Autodesk®
Scaleform®

文本繪製 API

本文檔描述了 Scaleform 4.4 中的 DrawTextAPI 函數。該 API 函數能用來作爲 C++ 文本渲染驅動以及 GFx::Movie 和 ActionScript 之外的格式化處理。

作者： Artem Bolgar
版本： 2.0
最後修訂： 2011 年 4 月 22 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Autodesk Scaleform 聯繫方式：

文檔	文本繪製 API
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網站	www.scaleform.com
郵箱	info@scaleform.com
電話	(301) 446-3200
傳真	(301) 446-3199

目錄

1 介紹 1

1.1 DrawTextManager 2

1.1.1 創建 DrawText 3

1.1.2 文本 Rendering 5

1.1.3 測量文本尺寸 5

1.2 DrawText 6

1.2.1 設置和獲取純文本方法 7

1.2.2 設置和獲取 HTML 文本方法 7

1.2.3 設置文本格式方法 8

1.2.4 文本對齊方法 9

1.2.5 改變文本位置和方位 9

2 DrawText 實例代碼總覽 11

1 介紹

Scaleform SDK 執行體包括功能強大的字體渲染庫和格式化引擎，主要作爲基於 Flash®的 UI 介面的渲染文本。Scaleform 中的文本渲染支援幾項高級特性如光柵和隱藏，文本區域段落對齊和從系統字
體庫或 SWF/GFX 內嵌文件中獲取 HTML 格式化標簽文本。

在多數情況下，Scaleform 文本渲染系統在播放 Flash 文件時自動使用，可以在 Flash Studio 中顯性得看到文本域格式。作爲輔助性控制，文本格式通過 TextField 和 TextFormat 動態腳本（ActionScript,AS）類展現，這兩個類提供了首選的文本介面。有了這幾個 API 函數，很少需要用到 C++ API 來實現這幾項功能。

然而，也有在一定的情況下用 ActionScript 不方便來處理文本渲染，或者相關不符合需求的情況。螢幕上移動名字條的渲染或文本標簽可以用 C++更加高效地進行處理，可以彌補遊戲不支援完全的 Flash UI 描述條目的缺陷。此外，如果遊戲用 3D 引擎渲染 HUD 整合了基於 Scaleform 功能表系統，在兩種情況下仍然需要使用相同的字體系統。

Scaleform 4 或者更高版本 引入了 C++ 文本繪製驅動 API，實現了 Scaleform 字體渲染和文本格式化 引擎。新的文本 API 函數仍然使用 Scaleform 播放器的相同 Render::Renderer 介面；但是，不需要 GFx::Movie 物件的創建，允許開發者在視窗內直接定位和操縱文本域，無需用到 ActionScript。

文本繪製 API 函數有兩個字體庫：系統字形檔和從 SWF/GFX 文件導入的字形檔。第一種情況，需要系統字形檔源。第二種情況，包含字體資訊的 SWF 文件需要作爲 GFx::MovieDef 導入（使用 GFx::Loader::CreateMovie 方法），創建的 GFx::MovieDef 可作爲參數傳遞到 GFx::DrawTextManager 構造器。

兩個 C++分類文本渲染功能器爲 GFx::DrawTextManager 和 GFx::DrawText；
它們的用法在下面列出並在文檔的接下來部分有詳細描述。

- *`DrawTextManager`* – 創建 DrawText 類的實例。用來初始化文本渲染和視窗、字體配置以及啓動/停止文本渲染。

- *`DrawText`* – 螢幕上繪製一個矩形文本域，展示文本格式和和渲染能力。本類可以解析 Flash HTML 標簽或使用具有輔助格式化資料的純文本。
  文本區域分配可以通過成員函數來改變，如 SetColor,，SetFont,和 SetFontStyle。

