFontを構築します。
//!
//! @param[out] pFont 構築するフォントへのポインタ。
//! @param[in]  filePath ロードするフォントリソースファイル名。
//!
//! @return     ResFont構築の成否を返します。
//------------------------------------------------------------------------
bool
InitFont(
    nn::font::ResFont*  pFont,
    const char*         filePath)
{
    // フォントリソースをロードします
    nn::fs::FileReader fontReader(filePath);

    s32 fileSize = (s32)fontReader.GetSize();
    if ( fileSize <= 0 )
    {
        return false;
    }

    void* buffer = s_AppHeap.Allocate(fileSize, nn::font::GlyphDataAlignment);
    if (buffer == NULL)
    {
        return false;
    }

    s32 readSize = fontReader.Read(buffer, fileSize);
    if (readSize != fileSize)
    {
        s_AppHeap.Free(buffer);
        return false;
    }

    // フォントリソースをセットします
    bool bSuccess = pFont->SetResource(buffer);
    NN_ASSERT(bSuccess);

    //--- 既にリソースをセット済みであるか，ロード済みであるか、
    //    リソースが不正な場合に失敗します。
    if (! bSuccess)
    {
        s_AppHeap.Free(buffer);
    }

    // 描画用バッファを設定します。
    const u32 drawBufferSize = nn::font::ResFont::GetDrawBufferSize(buffer);
    void* drawBuffer = s_AppHeap.Allocate(drawBufferSize, 4);
    NN_NULL_ASSERT(drawBuffer);
    pFont->SetDrawBuffer(drawBuffer);
```

```
    return bSuccess;
}
```

FONT ライブラリでは、ロードされたフォントリソースの、グリフイメージが描かれているシートに GPU が直接アクセスする設定でテクスチャをロードするため、フォントリソースがデバイスメモリ上の 128 Byte アライメントのバッファに読み込まれていなければならないことに注意してください。なお、描画用バッファは 4 Byte アライメントの制限のみで、バッファをデバイスメモリ上に確保しなくてもかまいません。

ResFont クラス以外のフォントリソースを使用した場合の制限については、「3.3. ArchiveFont クラス」や「3.4. PackedFont クラス」を参照してください。

4.2.1.2. シェーダの初期化

表示で使用するシェーダバイナリをロードし、RectDrawer クラスが使用するバッファの確保を行います。サンプルデモでは、`InitShaders()` に RectDrawer クラスのインスタンスへのポインタを渡しています。

コード 4-4. シェーダの初期化

```
//------------------------------------------------------------------------
//! @brief          シェーダの初期化を行います。
//!
//! @param[in,out]  pDrawer 初期化するインスタンスへのポインタ。
//------------------------------------------------------------------------
void*
InitShaders(nn::font::RectDrawer* pDrawer)
{
    nn::fs::FileReader shaderReader(s_ShaderBinaryFilePath);

    const u32 fileSize = (u32)shaderReader.GetSize();

    void* shaderBinary = s_AppHeap.Allocate(fileSize);
    NN_NULL_ASSERT(shaderBinary);

#ifndef NN_BUILD_RELEASE
    s32 read =
#endif // NN_BUILD_RELEASE
    shaderReader.Read(shaderBinary, fileSize);
    NN_ASSERT(read == fileSize);

    const u32 vtxBufCmdBufSize =
        nn::font::RectDrawer::GetVertexBufferCommandBufferSize(
            shaderBinary, fileSize);
    void *const vtxBufCmdBuf = s_AppHeap.Allocate(vtxBufCmdBufSize);
    NN_NULL_ASSERT(vtxBufCmdBuf);
    pDrawer->Initialize(vtxBufCmdBuf, shaderBinary, fileSize);

    s_AppHeap.Free(shaderBinary);

    return vtxBufCmdBuf;
}
```

RectDrawer クラスでは、頂点バッファとコマンドバッファの 2 つのバッファを使用します。頂点バッファには、描画のための頂点座標が格納されます。コマンドバッファには描画のコマンドが格納されますが、シェーダバイナリのロードコマンド(プログラム、Swizzle パターン、定数レジスタ)などが含まれます。これらのバッファは GPU が直接アクセスするため、デバイスメモリか

ら確保しなければなりません。バッファのアライメントの指定は特にありませんが、4 Byte または 16 Byte にすることを推奨します。

サンプルデモでは、2 つのバッファを 1 つの領域で確保していますが、それぞれのバッファに必要なサイズを取得する関数や、バッファを分けて確保した場合の初期化関数が用意されています。初期化に成功したあとは、シェーダバイナリをロードしていたバッファを解放することができます。

SDK には、RectDrawer クラスで使用するシェーダバイナリとして nnfont_RectDrawerShader.shbin が収録されています。このシェーダは、射影行列にスクリーンの左上が原点でスクリーンと画面のサイズが一致する正射影行列(ただし、Y 軸と Z 軸の方向が 3DS とは逆です)、モデルビュー行列に単位行列を設定することを想定しています。文字の表示位置や大きさの調整には、変形用の行列を浮動小数点定数レジスタに蓄積して対応しています。

4.2.1.3. 表示文字列用バッファの確保

文字列表示情報を蓄積するバッファ(表示文字列用バッファ)を確保します。表示文字列用バッファに必要なサイズは最大の表示文字数から計算されます。文字列の描画が完了したあとならば、同じバッファを使って文字列表示情報の蓄積を行うことができます。フォントや表示する文字を変更しない場合は、蓄積していた文字列表示情報をそのまま再利用することができ、さらに単一色で文字列を描画していた場合は、再度文字列表示情報の蓄積を行わずに、文字色のみを変更することができます。

コード 4–5. 表示文字列用バッファの確保

```
//------------------------------------------------------------------------
//! @brief      表示文字列用バッファを確保します。
//!
//! @param[in]  charMax 表示する文字列の最大文字数。
//!
//! @return     確保した表示文字列用バッファへのポインタを返します。
//------------------------------------------------------------------------
nn::font::DispStringBuffer*
AllocDispStringBuffer(int charMax)
{
    const u32 DrawBufferSize =
        nn::font::CharWriter::GetDispStringBufferSize(charMax);
    void *const bufMem = s_AppHeap.Allocate(DrawBufferSize);
    NN_NULL_ASSERT(bufMem);

    return nn::font::CharWriter::InitDispStringBuffer(bufMem, charMax);
}
```

表示文字列用バッファはデバイスメモリ上に確保する必要はありません。バッファのアライメントの指定は特にありませんが、4 の倍数にすることを推奨します。

4.2.1.4. 描画設定

FONT ライブラリは描画のために必要最小限の設定を行いますので、以下の項目については、文字を描画する前にアプリケーションで適切に設定する必要があります。

- カリング
- シザーテスト
- ポリゴンオフセット
- アーリーデプステスト
- デプステスト
- ステンシルテスト

- マスク処理
- フレームバッファオブジェクト

アプリケーションで 3D モデル等を表示するためにカリングやデプステストを設定している場合、文字を描画する前に設定を変更しなければ、文字の描画に直前の設定が影響を与えることになります。
サンプルデモでは、InitDraw() で描画設定を行うコマンドを生成しています。

コード 4-6. 描画設定

```
//------------------------------------------------------------------------
//! @brief      描画の初期設定を行います。
//!
//! @param[in]  width   画面の幅。
//! @param[in]  height  画面の高さ。
//------------------------------------------------------------------------
void
InitDraw(
    int width,
    int height
)
{
    // カラーバッファ情報
    // LCDの向きに合わせて、幅と高さを入れ替えています。
    const nn::font::ColorBufferInfo colBufInfo =
    { width, height, PICA_DATA_DEPTH24_STENCIL8_EXT };

    const u32 screenSettingCommands[] =
    {
        // ビューポートの設定
        NN_FONT_CMD_SET_VIEWPORT( 0, 0, colBufInfo.width, colBufInfo.height ),

        // シザー処理を無効
        NN_FONT_CMD_SET_DISABLE_SCISSOR( colBufInfo ),

        // wバッファの無効化
        // デプスレンジの設定
        // ポリゴンオフセットの無効化
        NN_FONT_CMD_SET_WBUFFER_DEPTHRANGE_POLYGONOFFSET(
            0.0f,   // wScale : 0.0 でWバッファが無効
            0.0f,   // depth range near
            1.0f,   // depth range far
            0,      // polygon offset units : 0.0 でポリゴンオフセットが無効
            colBufInfo),
    };
    nngxAdd3DCommand(screenSettingCommands, sizeof(screenSettingCommands), true);

    static const u32 s_InitCommands[] =
    {
        // カリングを無効
        NN_FONT_CMD_SET_CULL_FACE( NN_FONT_CMD_CULL_FACE_DISABLE ),

