Autodesk®
Scaleform®

DrawText API

Scaleform 4.3 の DrawText API についての資料です。この API は GFx::Movie や ActionScript サンドボックスではなく、C++で操作するテキスト レンダリングとテキスト フォーマットに使用されます。

著者: Artem Bolgar
バージョン: 2.0
最終更新日: 2010 年 7 月 30 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

Autodesk Scaleform の連絡先:

ドキュメント	DrawText API
住所	Scaleform Corporation 6305 Ivy Lane, Suite 310

	Greenbelt, MD 20770, USA
ホームページ	www.scaleform.com
電子メール	info@scaleform.com
電話	(301) 446-3200
Fax	(301) 446-3199

目次

1. はじめに

Scaleform SDK インプリメンテーションには、フォント レンダリングとテキスト フォーマットを行う高性能エンジンが含まれていて、主に Flash®ベースの UI として表示されるテキストのレンダリングに使用されます。Scaleform のテキスト レンダリングは、画面のピクセルに沿ったグリフのオンザフライのラスタライズやキャッシング、段落の配置を伴うスクロール可能なテキスト フィールド、さらにシステム フォント プロバイダか、SWF/GFX 埋め込みファイル データのどちらかから取得したフォントを含む HTML タグ形式のリッチ テキストなど、多くの高度な機能をサポートします。

ほとんどの場合、Scaleform のテキスト レンダリング システムは、Flash ファイルを再生するときに自動的に使用されます。これらのファイルには、Flash Studio でデザイナーが目視で配置したテキスト フィールドが含まれています。さらに制御するため、テキスト フォーマットは、TextField や TextFormat といった ActionScript (AS) クラスを通して公開されます。これらのクラスはテキストを使ったインターフェイス化の推奨される方法を提供します。これらの API を適切に設定していれば、C++ API をこの目的で使用する必要はほとんどありません。

ただし、テキスト レンダリングのために ActionScript を使用するのが不便な状況や、関連するオーバーヘッドが望ましくない場合があります。画面上で動いているアバター上にネーム札を表示したり、レーダー上のアイテムの横にテキスト ラベルを表示することは、そのようなアイテムに本格的な Flash UI を持つ余裕のないゲームの場合、C++を使ったほうが効率が良い場合があります。さらに、Scaleform ベースのメニュー システムと、3D エンジンで表示した HUD とを組み合わせて使っているゲームでも、同じフォント システムを、その両方で使用することが望まれます。

Scaleform SDK は C++で操作する DrawText API を 導入して、ムービー ビューのサンドボックス外で Scaleform フォント レンダリング エンジンやテキスト フォーマット エンジンを公開しています。この新たなテキスト API は、Scaleform プレイヤーが他に使うのと同じ Render::Renderer インターフェイスを使用しますが、GFx::Movie オブジェクトを作成する必要がなく、開発者は関連する ActionScript オーバーヘッドなしで、ビューポート内で直接テキスト フィールドを配置し、操作することができます。

DrawText API では、フォント ソースが 2 つ使えます。システム フォントと SWF/GFX ファイルからロードされたフォントです。1 つ目の場合、システム フォント プロバイダを使用する必要があります。2 つ目の場合には、フォントを持つ SWF ファイルを GFx::MovieDef としてロードする (GFx::Loader::CreateMovie メソッドを使って) 必要があります。そうすれば作成された GFx::MovieDef を、パラメータとして GFx::DrawTextManager コンストラクタに渡すことができます。

テキスト レンダリング機能を提供する 2 つの C++クラスは、GFx::DrawTextManager と GFx::DrawText で、その使用法の概略は下記のとおりです。詳細は後で説明します。

- *GFx::DrawTextManager* – DrawText クラスのインスタンスを作成します。テキスト レンダリングとビューポートの初期化、フォントの構成、テキスト レンダリングの開始/中止に使用されます。

- *GFx::DrawText* – 画面上に個別の矩形テキスト フィールドを表示して、テキストのフォーマットとレンダリングの機能を公開します。このクラスは、Flash HTML タグを解析するか、補助的なフォーマット データを持つプレーン テキストを 使用するかのいずれかを行うことできます。テキスト フィールド属性は SetColor、SetFont、SetFontStyle などのメンバー関数で変更することができます。