1.1 DrawTextManager

DrawTextManager 實例類管理 DrawText 實例類，創建、渲染和測量文本範圍。

有多重創建 DrawTextManager 類的方法：

1. `DrawTextManager();`

   一個默認的構造器。將創建字體緩存管理器的內部實例，資源庫和日誌。系統字體提供者將被用作字體資源（SetFontProvider 方法）。

實例：

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

2. `DrawTextManager(MovieDef* pmovieDef);`

   這個構造器獲取一個指向 GFx::MovieDef 的指標作爲參數。DrawTextManager 實例繼承了已被傳遞的 MovieDef 實例的狀態，包括字體緩存管理、字體提供者等。所有包含在 MovieDef 實例中的字體都可被 DrawTextManager 訪問（所有 DrawText instances 由這個 DrawTextManager 創建）。

實例：

```
Ptr<MovieDef> pmd = *loader.CreateMovie("drawtext fonts.swf");
Ptr<DrawTextManager> pdm = *new DrawTextManager(pmd);
```

3. `DrawTextManager(Loader* ploader);`

   該構造體擁有一個指向 GFx::Loader 的指標參數並在載入時候集成了所有的狀態資訊，包括字體庫。

實例：

```
Loader loader;
..
Ptr<DrawTextManager> pdm = *new DrawTextManager(&loader);
```

1.1.1 創建 DrawText

在 DrawTextManager 類中創建 DrawText 實例有多種方法。DrawTextManager 的例子之一爲可以創建任意多的 DrawText 實例。

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

上面的方法中創建 DrawText 實例使用純文本（空結束符 UTF8 字元，空結束符 Unicode/UCS-2 和 Scaleform::String）

`putf8str` ：指定一個空結束符 UTF-8 字元。

`pwstr` ：指定一個空結束符 Unicode/UCS-2 字元。

`str`:：指定一個 Scaleform::String.

`viewRect`:指定匹配的文本可是範圍（圖元），視圖中對應于左上角的視圖位置（查看 BeginDisplay）。

`ptxtParams`:可選參數。指定新創建文本參數。文本參數 TextParams 具有以下結構：

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

`TextColor` ：指定文本顏色，包括通道透明值。

`HAlignment`：指定水平對齊。可能的值爲：DrawText::Align_Left (默認), DrawText::Align_Right, DrawText::Align_Center, DrawText::Align_Justify。

VAlignment：指定垂直對齊，可能值爲：DrawText::VAlign_Top (默認), DrawText::VAlign_Bottom, DrawText::VAlign_Center。

FontStyle ：指定字體類型，如粗體、下劃線、常規或粗斜體。可能的值爲： DrawText::Normal, DrawText::Bold, DrawText::Italic, DrawText::BoldItalic (DrawText::ItalicBold 意義相同)。

FontSize：指定字體大小，以圖元爲單位。可能有分片 。

FontName: 指定字體名字，例如“Arial”，“Times New Roman”。名字中不要含有“Bold”或“Italic”尾碼；使用字體類型代替。

Underline：指定是否用到下劃線。

Multiline:多行/單行文本選擇。

WordWrap：打開/關閉文本邊框，只在多行文本框有用。

若 ptxtParams 選項參數沒有指定，DrawTextManager 使用默認的文本參數。通過以下方法可以設置或者獲取默認文本參數：

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

下列 DrawTextManager 方法 - CreateHtmlText 與前面已經描述的 CreateText 方法類似，但是這裏是從 HTML 創建 DrawText 實例：

```
// 使用指定HTML創建和初始化一個DrawText物件。
DrawText* CreateHtmlText(const char* putf8Str, const RectF&  viewRect);
DrawText* CreateHtmlText(const wchar_t* pwstr, const RectF& viewRect);
DrawText* CreateHtmlText(const String& str, const RectF& viewRect);
```

示例：

```
// 使用純文本和參數創建GFx::DrawText物件。
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);

// 從HTML創建GFx::DrawText物件。
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 文本 Rendering

渲染文本的方式与渲染 GFx::Movie 的方式相同。必须获取 DisplayHandle (GFx::DrawTextManager) 和设置视窗 (DrawTextManager::SetViewport)。有关"视窗"(Viewport) 的详细信息，请参阅 GFx 文档。

1.1.3 測量文本尺寸

DrawTextManager 提供了測量文本矩形尺寸的功能，便於渲染文本。