        // ステンシルテストを無効
        NN_FONT_CMD_SET_DISABLE_STENCIL_TEST(),
```

```
        // デプステストを無効
        // カラーバッファの全ての成分を書き込み可
        NN_FONT_CMD_SET_DEPTH_FUNC_COLOR_MASK(
            false,  // isDepthTestEnabled
            0,      // depthFunc
            true,   // depthMask
            true,   // red
            true,   // green
            true,   // blue
            true),  // alpha

        // アーリーデプステストを無効
        NN_FONT_CMD_SET_ENABLE_EARLY_DEPTH_TEST( false ),

        // フレームバッファアクセス制御
        NN_FONT_CMD_SET_FBACCESS(
            true,   // colorRead
            true,   // colorWrite
            false,  // depthRead
            false,  // depthWrite
            false,  // stencilRead
            false), // stencilWrite
    };
    nngxAdd3DCommand(s_InitCommands, sizeof(s_InitCommands), true);
}
```

4.2.1.5. 描画ごとに行う処理

ここまでに行った処理は初期化にあたり、ここでは描画ごとに行う処理を説明します。

表示位置や文字色を指定した文字列の描画は (Wide)TextWriter クラスで行います。このクラスは、フレームごとにインスタンスを生成することを想定していますので、明示的な初期化関数を持っていません。デフォルトでタグ文字の処理を行うクラスも設定されますので、独自のタグ文字を処理する必要がない限りは、生成したインスタンスをそのまま使用することができます。また、文字列の書式を展開するバッファを設定していない場合は、書式つきの `Printf()` を呼び出したときに、スタックから 256 文字分のメモリが確保されます。スタックを消費したくない場合や、256 文字分以上のバッファが必要な場合はアプリケーションで確保し、`SetBuffer()` で設定してください。返り値には以前に設定されていたバッファへのポインタが返されます。

描画された文字列は、文字列表示情報として 1 文字ずつ蓄積されますので、「4.2.1.3. 表示文字列用バッファの確保」で確保した文字列情報を蓄積するバッファを `SetDispStringBuffer()` で TextWriter クラスに設定してください。描画に使用するフォントリソースは `SetFont()` で設定します。

表示位置(カーソル位置)の設定は `SetCursor()` または `MoveCursor()` で行います。前者は新しい表示位置そのままを指定し、後者は差分で表示位置を指定します。X, Y, Z 座標それぞれを個別に設定することも、まとめて設定することもできます。デフォルトの設定は (0.0, 0.0, 0.0) です。

表示位置を基準にして、文字列をどのように配置するのかは `SetDrawFlag()` で設定します。文字列の配置は、文字列を描画する矩形領域の原点の配置を指定するフラグ(水平・垂直方向)と、文字列を寄せる方向を指定するフラグ(水平方向のみ)との論理和で指定します。フラグは `nn::font::PositionFlag` 列挙子で定義されています。デフォルトの設定は、`HORIZONTAL_ALIGN_LEFT | HORIZONTAL_ORIGIN_LEFT | VERTICAL_ORIGIN_TOP` で、下図はデフォルト設定に対して各フラグを変更したときの配置の変化を示したものです。

図 4-3. フラグの変更による配置の変化

文字の拡大縮小は SetScale() または SetFontSize() で行います。前者はフォントの幅と高さを基準とする倍率で指定し、後者は表示する文字の幅と高さをピクセルで指定します。拡大縮小はアセントとディセントにも影響を及ぼします。そのため、GetFontAscent() や GetFontDescent() は、フォントの持つアセントやディセントとは異なる値を返す可能性があります。デフォルトの設定は等倍です。

文字色の設定は SetTextColor() で行います。また、SetGradationMode() で水平・垂直方向にグラデーション効果を与えることができます。グラデーション効果に対しては、SetColorMapping() で線形変換を設定することができ、設定によっては色の変化を逆転させることが可能です。なお、SetAlpha() で文字色とは別に、追加のアルファ値を設定することができます。デフォルトの設定は、文字色が (255, 255, 255, 255)、グラデーション効果なし、線形変換なし、追加のアルファ値が 255 です。

1 行の高さの設定は SetLineHeight() で行います。この設定は改行幅ではなく、行間を自動的に調整します。行間を直接設定する場合は SetLineSpace() で行ってください。そのほか、タブ幅の設定は SetTabWidth()、文字間の設定は SetCharSpace() でそれぞれ行います。デフォルトの設定は、行間と文字間が 0、タブ幅は 4 文字です。

フォントの設定に関係なく、強制的に文字列を等幅で描画する場合は SetFixedWidth() で描画幅を設定し、EnableFixedWidth(true) で強制等幅描画を有効に設定してください。EnableFixedWidth() に false を指定すると強制等幅描画を無効に戻すことができます。デフォルトの設定は無効です。

ここまでの設定は文字の表示に関する設定であり、TextWriter クラスに対して行っていました。文字を描画するためのグラフィックス関連の設定は RectDrawer クラスに対して行います。

DrawBegin() で描画を開始するための初期化を行います。描画モードやテクスチャなどを初期化するための描画コマンドが生成され、内部変数などの初期化も行いますので、SetParallax() で視差を設定する場合はこの関数を呼び出したあとで設定しなければなりません。

射影行列とビュー行列の設定は SetProjectionMtx() と SetViewMtxForText() で行います。ライブラリが想定している射影行列は左上原点、スクリーンサイズと画面サイズが一致し、Y 軸と Z 軸の正方向が CTR の座標系と逆転している行列です。ビュー行列は単位行列を想定しています。

文字列の描画は TextWriter クラスで行います。文字を描画すると、文字列表示情報として表示文字列用バッファに一旦蓄積され、蓄積された情報から RectDrawer クラスが描画コマンドを生成します。

TextWriter::StartPrint() で文字列描画の開始を宣言します。このとき、表示文字列用バッファがクリアされます。

TextWriter::Print() または TextWriter::Printf() で文字列を描画します。前者は書式文字列の指定なし、後者は書式文字列の指定ありです。

文字列の描画が完了したら、TextWriter::EndPrint() で描画完了を宣言します。そのあとに、RectDrawer::BuildTextCommand() で蓄積された情報をもとに描画コマンドを生成し、TextWriter::UseCommandBuffer() で描画コマンドをコマンドバッファへ送ります。

最後に RectDrawe::DrawEnd() を呼び出すことで描画コマンドのキックなどが行われ、レンダーバッファに文字が描画されます。

描画する文字列の内容や表示位置に変更がない場合は、TextWriter::StartPrint() から TextWriter::EndPrint() と RectDrawer::BuildTextCommand() を再度実行することなく、同じ描画結果を得ることができます。さらに、単一色で描画していた場合は、文字色のみを変更することもできます。ただし、これは蓄積された文字列表示情報が保持されている場合に限ります。

以下のコードは、サンプルデモの該当部分です。

コード 4-7. 行列の設定と文字列の描画

```
//-----------------------------------------------------------------------------
//! @brief      文字列表示用にモデルビュー行列と射影行列を設定します。
//!
//! @param[in]  pDrawer RectDrawerオブジェクトへのポインタ。
//! @param[in]  width   画面の幅。
//! @param[in]  height  画面の高さ。
//-----------------------------------------------------------------------------
void
SetupTextCamera(
    nn::font::RectDrawer*   pDrawer,
    int                     width,
    int                     height
)
{
    // 射影行列を正射影に設定
    {
        // 左上原点とし、Y軸とZ軸の向きが逆になるように設定します。
        nn::math::MTX44 proj;
        f32 znear   = 0.0f;
        f32 zfar    = -1.0f;
        f32 t       = 0;
        f32 b       = static_cast<f32>(width);
        f32 l       = 0;
        f32 r       = static_cast<f32>(height);
        nn::math::MTX44OrthoPivot(
            &proj, l, r, b, t, znear, zfar, nn::math::PIVOT_UPSIDE_TO_TOP);
```