1.1. GFx::DrawTextManager

DrawTextManager クラス インスタンスは、DrawText クラスのインスタンスを管理し、テキスト範囲の作成、表示、計測を行います。このクラスは GFx::StateBag クラスを拡張して、その機能を継承しています。

DrawTextManager クラス インスタンスの作成にはいくつか方法があります:

1. `DrawTextManager();`

デフォルトのコンストラクタです。フォント キャッシュ マネージャ、リソース ライブラリ、ログの内部インスタンスを作成します。システム フォント プロバイダをフォント ソースとして使用する必要があります (SetFontProvider メソッド)。

例:

```
Ptr<DrawTextManager> pdm = *new DrawTextManager();
```

2. `DrawTextManager(MovieDef* pmovieDef);`

このコンストラクタは、パラメータとして GFx::MovieDef へのポインタを使用します。DrawTextManager のインスタンスは、フォント キャッシュ マネージャ、フォント プロバイダなどを含めて、渡された MovieDef インスタンスからすべてのステートを継承します。その MovieDef インスタンスに含まれているすべてのフォントは、DrawTextManager によって (そしてこの DrawTextManager に作成されたすべての DrawText インスタンスによって) アクセス可能になります。

例:

```
Ptr<GFx::MovieDef> pmovieDef = *mLoader.CreateMovie("drawtext_fonts.swf",
Loader::LoadAll);
Ptr<GFx::DrawTextManager> pDm2 = *new DrawTextManager(pmovieDef);
```

3. `DrawTextManager(GFx::Loader* ploader);`

このコンストラクタはパラメータとして GFx::Loader へのポインタを使用し、フォントライブラリを含み、このローダに設定された全てのステートを継承します。

例:

```
GFx::Loader mLoader;
..
Ptr<GFx::DrawTextManager> pDm1 = *new DrawTextManager(&mLoader);
```

1.1.1. GFx::DrawText の作成

DrawTextManager クラスで DrawText インスタンスを作成するには、複数のメソッドがあります。DrawTextManager の 1 つのインスタンスを使って、必要なだけ DrawText インスタンスを作成することができます。

- `DrawText* CreateText(const char* putf8Str, const RectF& viewRect, const TextParams* ptxtParams = NULL, unsigned depth = ~0u);`
- `DrawText* CreateText(const wchar_t* pwstr, const RectF& viewRect, const TextParams* ptxtParams = NULL, unsigned depth = ~0u);`
- `DrawText* CreateText(const String& str, const RectF& viewRect, const TextParams* ptxtParams = NULL, unsigned depth = ~0u);`

上記のメソッドは、プレーン テキストを使って DrawText インスタンスを作成します (上から順に、NULL で終了する UTF-8 文字列、NULL で終了する Unicode/UCS-2、最後に Scaleform::String)。
`putf8str` ：NULL で終了する UTF-8 文字列を指定します。
`pwstr` ：NULL で終了する Unicode/UCS-2 文字列を指定します。
`str`：Scaleform::String を指定します。

`viewRect` ：ビューポートの左上角に対する、テキストの表示エリア(ピクセル数)の座標を指定します (BeginDisplay を参照してください)。

`ptxtParams`：オプションのパラメータです。新規に作成されたテキストのパラメータを指定します。TextParams の構造は以下のとおりです:

```
struct TextParams
{
    Color                  TextColor;
    DrawText::Alignment  HAlignment;
    DrawText::VAlignment VAlignment;
    DrawText::FontStyle  FontStyle;
    float                FontSize;
    String              FontName;
    bool                 Underline;
```

```
    bool                Multiline;
    bool                WordWrap;
};
```

TextColor： アルファ チャンネルを含め、テキストの色を指定します。

HAlignment ：水平方向の配置を指定します。可能な値は、DrawText::Align_Left (デフォルト)、DrawText::Align_Right、DrawText::Align_Center、DrawText::Align_Justify です。

VAlignment：垂直方向の配置を指定します。可能な値は DrawText::VAlign_Top (デフォルト)、DrawText::VAlign_Bottom、DrawText::VAlign_Center です。

FontStyle ：ボールド、イタリック、標準、またはボールドイタリックなどのフォント スタイルを指定します。可能な値は DrawText::Normal、DrawText::Bold、DrawText::Italic、DrawText::BoldItalic (さらに同意語として DrawText::ItalicBold) です。