```
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 方法使用純文本，也可以用 TextParams 參數和文本預期寬度值來獲取文本框矩形尺寸。

GetHtmlTextExtent 方法使用 HTML 文本，也可用 TextParams 參數和文本預期寬度值來獲取文本框舉行尺寸。

選項‘width’參數在多行文本框和文字邊框應用中用來指定文本矩形大小。在這種情況下，只計算出文本矩形的高度。

ptxtParams 參數指定文本參數，在計算文本尺寸的時候被用到。如果該參數沒被指定則使用默認文本參數（查看 SetDefaultTextParams/GetDefaultTextParams）。在 HTML 版本中，ptxtParams 參數作爲一套默認的文本參數，所以，所有從 HTML 解析而來的類型實際上應用在類型‘ptxtParams’參數之上。

示例：

```
// 純文本擴展
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 文本擴展
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 DrawText

DrawText 類提供文本設置、HTML 解析、格式化和文本渲染功能。

1.2.1 設置和獲取純文本方法

SetText 方法設置 UTF8，UCS2 或文本物件的 Scaleform::String 文本值。可選參數‘lengthInBytes’指定 UTF8 字元的位元組數；‘lengthInChars’指定字串字元數。如果這些參數沒有指定，則 UTF-8 和 UCS-2 字串應該爲空結束符。

```
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 方法返回 UTF8 格式的當前設置文本。返回純文本值；即使使用了 HTML 也如此，其返回了所有 HTML 標簽的字串。

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

1.2.2 設置和獲取 HTML 文本方法

SetHtmlText 方法解析 UTF8,UCS2 或 Scaleform::String HTML 編碼和用 HTML 文本初始化文本對。可選參數‘lengthInBytes’指定 UTF8 字元的位元組數；‘lengthInChars’指定字串字元數。如果這些參數沒有指定，則 UTF-8 和 UCS-2 字串應該爲空結束符。

```
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 方法以 HTML 格式返回當前設置文本。如果純文本用調用方法設置格式使，如 SetColor, SetFont 等，然後這些文本將會通過本方法轉換成對應的 HTML 格式。

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

1.2.3 設置文本格式方法

有多種方法設置和獲取文本格式。

```
void SetColor(Color c, UPInt startPos = 0, UPInt endPos = UPInt(-1));
```

設置整個文本或以[startPos..endPos]爲間隔的部分文本顏色 (R, G, B, A) 。

```
void SetFont (const char* pfontName, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```

設置整個文本或者以[startPos..endPos]爲間隔的部分文本字體。‘startPos’和‘endPos’都爲可選項。

```
void SetFontSize(Float fontSize, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```

設置整個文本或者以[startPos..endPos]爲間隔的部分文本字體大小。‘startPos’和‘endPos’都爲可選項。

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

    enum FontStyle
    {
        Normal,
        Bold,
        Italic,
        BoldItalic,
        ItalicBold = BoldItalic
    };
```

設置整個文本或者以[startPos..endPos]爲間隔的部分文本字體類型。‘startPos’和‘endPos’都爲可選項。

```
void SetUnderline(bool underline, UPInt startPos = 0, UPInt endPos = UPInt(-1))
```

設置或清楚整個文本或者以[startPos..endPos]爲間隔的部分文本下劃線。

```
void SetMultiline(bool multiline);
```

設置多行（參數‘multiline’設置爲 true）或單行（false）文本類型。

```
bool IsMultiline() const;
```

如果文本爲多行則返回‘true’；否則返回‘false’。

```
void SetWordWrap(bool wordWrap);
```

文字邊框打開/關閉。

```
bool Wrap() const;
```

返回文字邊框狀態。

1.2.4 文本對齊方法

Alignment 和 Valignment 類型定義：

```
    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
   };
```