```
        pDrawer->SetProjectionMtx(proj);
    }

    // モデルビュー行列を単位行列に設定
    {
        nn::math::MTX34 mv;
        nn::math::MTX34Identity(&mv);
        pDrawer->SetViewMtxForText(mv);
    }
}

//------------------------------------------------------------------------
//! @brief ASCII文字列を描画します。
//!
//! @param[in]  pDrawer         RectDrawerオブジェクトへのポインタ。
//! @param[in]  pDrawStringBuf  DispStringBufferオブジェクトへのポインタ。
//! @param[in]  pFont           フォントへのポインタ。
//! @param[in]  width           画面の幅。
//! @param[in]  height          画面の高さ。
//------------------------------------------------------------------------
void
DrawAscii(
    nn::font::RectDrawer*       pDrawer,
    nn::font::DispStringBuffer* pDrawStringBuf,
    nn::font::ResFont*          pFont,
    int                         width,
    int                         height
)
{
    nn::font::TextWriter writer;
    writer.SetDispStringBuffer(pDrawStringBuf);
    writer.SetFont(pFont);
    writer.SetCursor(0, 0);

    // 文字列が変更されないので、文字列の描画コマンドを一度だけ作成します。
    if (! s_InitAsciiString)
    {
        writer.StartPrint();
            (void)writer.Print("DEMO: ResFont\n");
            (void)writer.Print("\n");
            // ASCIIの文字見本を表示
            (void)writer.Print("All ASCII Character listing:\n");
            (void)writer.Print("\n");
            (void)writer.Print(" !\"#$%&'()*+,-./\n");
            (void)writer.Print("0123456789:;<=>?\n");
            (void)writer.Print("@ABCDEFGHIJKLMNO\n");
            (void)writer.Print("PQRSTUVWXYZ[\\\]^_\n");
            (void)writer.Print("`abcdefghijklmno\n");
            (void)writer.Print("pqrstuvwxyz{|}~\n");
        writer.EndPrint();
        pDrawer->BuildTextCommand(&writer);

        s_InitAsciiString = true;
```

```
    }

    // 文字の色は、文字列の描画コマンドを再作成しなくても変更できます。
    writer.SetTextColor(nn::util::Color8(s_Color, 255, s_Color, 255));
    s_Color++;

    pDrawer->DrawBegin();
        SetupTextCamera(pDrawer, width, height);
        writer.UseCommandBuffer();
    pDrawer->DrawEnd();
}
```

4.2.1.6. ライブラリによる GPU 設定の変化

RectDrawer クラスを利用してフォントを描画すると、テクスチャのサンプラータイプや頂点属性のロードアドレスなど、様々な GPU 設定が変更されます。そのため、フォントを描画したあとは GPU のすべてのステートをバリデートし、アプリケーションで GPU 設定をすべてやり直すことを推奨していますが、これは冗長な処理となる可能性があります。

ここでは、フォントの描画時に行われる GPU 設定を、GPU の再設定にかかるコストを下げるための参考資料として紹介します。

コンバイナの設定

コンバイナは 3 ～ 5 を使用し、以下のように設定されています。シートのテクスチャフォーマットがアルファ成分を持つものかどうかで、一部の設定が異なります。ちなみに、文字色の表現は頂点カラーの設定で行われています。

表 4-4. コンバイナ 3 の設定

設定	カラー	アルファ
ソース 0	GL_TEXTURE0	GL_TEXTURE0
ソース 1	GL_CONSTANT	GL_CONSTANT
ソース 2	GL_CONSTANT	GL_CONSTANT
オペランド 0	(アルファなし)GL_SRC_COLOR (アルファあり)GL_ONE_MINUS_SRC_COLOR	GL_SRC_ALPHA
オペランド 1	GL_SRC_COLOR	GL_SRC_ALPHA
オペランド 2	GL_SRC_COLOR	GL_SRC_ALPHA
コンバイン	GL_MODULATE	GL_MODULATE
スケール	1.0	1.0
定数カラー	白(1.0, 1.0, 1.0, 1.0)	

表 4-5. コンバイナ 4 の設定

設定	カラー	アルファ
ソース 0	GL_TEXTURE0	GL_TEXTURE0
ソース 1	GL_CONSTANT	GL_CONSTANT
ソース 2	GL_PREVIOUS	GL_PREVIOUS

オペランド 0	(アルファなし)GL_ONE_MINUS_SRC_COLOR (アルファあり)GL_SRC_COLOR	GL_ONE_MINUS_SRC_ALPHA
オペランド 1	GL_SRC_COLOR	GL_SRC_ALPHA
オペランド 2	GL_SRC_COLOR	GL_SRC_ALPHA
コンバイン	GL_MULT_ADD_DMP	GL_MULT_ADD_DMP
スケール	1.0	1.0
定数カラー	黒(0.0, 0.0, 0.0, 0.0)	

表 4-6. コンバイナ 5 の設定

設定	カラー	アルファ
ソース 0	GL_PRIMARY_COLOR	GL_PRIMARY_COLOR
ソース 1	GL_PREVIOUS	GL_PREVIOUS
ソース 2	GL_PREVIOUS	GL_PREVIOUS
オペランド 0	GL_SRC_COLOR	GL_SRC_ALPHA
オペランド 1	GL_SRC_COLOR	GL_SRC_ALPHA
オペランド 2	GL_SRC_COLOR	GL_SRC_ALPHA
コンバイン	GL_MODULATE	GL_MODULATE
スケール	1.0	1.0
定数カラー	※ 未設定のため不定	

予約フラグメントシェーダの設定

予約フラグメントシェーダのうち、以下の機能の設定が変更されます。

表 4-7. 予約フラグメントシェーダの設定

機能	設定
フラグメントオペレーションモード	標準モード(GL_FRAGOP_MODE_GL_DMP)
フラグメントライティング	無効
シャドウ(シャドウテクスチャ)	無効
フォグ(ガス)	無効
アルファテスト	無効
ブレンディング	有効

テクスチャの設定

テクスチャ 0 のみを使用し、GL_TEXTURE_2D に設定します。テクスチャイメージのアドレスや解像度、フィルタ設定などが変更されます。

シェーダの設定

整数レジスタ i0 と浮動小数点定数レジスタ c0～c95 に、描画に必要なデータを設定しています。

シェーダプログラムは、頂点座標、頂点カラー、テクスチャ座標 0、テクスチャ座標 1、テクスチャ座標 2 を出力していますが、ジオメトリシェーダを使用することはできません。

4.2.2. タグ文字の処理をカスタマイズする

nn::font::TagProcessorBase クラスを継承したクラスをカスタマイズすることで、アプリケーション独自のタグ文字（0x0000～0x001F）の処理を実装することができます。

文字列を描画している途中に描画すべき文字としてタグ文字が現れたとき、TextWriter クラスは自身に設定されている TagProcessor クラスにタグ文字の処理を行わせます。TextWriter クラスには、タブ（0x0009）と改行（0x000A）を処理する TagProcessor クラスがデフォルトで設定されていますが、`SetTagProcessor()` で独自の TagProcessor クラスを使用するように設定することができます。

タグ文字の処理をカスタマイズする場合は、`Process()` と `CalcRect()` をオーバーライドします。

コード 4-8. タグ文字の処理をカスタマイズする際にオーバーライドする関数