FontSize ：フォントのサイズを指定します (ピクセル数)。小数の場合もあります。

FontName :“Arial”、“Times New Roman”などのフォント名を指定します。名前に“Bold”や“Italic”などの接尾辞を付けないでください。FontStyle で指定してください。

Underline ：下線を使用するかしないかを指定します。

Multiline：テキストの複数行/単一行モードを切り替えます。

WordWrap ：テキストの折り返しのオン/オフを切り替えます。これは複数行のテキスト ボックスに限って有効です。

ptxtParams オプション パラメータが指定されていない場合、DrawTextManager はデフォルトのテキスト パラメータを使用します。以下のメソッドを使って、これらのデフォルトのテキスト パラメータの設定/取得が可能です:

```
void SetDefaultTextParams(const TextParams& params);
const TextParams& GetDefaultTextParams() const;
```

以下の DrawTextManager のメソッド CreateHtmlText は、上記の CreateText メソッドに似ていますが、HTML から DrawText インスタンスを作成します:

```
    // 指定されたHTMLを使ってGFx::DrawTextオブジェクトを作成し初期化する。
DrawText* CreateHtmlText(const char* putf8Str, const RectF& viewRect,
                         const TextParams* ptxtParams = NULL, unsigned depth = ~0u);
    DrawText* CreateHtmlText(const wchar_t* pwstr, const RectF& viewRect,
                         const TextParams* ptxtParams = NULL, unsigned depth = ~0u);
    DrawText* CreateHtmlText(const String& str, const RectF& viewRect,
```

```
                        const TextParams* ptxtParams = NULL, unsigned depth =
~0u);
```

例:

```
    // パラメータ付きのプレーン テキストを使ったGFx::DrawTextオブジェクトの作成
    String str("String No 2");
    DrawTextManager::TextParams params;
    params.FontName   = "Symbol";
    params.FontSize   = 30;
    params.FontStyle  = DrawText::Italic;
    params.Multiline  = false;
    params.HAlignment = DrawText::Align_Right;
    params.VAlignment = DrawText::VAlign_Bottom;

    Ptr<DrawText> ptxt;
    ptxt = *pdm->CreateText(str, RectF(20, 300, 400, 700), &params);

    // HTMLからGFx::DrawTextオブジェクトを作成
    Ptr<DrawText> ptxt;
    ptxt = *pdm->CreateHtmlText("<p><FONT size='20'>AB
                                 <b>singleline</b><i> CD</i>O",
                                RectF(20, 300, 400, 700));
```

1.1.2. テキストのレンダリング

テキスト レンダリングは GFx::Movie のレンダリングと同様の方法で行われます。テキスト レンダリングでは DisplayHandle（GFx::DrawTextManager）を取得しビューポート（DrawTextManager::SetViewport）を設定する必要があります。Viewport の詳細については GFx のドキュメントを参照してください。

1.1.3. テキスト範囲の計測

DrawTextManager は、テキストの表示に必要なテキスト矩形のサイズを計測する機能を備えています:

```
    SizeF GetTextExtent(const char* putf8Str, float width = 0,
                        const TextParams* ptxtParams = 0);
    SizeF GetTextExtent(const wchar_t* pwstr, float width = 0,
                        const TextParams* ptxtParams = 0);
    SizeF GetTextExtent(const String& str, float width = 0,
                        const TextParams* ptxtParams = 0);

    SizeF GetHtmlTextExtent(const char* putf8Str, float width = 0,
```

```
                            const TextParams* ptxtParams = 0);
    SizeF GetHtmlTextExtent(const wchar_t* pwstr, float width = 0,
                            const TextParams* ptxtParams = 0);
    SizeF GetHtmlTextExtent(const String& str, float width = 0,
                            const TextParams* ptxtParams = 0);
```

GetTextExtent メソッドはプレーン テキストを使って、さらにオプションで TextParams と、希望のテキスト幅を使って、テキスト矩形のサイズを計算します。

GetHtmlTextExtent メソッドは HTML テキストを使い、さらにオプションで TextParams と希望のテキスト幅を使って、テキスト矩形のサイズを計算します。

オプションの'width'パラメータは、複数行でテキストの折り返しが使用される場合に、テキスト矩形の希望の幅を指定します。この場合、テキスト矩形の高さだけが計算されます。