文本對齊的不同方法如下所示：

```
void SetAlignment(Alignment);
```
設置水平文本對齊（右，左，居中）。

```
Alignment   GetAlignment() const;
```
返回水平文本對齊（右，左，居中）。

```
void SetVAlignment(VAlignment);
```
設置垂直文本對齊（置頂，置底，居中）。

```
VAlignment  GetVAlignment() const;
```
返回垂直文本對齊（置頂，置底，居中）。

1.2.5 改變文本位置和方位

DrawText 類具有定位和定向文本物件的多種不同方法。

```
void SetRect(const RectF& viewRect);
```

設置試圖矩形，以圖元爲座標。

```
RectF  GetRect() const;
```

返回當前使用的試圖矩形，以圖元爲座標。

```
void SetMatrix(const Matrix& matrix);
```

設置文本物件的轉換矩陣。

```
const Matrix GetMatrix() const;
```

返回當前使用的轉換矩陣。

```
void SetCxform(const Cxform& cx);
```

設置文本物件顏色轉換矩陣。

```
const Cxform& GetCxform() const;
```

返回當前使用的顏色轉換矩陣。

2 DrawText 實例代碼總覽

本節介紹了 Scaleform 3.3 中包含的 DrawText API 實現例子。在 Scaleform SDK 中包含的 DrawTextAPI 函數可以用來在螢幕上易於繪製簡單文本物件，而不會消耗更多記憶體，降低標準性能。在 DrawText API 函數中的實例代碼描述類用來繪製
文本並能方便地包含在文本引擎中來顯示文本。

基本步骤包括设置窗口、加载电影数据以及添加一个显示句柄以捕获和渲染文本。

FxPlayerApp 类是播放器应用类，它定义设置窗口、加载电影和渲染文本的所有属性和函数。FxPlayerApp 继承自 FxPlayerAppBase 类，后者是创建和初始化窗口、渲染线程和图形的基类。

FxPlayerApp 的 OnInit 方法初始化 DrawText 实例，创建文本，并应用各种文本转换。OnUpdateFrame 方法在文本上应用转换矩阵以创建动画。

```
      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;
};
```

DrawText 實例描述了如何使用 DrawTextManager 類介面的不同實例完成文本的渲染。

方法 1

本例中，DrawTextManager 實例使用一個指向 GFx::Loader 的指標以從導入器繼承所有的的狀態。

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

在这里，我们使用系统字体以便于显示文本，并因此而使用 GFx::FontProvider。FontProvider 是在 FxPlayerAppBase::OnInit 中创建的。

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

如 API 函數中所提到的，有多種使用純文本或者 HTML 文本創建 DrawText 實例的方法。本例中，使用純文本和 Scaleform::String 來創建 DrawText 實例。

```
// 使用純文本和默認文本參數來創建文本
DrawTextManager::TextParams defParams =
     pDm1->GetDefaultTextParams();
defParams.TextColor = Color(0xF0, 0, 0, 0xFF); // red, alpha = 255
defParams.FontName  = "Arial";
defParams.FontSize  = 16;
pDm1->SetDefaultTextParams(defParams);
```

通過調用 CreateText 方法創建文本，通過預先調用 SetDefaultTextParams 使用默認的文本參數設置。

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

DrawTextManager 类的 GetTextExtent 方法測量文本矩形的尺寸。而且 DrawText 类可用来操作文本，例如，格式化字体、对齐文本或更改文本对象的位置。

```
// 使用Scaleform::String和TextParams創建文本
   String str(
      "Scaleform GFx 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);
```

一旦使用系统字体创建文本，就给 DrawText 设置视窗，捕获当前 DrawText 状态，并将 DrawText DisplayHandle 添加到 RenderThread:

```
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

本實例創建一個 DrawTextManger 實例，使用 GFx::MovieDef 指標共用 MovieDef 中包含的字體。從 MovieDef 導入的 SWF 文件包含了文本字體。請參考 Scaleform Integration Tutorial 獲得更多關於 Scaleform 導入視頻物件的資訊。

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

與之前的實例不同，我們使用 HTML 文本替代純文本來顯示文本（當然 HTML 也會在前面的例子中用到）。文本矩形的尺寸由 **DrawTextManager::GetHtmlExtent method** 方法來計算。