```
virtual Operation Process( u16                      code,
                           PrintContext<CharType>*  pContext);
virtual Operation CalcRect(util::Rect*              pRect,
                           u16                      code,
                           PrintContext<CharType>*  pContext);
```

これらの関数で返した値により、TextWriter クラスは文字を描画する座標の調整を行います。同じタグ文字に対して、`Process()` と `CalcRect()` が同じ返り値となるように実装してください。

表 4-8. タグ文字の処理で返す値の定義

定義	TextWriter 側で行われる処理
OPERATION_DEFAULT	次の文字との文字間を、行頭ならば空けず、行頭以外ならば空けます。
OPERATION_NO_CHAR_SPACE	次の文字との文字間を必ず空けません。
OPERATION_CHAR_SPACE	次の文字との文字間を必ず空けます。
OPERATION_NEXT_LINE	改行時の処理を行います。X 座標のみが調整され、Y 座標の調整はタグの処理中に行う必要があります。
OPERATION_END_DRAW	文字列の途中で文字列の描画を終了します。

引数 *pContext* のメンバには、TextWriter クラスへのポインタがありますので、タグ文字によって文字色を変更するなどの処理を行うことができます。

5. フォントの作成

FONT ライブラリで使用するフォントリソースは、ctr_FontConverter（CUI 版は ctr_FontConverterConsole）を利用して、PC にインストールされているフォントや画像データから作成することができます。

> **注意：** CTR-SDK には、いかなるフォントのライセンスも付属していません。
>
> ctr_FontConverter では FONT ライブラリによる文字表示に利用するフォントリソースを、PC にインストールされているフォントや画像データから作成することができますが、これらのフォントを利用したソフトを発売するには利用するフォントのライセンスが必要になります。必ず、ゲームソフトごとに適切なライセンスを取得してください。

ctr_FontConverter はフォントリソースの作成だけでなく、フォントを画像データとして出力することができます。出力可能な画像データのフォーマットは BMP 形式と TGA 形式で、これらの画像データはフォントリソースの作成に利用可能です。つまり、出力した画像データを手直ししてフォントリソースを再作成したり、画像データを結合して複数のフォントが含まれているフォントリソースを作成したりすることができます。

5.1. 作成に利用するファイル

ctr_FontConverter がフォントリソースを作成する際には、以下のファイルを利用します。

- 文字フィルタファイル
- グリフグループファイル
- 文字順序ファイル

これらのファイルはすべて XML（eXtensible Markup Language）で記述し、おおまかな構造は共通しています。

5.1.1. 共通する構造

最上位の要素の名前は異なりますが、version 属性を持ち、head 要素と body 要素を包含している構造は共通しています。また、head 要素の構造は共通していますが、body 要素の構造はそれぞれ異なります。

下表は共通部分の要素の一覧です。表中で太字の要素および属性は必須の項目です。

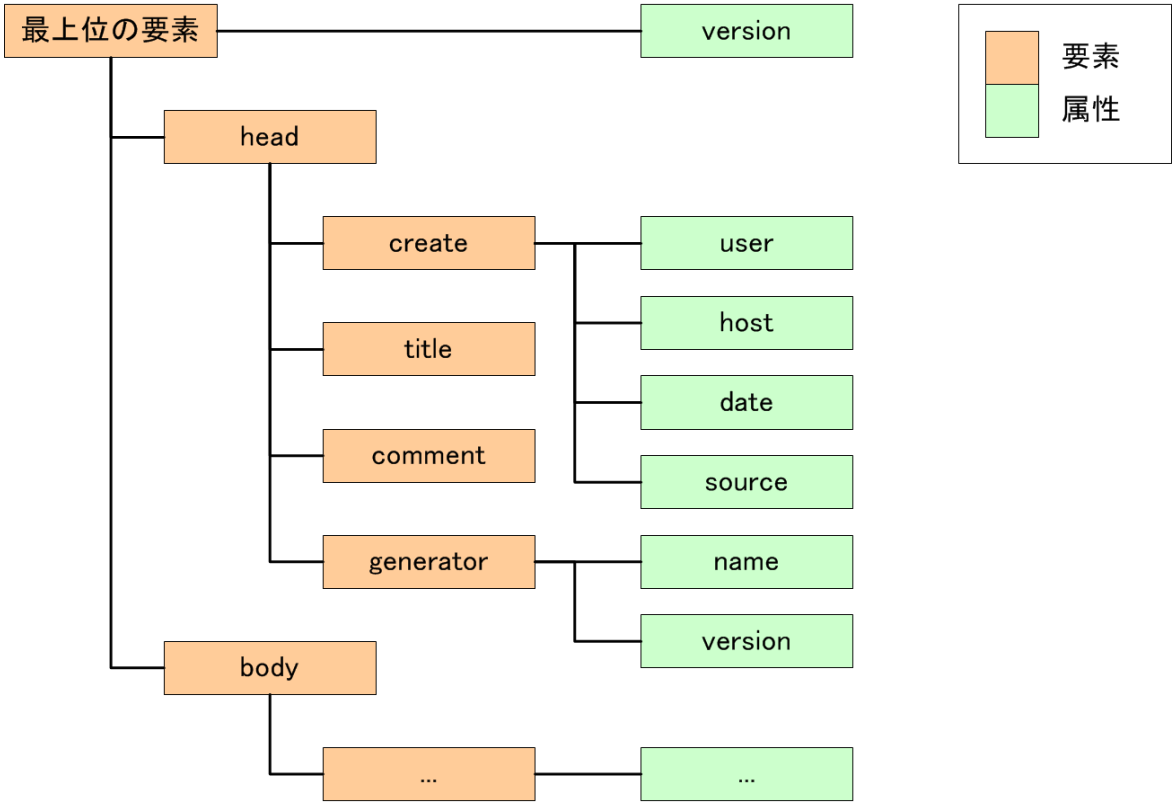
表 5-1. 共通部分の要素一覧

要素	包含可能要素（上段） 属性（下段）	説明
最上位の要素	head, **body**	最上位の要素で、要素名はファイルごとに異なります。 包含可能要素と属性はすべてのファイルで共通です。
	version	
head	create, title, comment, generator	ファイルそのものの情報（ヘッダ）を格納します。 すべてのファイルで共通する構造を持つ要素です。
	なし	
create	なし	ファイル作成時の情報を格納します。 user 属性にはファイルを作成した PC のユーザー名を、host 属性には PC 名を格納します。date 属性には作成日時を ISO 8601 に規定される日付と時刻の拡張形式で格納します。 source 属性にはデータの元となるファイルが存在する場合に、元データのファイル名を格納します。
	user, host, date, source	

title	なし	ファイルのタイトルを格納します。 GUI 版 ctr_FontConverter で表示される選択項目に使用されることがあります。
	なし	
comment	なし	ファイルの作成者のコメントを格納します。
	なし	
generator	なし	ファイルを作成したアプリケーションの情報を格納します。 name 属性にはアプリケーションを識別するための文字列を、version 属性にはアプリケーションのバージョン文字列を格納します。
	name, version	
body	ファイルによる	ファイルの本体となる情報を格納します。
	なし	

下図は要素の階層を図示したものです。

図 5-1. 共通部分の要素の階層図



5.1.2. 文字フィルタファイル

文字フィルタファイル（拡張子 xllt）は、必要な文字だけを含む、コンパクトなフォントリソースの作成に利用します。フォントの変換時に、ファイルに指定されている文字以外を出力しないようにすることができます。変換元のデータに含まれている、すべての文字を対象にして作成する場合は不要です。

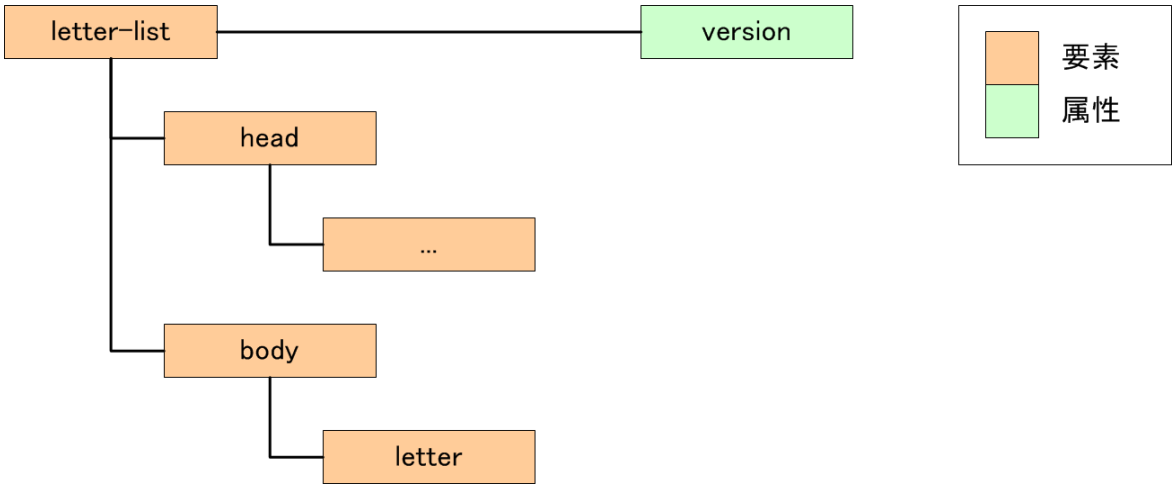
最上位の要素は「letter-list」で、version 属性には「1.0」を格納します。body 要素は letter 要素を 1 つだけ包含します。

表 5-2. 文字フィルタファイルの要素一覧(共通部分以外)

要素	包含可能要素(上段) 属性(下段)	説明
letter-list	head, **body**	最上位の要素です。 version 属性には 1.0 を格納します。
	version	
body	**letter**	ファイルの本体となる情報を格納します。 包含可能な要素は letter 要素 1 つだけです。
	なし	
letter	なし	出力する文字を定義します。 文字の指定は出力したい文字を直接記述することで行います。ただし空白文字(半角スペースとタブ文字)は無視され、変換結果には必ず半角スペースが追加されます。
	なし	

下図は要素の階層を図示したものです。

図 5-2. 文字フィルタファイルの要素の階層図



以下のコードは、文字フィルタファイルのサンプル(sample.xllt)の内容です。

コード 5-1. 文字フィルタファイルのサンプル(sample.xllt)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE letter-list SYSTEM "letter-list.dtd">
<letter-list version="1.0">
    <head>
        <create user="sample" date="2005-02-18T10:51:13" />
        <title>xlltサンプル</title>
        <comment>文字フィルタファイルのサンプルです</comment>
    </head>
    <body>
        <letter>
            FontConverter
            あいうえかきくけ
            任天堂
        </letter>
    </body>
</letter-list>
```