オプションの ptxtParams はテキスト パラメータを指定します。これらのパラメータはテキスト サイズの計測中に使用されます。指定されていない場合、デフォルトのテキスト パラメータが使用されます (SetDefaultTextParams / GetDefaultTextParams を参照してください)。HTML バージョンについては、この `ptxtParams` パラメータはデフォルトのテキスト パラメータ セットを動作させるので、解析された HTML のすべてのスタイルには、上記の'`ptxtParams`'のスタイルが実質的に適用されます。

例:

```
    // プレーン テキスト範囲
    String str("String No 2");
    DrawTextManager::TextParams params;
    params.FontName   = "Symbol";
    params.FontSize   = 30;
    params.FontStyle  = DrawText::Italic;
    params.Multiline  = false;
    params.HAlignment = DrawText::Align_Right;
    params.VAlignment = DrawText::VAlign_Bottom;

    SizeF sz = pdm->GetTextExtent(str, 0, params);

    params.WordWrap  = true;
    params.Multiline = true;
    sz = pdm->GetTextExtent(str, 120, params);

    // HTMLテキスト範囲
    const wchar_t* html =
        L"<p><FONT size='20'>AB <b>singleline</b><i> CD</i>O";
    sz = pdm->GetHtmlTextExtent(html);

    sz = pdm->GetHtmlTextExtent(html, 150);
```

1.2. GFx::DrawText

DrawText クラスはテキストの設定、HTML の解析、テキストのフォーマットとレンダリングを行う機能を備えています。

1.2.1. プレーン テキスト メソッドの設定と取得

SetText メソッドは UTF-8、UCS-2、または Scaleform::String テキスト値をテキスト オブジェクトに設定します。オプションのパラメータ 'lengthInBytes'は UTF-8 文字列のバイト数を指定し、'lengthInChars'はワイド文字の文字列の文字数を指定します。これらのパラメータが指定されていない場合、UTF-8 と UCS-2 文字列は、NULL で終了する必要があります。

```
void SetText(const char* putf8Str, UPInt lengthInBytes = UPInt(-1));
void SetText(const wchar_t* pstr,  UPInt lengthInChars = UPInt(-1));
void SetText(const String& str);
```

GetText メソッドは現在設定されている UTF-8 形式のテキストを返します。このメソッドはプレーンテキスト値を返します。HTML が使用されている場合でも、すべての HTML タグを削除した状態で文字列を返します。

```
String GetText() const;
```

1.2.2. HTML テキスト メソッドの設定と取得

SetHtmlText メソッドは UTF-8、UCS-2、または Scaleform::String でエンコードされた HTML を解析して、その HTML テキストでテキスト オブジェクトを初期化します。オプションのパラメータ 'lengthInBytes'は UTF-8 文字列のバイト数を指定し、'lengthInChars'はワイド文字の文字列の文字数を指定します。
これらのパラメータが指定されていない場合、UTF-8 と UCS-2 文字列は、NULL で終了する必要があります。

```
void SetHtmlText(const char* putf8Str, UPInt lengthInBytes = UPInt(-1));
void SetHtmlText(const wchar_t* pstr,  UPInt lengthInChars = UPInt(-1));
void SetHtmlText(const String& str);
```

GetHtmlText メソッドは現在設定されている HTML 形式のテキストを返します。SetColor、SetFont などのメソッドの呼び出しによって、プレーン テキストがフォーマットの設定と併用される場合、そのテキストはこのメソッドによって適切な HTML 形式に変換されます。

```
String GetHtmlText() const;
```

1.2.3. テキスト フォーマットを設定するメソッド

テキスト フォーマットを設定し取得するメソッドはいくつかあります。

```
void SetColor(Color c, UPInt startPos = 0, UPInt endPos = UPInt(-1));
```
色 (R、G、B、A) をテキスト全体、または [startPos..endPos] 区間のテキストの一部に設定します。‘startPos’と ‘endPos’パラメータは両方ともオプションです。

```
void SetFont (const char* pfontName, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```
フォントをテキスト全体、または [startPos..endPos] 区間のテキストの一部に設定します。‘startPos’と ‘endPos’パラメータは両方ともオプションです。