```
      // 創建HTML文本，使用GFx::MovieDef中的字體
      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>адабрA!</FONT></P>"
            L"<P><FONT FACE='Arial Unicode MS'>Hànyǔ; 華語/華語</FONT></P>"
            L"<P><FONT FACE='Batang'>     /     </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);
```

我們也需要創建文本來證明文本動畫（旋轉和顏色改變）：

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

以及过滤器:

```
      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);
```

为 DrawText 设置视窗，捕获当前 DrawText 状态，并将 DrawText DisplayHandle 添加到 RenderThread：

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

作爲例子，旋轉和改變由DrawText的**SetMatrix**和**SetCxform**方法創建的文本顏色。我们是在 FxPlayerApp::OnUpdateFrame 内完成的：

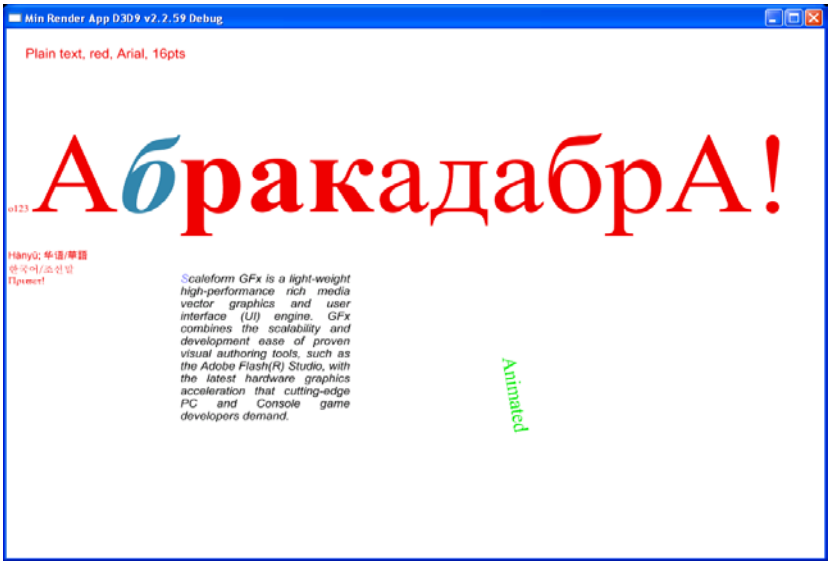
```
void FxPlayerApp::OnUpdateFrame( bool needRepaint )
{
    DrawText::Matrix txt21matrix;
    Angle += 1;
    RectF r = ptxt21->GetRect();
    txt21matrix.AppendTranslation(-r.x1, -r.y1);
    txt21matrix.AppendRotation(Angle*3.14159f / 180.f);
    txt21matrix.AppendScaling(2);
    txt21matrix.AppendTranslation(r.x1,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

    pDm2->Capture();

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

注意：從 Bin/Samples 目錄拷貝 drawtext_fonts.swf 文件到執行文件所在的目錄（如果手動運行可執行程式）或者到工程目錄中去（例如：Projects/Win32/Msvc90/Samples/DrawText，若從 Visual Studio 2008 運行程式）。否則，只能看到字體輪廓。

截圖：


Min Render App D3D9 v2.2.59 Debug
Plain text, red, Arial, 16pts
АбракадабрА!
Hànyǔ; 华语/華語
Привет!
Scaleform GFx is a light-weight high-performance rich media vector graphics and user interface (UI) engine. GFx combines the scalability and development ease of proven visual authoring tools, such as the Adobe Flash(R) Studio, with the latest hardware graphics acceleration that cutting-edge PC and Console game developers demand.
Animated