5.1.3. グリフグループファイル

グリフグループファイル(拡張子 xggp)は、アーカイブフォント(圧縮フォント)の作成に利用します。グループセットを指定(CUI 版ならばコマンド引数 -op にファイルを指定)することで、ファイルに設定されている、グループの情報を含んだフォントリソースを作成することができます。

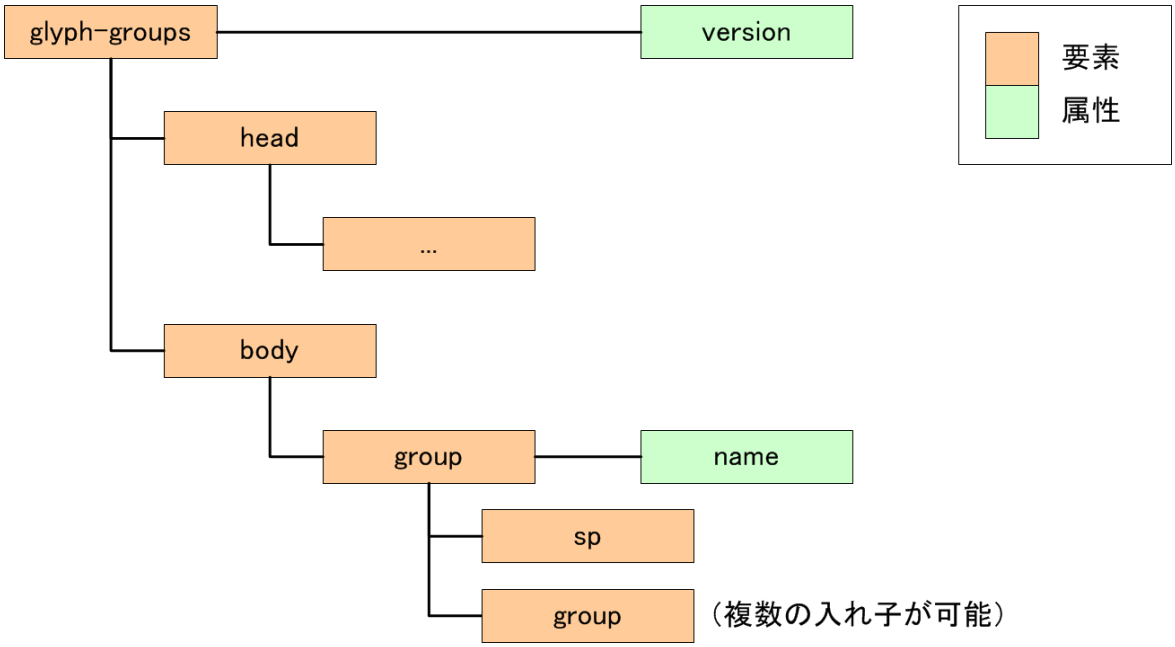
最上位の要素は「glyph-groups」で、version 属性には「1.0」を格納します。body 要素は group 要素を 1 つだけ包含します。

表 5-3. グリフグループファイルの要素一覧(共通部分以外)

要素	包含可能要素(上段) 属性(下段)	説明
glyph-groups	head, **body**	最上位の要素です。 version 属性には 1.0 を格納します。
	version	
body	**group**	ファイルの本体となる情報を格納します。 包含可能な要素は group 要素 1 つだけです。
	なし	
group	sp, group	グリフグループを定義します。 列挙された文字は、name 属性で指定された名前のグリフグループに含まれます。group 要素内では、複数の group 要素を入れ子にすることができます。 name 属性に使用できる文字は半角の英数字(0～9, a～z, A～Z)とアンダーバー(_)のみです。name 属性が重複するような group 要素は、同じファイル内に定義することができません。
	name	
sp	なし	group 要素内では半角スペースを直接記述しても無視されるため、 group 要素内で半角スペースを指定するために使用します。
	なし	