```
void SetFontSize(float fontSize, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```
フォント サイズをテキスト全体、または [startPos..endPos] 区間のテキストの一部に設定します。‘startPos’と ‘endPos’パラメータは両方ともオプションです。

```
void SetFontStyle(FontStyle, UPInt startPos = 0, UPInt endPos = UPInt(-1));

    enum FontStyle
    {
        Normal,
        Bold,
        Italic,
        BoldItalic,
        ItalicBold = BoldItalic
    };
```
フォント スタイルをテキスト全体、または [startPos..endPos] 区間のテキストの一部に設定します。‘startPos’と ‘endPos’パラメータは両方ともオプションです。

```
void SetUnderline(bool underline, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```
下線をテキスト全体、または [startPos..endPos] 区間のテキストの一部に設定、または設定解除します。‘startPos’と ‘endPos’パラメータは両方ともオプションです。

```
void SetMultiline(bool multiline);
```
複数行 (パラメータ ‘multiline’が true に設定されます)、または単一行 (false) のテキスト タイプを設定します。

```
bool IsMultiline() const;
```
テキストが複数行の場合は 'true' を、それ以外は ‘false’ を返します。

```
void SetWordWrap(bool wordWrap);
```
テキストの折り返しのオン/オフを切り替えます。

```
bool Wrap() const;
```
テキストの折り返しのステートを返します。

1.2.4. テキストの配置メソッド

Alignment と VAlignment の種類の定義:

```
enum Alignment
{
    Align_Left,
    Align_Default = Align_Left,
    Align_Right,
    Align_Center,
    Align_Justify
};
enum VAlignment
{
    VAlign_Top,
    VAlign_Default = VAlign_Top,
    VAlign_Center,
    VAlign_Bottom
};
```
テキストを配置するメソッドは以下のとおりです:

```
void SetAlignment(Alignment);
```
水平方向のテキスト配置 (右揃え、左揃え、中央揃え) を設定します。

```
Alignment   GetAlignment() const;
```
水平方向のテキスト配置 (右揃え、左揃え、中央揃え) を返します。

```
void SetVAlignment(VAlignment);
```
垂直方向のテキスト配置 (上揃え、下揃え、中央揃え) を設定します。

```
VAlignment  GetVAlignment() const;
```
垂直方向のテキスト配置 (上揃え、下揃え、中央揃え) を返します。

1.2.5. テキストの位置と向きの変更

DrawText クラスは、テキスト オブジェクトの配置と向きを設定するさまざまなメソッドを備えています。

```
void SetRect(const RectF& viewRect);
```
表示矩形を設定します。座標はピクセル数で表します。

```
RectF  GetRect() const;
```
現在使用されている表示矩形を返します。座標はピクセル数で表します。

```
void SetMatrix(const Matrix& matrix);
```
テキスト オブジェクトに変換マトリックスを設定します。

```
const Matrix GetMatrix() const;
```
現在使用されている変換マトリックスを返します。

```
void SetCxform(const Cxform& cx);
```
テキスト オブジェクトに色変換マトリックスを設定します。

```
const Cxform& GetCxform() const;
```
現在使用されている色変換マトリックスを返します。

2. DrawText サンプル コードの概要

この章では、Scaleform に含まれている DrawText API サンプル インプリメンテーションについて説明します。DrawText API は以前から Scaleform SDK に含まれており、メモリやパフォーマンス条件のオーバーヘッドなしで、画面にシンプル テキスト オブジェクトを簡単に描画できるようにしています。このサンプル コードはテキストを描画するために DrawText API で使用されるクラスを記述しており、開発者のテキスト エンジンに簡単に組み込んでテキストを表示することができます。

基本ステップは、ウィンドウの設定、ムービー データのロード、および、テキストのキャプチャとレンダリングに必要なディスプレイ ハンドルの追加で構成されます。

FxPlayerApp クラスは、ウィンドウの設定、ムービーのロード、およびテキストのレンダリングにそれぞれ必要なすべてのプロパティと関数を定義したプレーヤー アプリケーション クラスです。FxPlayerApp は、ウィンドウ、レンダー スレッド、およびグラフィクスのそれぞれの作成と初期化が行われるベース クラスである FxPlayerAppBase クラスを継承します。

FxPlayerApp の OnInit メソッドは、DrawText を初期化し、テキストを作成し、さまざまなテキスト変換を適用します。OnUpdateFrame メソッドは、変換マトリックスをテキストに適用して、アニメーションを生成します。

```
class   FxPlayerApp : public FxPlayerAppBase
{
    public:
    FxPlayerApp();

    virtual bool            OnInit(Platform::ViewConfig& config);
    virtual void            OnUpdateFrame(bool needRepaint);

    Ptr<GFx::DrawTextManager> pDm1, pDm2;
    Ptr<GFx::DrawText>          ptxt11, ptxt12,  ptxt21, ptxt22, ptxtImg,
                                pblurTxt, pglowTxt, pdropShTxt, pblurglowTxt;
    float   Angle;
    UInt32  Color;
};
```

この DrawText サンプルは、DrawTextManager クラス インターフェイスの 2 つの異なるインスタンスを使って、テキストをレンダリングする方法について記述しています。

方法 1

この例では DrawTextManager インスタンスは、GFx::Loader へのポインタを使って作成され、そのローダのすべてのステートを継承します。

```
pDm1 = *new DrawTextManager(&mLoader);
```

ここで、表示テキストにはシステム フォントを使用することにし、GFx::FontProvider を使います。FontProvider は FxPlayerAppBase::OnInit 内で作成されます。

```
pDm1->SetFontProvider(mLoader.GetFontProvider());
```

この API で述べているように、プレーン テキスト、または HTML テキストのいずれかを使った DrawText インスタンスの作成方法はいくつかあります。この例では、DrawText インスタンスはプレーン テキストと Scaleform::String を使って、作成されています。

```
// プレーン テキストとデフォルトのテキスト パラメータを使ってテキストを作成
DrawTextManager::TextParams defParams =
     pDm1->GetDefaultTextParams();
defParams.TextColor = Color(0xF0, 0, 0, 0xFF); // red, alpha = 255
defParams.FontName  = "Arial";
defParams.FontSize  = 16;
pDm1->SetDefaultTextParams(defParams);
```

テキストは CreateText メソッドを呼び出して作成され、前の SetDefaultTextParams 呼び出しで設定したデフォルトのテキスト パラメータを使用します。

```
ptxt11 = *pDm1->CreateText ("Plain text, red, Arial,
                                    16pts",RectF(20, 20, 500, 400));
```

DrawTextManager クラスの GetTextExtent メソッドはテキスト矩形のサイズを計測します。DrawText クラスは、フォントのフォーマット、テキストの整列、またはテキスト オブジェクト位置の変更など、テキストの操作に便利です。

```
// StringとTextParamsを使ってテキストを作成
String str(
   "Scaleform is a light-weight high-performance rich media vector
    graphics and user interface (UI) engine.");
GFx::DrawTextManager::TextParams params;
params.FontName   = "Arial";
params.FontSize   = 14;
params.FontStyle  = DrawText::Italic;
params.Multiline  = true;
params.WordWrap   = true;
params.HAlignment = DrawText::Align_Justify;
sz = pDm1->GetTextExtent(str, 200, &params);
ptxt12 = *pDm1->CreateText(str, RectF(200, 300, sz), &params);
ptxt12->SetColor(Render::Color(0, 0, 255, 130), 0, 1);
```

システム フォントを使ってテキストを作成したら、DrawText のビューポートを設定し、現在の DrawText ステートをキャプチャし、DrawText DisplayHandle を RenderThread に追加します。

```
Render::Size<unsigned> viewSize = GetViewSize();
Render::Viewport dmViewport(viewSize.Width, viewSize.Height,
        int(viewSize.Width * GetSafeArea().Width),
        int(viewSize.Height * GetSafeArea().Height),
        int(viewSize.Width - 2 * viewSize.Width * GetSafeArea().Width),
        int(viewSize.Height - 2 * viewSize.Height * GetSafeArea().Height));
pDm1->SetViewport(dmViewport);
pDm1->Capture();
pRenderThread->AddDisplayHandle(pDm1->GetDisplayHandle(),
                                FxRenderThread::DHCAT_Overlay, false);
```

方法 2

この例は、GFx::MovieDef ポインタを使用して、MovieDef に含まれるフォントを共有する DrawTextManger のインスタンスを作成します。このテキストのフォントは、MovieDef にロードされた SWF ファイルから取得されます。Scaleform の詳細やムービー オブジェクトのロードについては、Scaleform Integration Tutorial (日本語版) を参照してください。