下図は要素の階層を図示したものです。

図 5-3. グリフグループファイルの要素の階層図



以下のコードは、グリフグループファイルのサンプル（sample.xggp）の内容を抜粋したものです。

コード 5-2. グリフグループファイルのサンプル（sample.xggp から抜粋）

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE glyph-groups SYSTEM "glyph-groups.dtd">
<glyph-groups version="1.0">
    <head>
        <create user="sample" date="2006-09-05T14:50:23" />
        <title>sample</title>
    </head>
    <body>
        <group name="all">
            <group name="ascii">
                <sp/> ! " # $ % & ' ( ) * + , - . /
                (中略)
                p q r s t u v w x y z { | } ~
            </group>
            <group name="european">
                € , ƒ „ ... † ‡
                ˆ ‰ Š ‹ Œ Ž
                (中略)
            </group>
        (中略)
        </group>
    </body>
</glyph-groups>
```

5.1.4. 文字順序ファイル

文字順序ファイル（拡張子 xlor）は、画像データからフォントリソースを作成する場合やフォントを画像データとして出力する場合に必要となります。入力に使用した場合は、画像データ内に描かれている文字の順序と、このファイルに設定されてい

る文字の順序は一致していなければなりません。出力に使用した場合は、このファイルに設定されている順番に文字が描かれるため、フィルタとしての役割も果たすことになります。

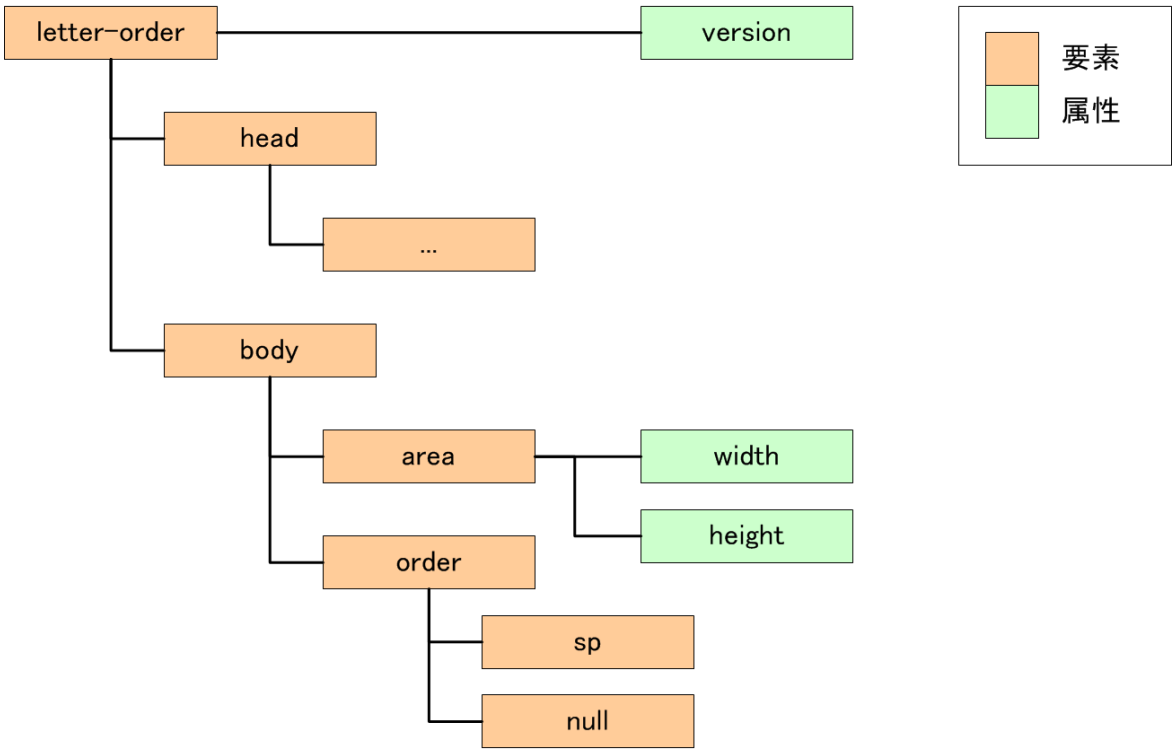
最上位の要素は「letter-order」で、version 属性には「1.0」または「1.1」を格納します。body 要素は area 要素と order 要素を包含します。

表 5-4. 文字順序ファイルの要素一覧（共通部分以外）

要素	包含可能要素（上段） 属性（下段）	説明
letter-order	head, **body**	最上位の要素です。 version 属性には 1.0 または 1.1 を格納します。 version 属性に 1.0 を指定した場合、order 要素内で同じ文字コードを持つ文字が複数記述されていてもエラーになりません。ただし、同じ文字コードのグリフが存在する場合は、その中で最初に記述されたグリフが採用されるため、グリフイメージを編集してもフォントリソースに反映されない可能性があります。version 属性に 1.1 を指定した場合は、order 要素内で同じ文字コードを持つ文字が複数記述されているとエラーになりますので、通常は 1.1 を指定することを推奨します。
	version	
body	area, **order**	ファイルの本体となる情報を格納します。 包含可能な要素は area 要素と order 要素です。
	なし	
area	なし	画像データに含まれているフォントイメージを、縦横の文字数で定義します。 width 属性は横方向の文字数を定義します。省略したときは 16 が指定されたものとして扱われます。 height 属性は縦方向の文字数を定義します。省略したときは order 要素で指定された文字を出力するために必要十分な値が指定されたものとして扱われます。
	width, height	
order	sp, null	出力する文字とその順序を定義します。 列挙された文字は、その順番どおりにフォントリソースに出力されます。
	なし	
sp	なし	order 要素内では半角スペースを直接記述しても無視されるため、order 要素内で半角スペースを指定するために使用します。
	なし	
null	なし	出力位置を 1 文字分飛ばします。該当位置のブロックをフォントリソースに出力したくない場合に使用します。
	なし	

下図は要素の階層を図示したものです。

図 5-4. 文字順序ファイルの要素の階層図



以下のコードは、文字順序ファイルのサンプル(cp1252.xlor)の内容を抜粋したものです。

コード 5-3. 文字順序ファイルのサンプル(cp1252.xlor から抜粋)

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE letter-order SYSTEM "letter-order.dtd">
<letter-order version="1.1">
    <head>
        <create user="sample" date="2005-02-18T10:51:13" />
        <title>European Language (CodePage 1252 / Latin-1)</title>
        <comment>Windows Code Page 1252 character set. It is super set of ISO 8859-
1 (Latin-1).</comment>
    </head>
    <body>
        <area width="16" />
        <order>
            <sp/> ! " # $ % & ' ( ) * + , - . /
            0 1 2 3 4 5 6 7 8 9 : ; < = > ?
            (中略)
            € <null/> ‚ ƒ „ ... †
            ‡ ˆ ‰ Š ‹ Œ <null/>
            Ž <null/>
            (中略)
        </order>
    </body>
</letter-order>
```

5.1.4.1. 記述上の注意

ctr_FontConverter の内部処理は文字コードとして Unicode を使用しているため、文字コードが同一かどうかの判定も Unicode で行われることに注意しなければなりません。例えば、CP1252 と ShiftJIS をそれぞれの文字コードで比較すると、ASCII 文字同士および欧州文字と半角カナ文字が同一の文字コードに割り当てられていますが、Unicode の文字コードで見ると欧州文字と半角カナ文字は異なる文字コードに割り当てられているため、同一ではないと判定されます。しかし、表 5-5にある 16 文字はそれぞれの文字セットでは異なる文字コードに割り当てられていますが、Unicode では同じ文字コードに割り当てられているため、同一であると判定されます。SDK に付属している文字順序ファイルの cp1252_JIS_X0201_X0208_012.xlor と cp1252_JIS_X0201_X0208_012_94.xlor では、この問題を避けるために ShiftJIS 側の 16 文字を <null/> に置き換えています。

表 5-5. Unicode で同じ文字コードにマッピングされる文字

グリフ	CP1252 での文字コード	ShiftJIS での文字コード	Unicode での文字コード
´	0xB4	0x814C	U+00B4
¨	0xA8	0x814E	U+00A8
…	0x85	0x8163	U+2026
’	0x91	0x8165	U+2018
’	0x92	0x8166	U+2019
″	0x93	0x8167	U+201C
″	0x94	0x8168	U+201D
±	0xB1	0x817D	U+00B1
×	0xD7	0x817E	U+00D7
÷	0xF7	0x8180	U+00F7
°	0xB0	0x818B	U+00B0
§	0xA7	0x8198	U+00A7
‰	0x89	0x81F1	U+2030
†	0x86	0x81F5	U+2020
‡	0x87	0x81F6	U+2021
¶	0xB6	0x81F7	U+00B6

5.2. PC にインストールされているフォントから作成する

ctr_FontConverter は、PC にインストールされているフォントからフォントリソースを作成することができます。ライセンスされていないフォントを商品に使用してしまう可能性を排除するために、自動フォントリンク機能を無効にした状態でフォントリソースを作成しますので、指定されたフォントに含まれていない文字はフォントリソースには含まれません。

自動フォントリンク機能のため、一般のテキストエディタなどで表示されるときには、フォントに含まれていない文字が別のフォントから補完されている場合があります。Windows のアプリケーション「文字コード表」ならば、フォントに含まれている文字だけを確認することができます。

5.3. 画像データから作成する

ctr_FontConverter は、フォントイメージが描かれている画像データと文字順序ファイルからフォントリソースを作成することができます。ctr_FontConverter の入力に使用可能な画像データは、以下の形式でなければなりません。

表 5-6. ctr_FontConverter が対応している画像データの形式

形式	対応範囲
インデックスカラー BMP	1 ピクセルあたり、1, 4, 8 ビット(2, 16, 256 色)。
ダイレクトカラー BMP	1 ピクセルあたり、24 ビット(RGB8)、32 ビット(RGBA8)。
ColorMap 形式の TGA	1 ピクセルあたり、インデックスは 8, 16 ビット(256, 65536 色)、パレットは 24 ビット(RGB8)、32 ビット(RGBA8)。ランレングス圧縮されたデータにも対応しています。
TrueColor 形式の TGA	1 ピクセルあたり、24 ビット(RGB8)、32 ビット(RGBA8)。ランレングス圧縮されたデータにも対応しています。

画像データからの入力オプションで指定されたカラーフォーマットと画像データの形式により、以下のように変換処理が行われます。インテンシティ形式のフォントでは、画像データの輝度値が逆転した値に変換されることに注意してください。つまり、黒に近いピクセルほど輝度値が高く、白に近いピクセルほど輝度値が低くなります。

表 5-7. カラーフォーマットと変換時に行われる処理

カラー	変換処理
A4	インデックスカラーの場合、インデックス値を 4 ビットに変換します。 ダイレクトカラーの場合、RGB 成分の平均値を 16 階調に変換します。
A8	インデックスカラーの場合、インデックス値を 8 ビットに変換します。 ダイレクトカラーの場合、RGB 成分の平均値を 256 階調に変換します。
LA4	インデックスカラーの場合、輝度値にはインデックス値を 4 ビットに変換したものを使用します。 ダイレクトカラーの場合、輝度値には RGB 成分の平均値を 16 階調に変換したものを使用します。 画像データのカラーフォーマットに関係なく、アルファ値にはピクセルのアルファ成分を 4 ビットに変換したものを使用します。
LA8	インデックスカラーの場合、輝度値にはインデックス値を 8 ビットに変換したものを使用します。 ダイレクトカラーの場合、輝度値には RGB 成分の平均値を 256 階調に変換したものを使用します。 画像データのカラーフォーマットに関係なく、アルファ値にはピクセルのアルファ成分を 8 ビットに変換したものを使用します。
RGB565	RGB 成分それぞれが 5, 6, 5 ビットになるように変換し、アルファ成分は使用しません。
RGB5A1	RGB 成分それぞれが 5 ビットになるように変換し、アルファ成分は上位 1 ビットのみ使用します。
RGBA4	RGBA 成分それぞれが 4 ビットになるように変換します。
RGB8	RGB 成分それぞれが 8 ビットになるように変換し、アルファ成分は使用しません。
RGBA8	RGBA 成分それぞれが 8 ビットになるように変換します。

線形補間対策処理

FONT ライブラリによる文字の描画では、グリフをポリゴンに貼り付けたテクスチャとして表示するため、拡大時にはハードウェアによるテクスチャの線形補間機能を使用することができます。しかし、フォントのフォーマットがアルファチャンネルを持つ場合、テクスチャの線形補間機能を使うとアルファ値が 0 のピクセルのカラーが参照され、意図しない表示になることがあります。

ctr_FontConverter では、フォントリソースへの変換時に線形補間対策処理を適用することができます。線形補間対策処理では、完全透明(アルファ値が 0)であるピクセルのカラーを、そのピクセルと隣り合う 8 つのピクセルのうち、完全透明ではない(アルファ値が 0 ではない)ピクセルのカラーを平均したものに書き換えます。アルファ値は書き換えません。このようにグリフを補正することで、線形補間機能による意図しない表示を解消することができます。

5.3.1. ブロック

ブロックとは、グリフイメージが描かれているセルと文字幅を示す幅線領域を含む画像データの部分領域のことです。文字順序ファイルに記述された 1 文字が 1 ブロックに対応し、入力に使用する画像データには、このブロックが隙間なく敷き詰められていなければなりません。

入力に使用する画像データは、以下のように制限されています。

ブロックの幅 × ブロックの横方向の数 = 画像の幅

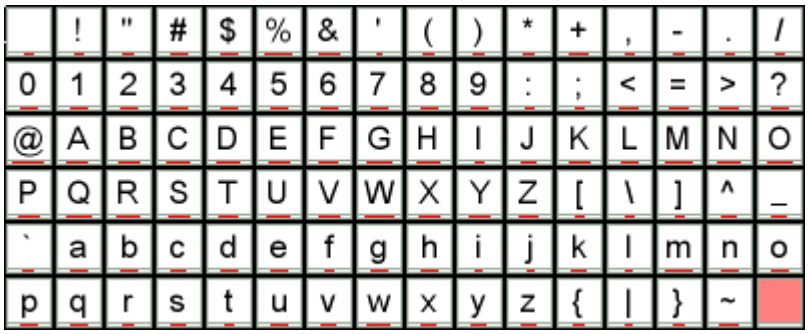
ブロックの高さ × ブロックの縦方向の数 = 画像の高さ

つまり、画像中にブロックは隙間なく並び、余白があってはいけません。

実際には、文字順序ファイルによりブロックの横(縦)方向の数が、画像データにより画像の幅(高さ)が決定し、そこからブロックの幅(高さ)が計算されます。そして、このブロックの幅(高さ)が整数でなければ入力に使用することができません。

以下の図は画像データの例です。

図 5-5. 画像データの例(ブロック数 16×6)



ブロック内には、グリフイメージの描画エリアであるセルがあり、この領域内にグリフイメージを描画します。セルの下には、1 ピクセルの余白エリアを挟んで高さ 1 ピクセルの幅線領域があり、幅線は文字幅と、グリフイメージの相対的な位置を定義します。セルと幅線領域の周囲 1 ピクセルが余白エリア、その周囲 1 ピクセルがグリッドエリアとなっており、隣接するブロックとのグリッドエリアの境がブロックの境となります。変換時にグリッドエリアのチェックは行われませんが、余白エリアが単一色でなければグリフイメージがはみ出していると判断されます。

幅線領域の高さや余白エリア、グリッドエリアのピクセル数は固定であるため、セルの大きさは常に、

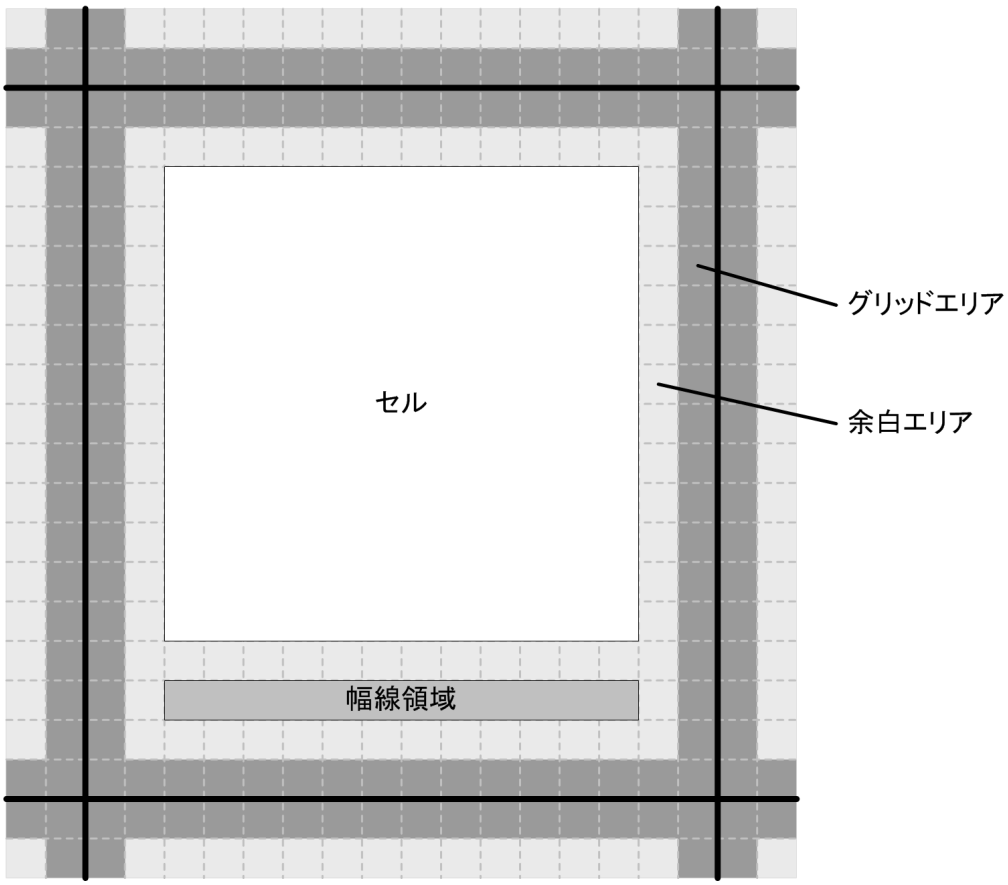
セルの幅 = ブロックの幅 – 4 ピクセル

セルの高さ = ブロックの高さ – 6 ピクセル

となります。

以下の図はブロックの模式図です。

図 5-6. ブロックの模式図



5.3.2. セルと幅線領域

セルの内部にはグリフイメージのみを配置します。ctr_FontConverter はセル内に存在する、アルファ値が 0 ではない、または色が白(255, 255, 255)ではないピクセルを含む最小の矩形を検出し、それをグリフイメージとして扱います。そのためセルのサイズが大きくてもグリフイメージの周囲の白色透明部分は無視され、出力されるフォントに影響を与えません。なお、画像データがアルファチャンネルを含まない場合は、色が白ではないピクセルを含む最小の矩形をグリフイメージとして扱います。

幅線領域には、幅線と呼ばれる 1 つの線分のみを描きます。幅線は文字幅と文字の左右(前後)に空けるスペースの幅を規定します。幅線の横幅がそのまま文字幅となり、線分が途中で途切れて 2 つになっていたりするとエラーとなります。文字幅はグリフイメージの幅より小さくすることができ、その場合は 3DS 上で前の文字と重なって表示されることになります。幅線との位置関係が同じであれば、セルの中でグリフイメージを左右に移動しても出力されるフォントは変化しません。

グリフイメージのサイズが 0 で幅線の幅も 0 である場合、またはセル内が単一色で塗りつぶされていて、幅線の幅が 0 の場合、このグリフは出力されません。これを利用して、文字フィルタファイルを使用せずに出力するグリフを制御することもできます。逆に、文字フィルタファイルでは出力指定されているのにグリフが出力に渡されない場合は、意図しない文字の抜け落ちの可能性があるため警告が表示されます。

5.3.2.1. 配置情報

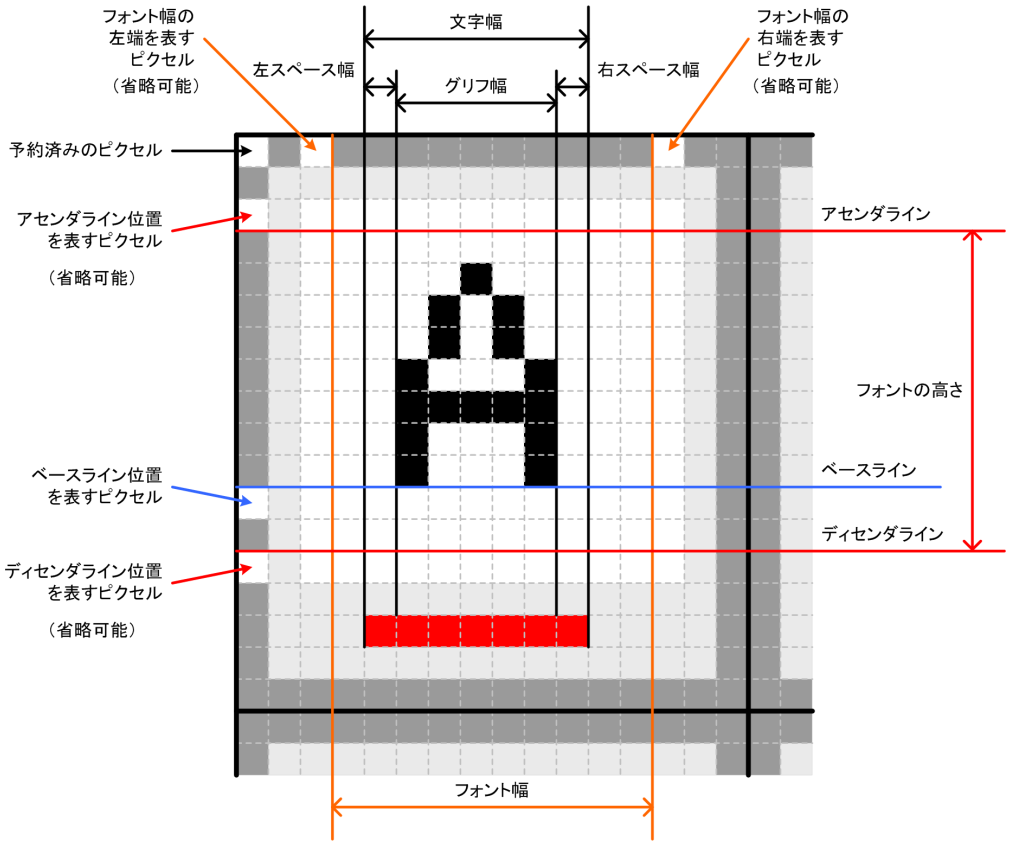
画像データの一番左上のブロックには、文字列を描画するときに文字列の配置の基準となる値を指定する 1 ピクセルの点(配置情報)がいくつか含まれます。配置の基準となる値とはベースライン、アセンダライン、ディセンダライン、フォント幅の 4 つで、この 4 つを 5 つの点で指定します。これらの値はフォント全体で共通となり、文字ごとに指定することはできません。

すべての点はグリッドエリアに存在し、グリッドを描画している場合は白(255, 255, 255)で、グリッドを描画していない場合は

グリッド色で描画します。ベースライン位置を表すピクセルは必ず存在しなければなりませんが、残りの 4 点は省略することができます。

画像の一番左上の 1 ピクセルは将来の用途のために予約されており、必ず白色(255, 255, 255)でなければなりません。また、この 1 ピクセルは抜き色としても使用しています。アルファ値が入力されている場合は、アルファ値を含む色として抜き色を判定します。

図 5-7. ブロックの配置情報



ベースライン

ベースライン位置を表す点はグリッドエリアの左側領域に描画します。このピクセルの上端がベースライン位置となります。つまり、ベースラインはピクセルとピクセルの間に存在します。ベースラインは異なるフォントを混ぜて使用する場合や拡大縮小を行う場合の上下方向の基準位置となります。

アセンダライン、ディセンダライン

アセンダライン位置と、ディセンダライン位置を表す点もグリッドエリアの左側領域に描画します。アセンダラインとディセンダラインも、ベースラインと同様にピクセルとピクセルの間に存在し、アセンダラインはアセンダライン位置を表すピクセルの下端が、ディセンダラインはディセンダライン位置を表すピクセルの上端がそれぞれの位置になります。アセンダラインは文字列の上端として、ディセンダラインは文字列の下端として扱われます。また、アセンダラインとディセンダライン間の距離をフォントの高さと呼び、縦方向に拡大縮小する場合の基準サイズとなります。

アセンダラインは必ずベースラインの上方に位置し、ディセンダラインは必ずベースラインの下方に位置します。また、ディセンダライン位置を表すピクセルはベースライン位置を表すピクセルと重ねることができ、この場合ディセントは 0 になります。

アセンダライン位置とディセンダライン位置を表す 2 点は省略することができ、点の数に応じて表 5-8のように解釈されます。アセンダライン位置とディセンダライン位置を指定しなかった場合は、セルの上端がアセンダライン、セルの下端がディ

センダラインとして扱われます。

ctr_FontConverter で画像データを出力すると、アセンダライン位置とディセンダライン位置を表すピクセルが描かれる場合と描かれない場合とがあります。Windows フォントを入力とする変換では常に描かれず、フォントリソースを入力とする変換では常に描かれます。画像データを入力とする場合は、入力された画像にアセンダライン位置、ディセンダライン位置を表すピクセルが含まれていれば描かれ、含まれていなければ描かれません。アセンダライン位置、ディセンダライン位置を表すピクセルが描かれるときに、セルのサイズがこれらのピクセルを表現するのに小さすぎる場合はセルのサイズが自動的に拡大されます。

表 5-8. 点の数と解釈

点の数	上から 1 つ目の点	上から 2 つ目の点	上から 3 つ目の点
0	エラー		
1	ベースライン	–	
2	アセンダライン	ベースラインとディセンダライン	–
3	アセンダライン	ベースライン	ディセンダライン
4 以上	エラー		

フォント幅

フォント幅を表す 2 つの点はグリッドエリアの上側領域に描画します。2 つの点がそれぞれ幅の左端と右端を表し、2 つのピクセルに挟まれるピクセル数がフォント幅となります。挟まれるピクセル数が同じであれば点の左右位置は影響しません。フォント幅を表す点を省略した場合はセルの幅がフォント幅として扱われます。省略しない場合は必ず 2 点存在しなければなりません。フォント幅は横方向に拡大縮小するときや、タブ幅の基準サイズとなります。

ctr_FontConverter で画像データを出力すると、フォント幅を表すピクセルが描かれる場合と描かれない場合とがあります。Windows フォントを入力とする変換では常に描かれず、フォントリソースを入力とする変換では常に描かれます。画像データを入力とする場合は、入力された画像にフォント幅を表すピクセルが含まれていれば描かれ、含まれていなければ描かれません。フォント幅を表すピクセルが描かれるときに、セルサイズがこれらのピクセルを表現するのに小さすぎる場合はセルのサイズが自動的に拡大されます。

5.4. 既存のフォントリソースから作成する

bcfnt や bcfna からも、フォントリソースの作成や画像データの出力が可能です。フォントリソースに含まれている文字のフィルタリングや調査、グリフイメージのチェックや編集などに利用することができます。

> **注意：** 元となるフォントリソースに含まれているフォントのライセンスに注意してください。

更新履歴

Version 1.1 2015-11-05

変更

- 3.2. ResFont クラス
  - アライメント制約の定義名を追記しました。

Version 1.0 2014-09-04

追加/変更

- 初版