```
Ptr<MovieDef> pmovieDef = *mLoader.CreateMovie("drawtext_fonts.swf",
                                               Loader::LoadAll);
pDm2 = *new DrawTextManager(pmovieDef);
```

前の例とは異なり、プレーン テキストの代わりに HTML テキストを使ってテキストを表示します (ただしもちろん、前の例で HTML を使用することもできます)。テキスト矩形のサイズは、**DrawTextManager::GetHtmlExtent** メソッドを使って計算されます。

```
// MovieDefのフォントを使って、HTMLテキストを作成
const wchar_t* html = L"<P>o123 <FONT FACE=\"Times New Roman\" SIZE
                       =\"140\">"L"A<b><i><FONT
        COLOR='#3484AA'>б</FONT></i>рак</b>адабрА!</FONT></P>"
        L"<P><FONT FACE='Arial Unicode MS'>Hànyǔ; 华语/華語</FONT></P>"
        L"<P><FONT FACE='Batang'>한국어/조선말</FONT></P>"
        L"<P><FONT FACE='Symbol'>Privet!</FONT></P>";

GFx::DrawTextManager::TextParams defParams2 = pDm2->GetDefaultTextParams();
defParams2.TextColor = Color(0xF0,0,0,0xFF); //red,alpha = 255
defParams2.Multiline = true;
defParams2.WordWrap  = false;
SizeF htmlSz = pDm2->GetHtmlTextExtent(HtmlText, 0, &defParams2);
ptxt22 = *pDm2->CreateHtmlText(HtmlText, RectF(00, 100, htmlSz), &defParams2);
```

また、テキスト アニメーション (回転やカラー マトリックスの変更) を示すために、テキストを作成する必要もあります:

```
SizeF sz;
  sz = pDm2->GetTextExtent("Animated");
  ptxt21 = *pDm2->CreateText("Animated", RectF(600, 400, sz));
  ptxt21->SetColor(Render::Color(0, 0, 255, 255));
  Angle = 0;
```

および、フィルタ処理:

```
SizeF sz;
Ptr<GFx::DrawText>  pglowTxt
pglowTxt = *pDm2->CreateText("Glow", RectF(800, 350, SizeF(200, 220)));
pglowTxt->SetColor(Render::Color(120, 30, 192, 255));
pglowTxt->SetFontSize(72);
GFx::DrawText::Filter glowF(GFx::DrawText::Filter_Glow);
glowF.Glow.BlurX = glowF.Glow.BlurY = 2;
glowF.Glow.Strength = 1000;
glowF.Glow.Color = Render::Color(0, 0, 0, 255).ToColor32();
pglowTxt->SetFilters(&glowF);
```

DrawText のビューポートを設定し、現在の DrawText ステートをキャプチャし、DrawText DisplayHandle を RenderThread に追加します：

```
pDm2->SetViewport(dmViewport);
pDm2->Capture();
pRenderThread->AddDisplayHandle(pDm2->GetDisplayHandle(),
                                    FxRenderThread::DHCAT_Overlay, false);
```

例として、DrawText の SetMatrix と SetCxform メソッドを使って、作成したテキストを回転し、色を変えます。この処理は FxPlayerApp::OnUpdateFrame 内部で実行します:

```
void FxPlayerApp::OnUpdateFrame( bool needRepaint )
{
    DrawText::Matrix txt21matrix;
    Angle += 1;
    RectF r = ptxt21->GetRect();
    txt21matrix.AppendTranslation(-r.x1, -r.y1);
    txt21matrix.AppendRotation(Angle*3.14159f / 180.f);
    txt21matrix.AppendScaling(2);
    txt21matrix.AppendTranslation(r.x1, r.y1);
    ptxt21->SetMatrix(txt21matrix);

    pDm2->Capture();

    FxPlayerAppBase::OnUpdateFrame(needRepaint);
}
```

注意: Bin/Samples ディレクトリから、実行ファイルが位置する同じディレクトリに (実行ファイルを手動で実行する場合)、またはプロジェクト ディレクトリに (たとえば、Projects/Win32/Msvc90/Samples/DrawText、Visual Studio 2008 から実行する場合) drawtext_fonts.swf をコピーします。そうしなければ、ほとんどのグリフはただの四角形にしか表示されません。

スクリーンショット: