Autodesk®
Scaleform®

文本绘制 API

本文档描述了 Scaleform 4.4 中的 DrawTextAPI 函数。该 API 函数能用来作为 C++ 文本渲染驱动以及 GFx::Movie 和 ActionScript 之外的格式化处理。

作者： Artem Bolgar
版本： 2.0
最后修订： 2011 年 4 月 22 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

Autodesk Scaleform 联系方式：

文档	文本绘制 API
地址	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
网站	www.scaleform.com
邮箱	info@scaleform.com
电话	(301) 446-3200
传真	(301) 446-3199

目录

1 介绍

Scaleform SDK 执行体包括功能强大的字体渲染库和格式化引擎，主要作为基于 Flash®的 UI 界面的渲染文本。Scaleform 中的文本渲染支持几项高级特性如光栅和隐藏，文本区域段落对齐和从系统字 体库或 SWF/GFX 内嵌文件中获取 HTML 格式化标签文本。

在多数情况下，Scaleform 文本渲染系统在播放 Flash 文件时自动使用，可以在 Flash Studio 中显性得看到文本域格式。作为辅助性控制，文本格式通过 TextField 和 TextFormat 动态脚本（ActionScript,AS）类展现，这两个类提供了首选的文本接口。有了这几个 API 函数，很少需要用到 C++ API 来实现这几项功能。

然而，也有在一定的情况下用 ActionScript 不方便来处理文本渲染，或者相关不符合需求的情况。屏幕上移动名字条的渲染或文本标签可以用 C++更加高效地进行处理，可以弥补游戏不支持完全的 Flash UI 描述条目的缺陷。此外，如果游戏用 3D 引擎渲染 HUD 整合了基于 Scaleform 菜单系统，在两种情况下仍然需要使用相同的字体系统。

Scaleform 3.3 或者更高版本 引入了 C++ 文本绘制驱动 API，实现了 Scaleform 字体渲染和文本格式化 引擎。新的文本 API 函数仍然使用 Scaleform 播放器的相同 Render::Renderer 接口；但是，不需要 GFx::Movie 对象的创建，允许开发者在视窗内直接定位和操纵文本域，无需用到 ActionScript。

文本绘制 API 函数有两个字体库：系统字库和从 SWF/GFX 文件导入的字库。第一种情况，需要系统字库源。第二种情况，包含字体信息的 SWF 文件需要作为 GFx::MovieDef 导入（使用 GFx::Loader::CreateMovie 方法），创建的 GFx::MovieDef 可作为参数传递到 GFx::DrawTextManager 构造器。

两个 C++分类文本渲染功能器为 GFx::DrawTextManager 和 GFx::DrawText；
它们的用法在下面列出并在文档的接下来部分有详细描述。

- *`GFx::DrawTextManager`* – 创建 DrawText 类的实例。用来初始化文本渲染和视窗、字体配置以及启动/停止文本渲染。

- *`GFx::DrawText`* – 屏幕上绘制一个矩形文本域，展示文本格式和和渲染能力。本类可以解析 Flash HTML 标签或使用具有辅助格式化数据的纯文本。
  文本区域分配可以通过成员函数来改变，如 SetColor,，SetFont,和 SetFontStyle。

1.1 GFx::DrawTextManager

DrawTextManager 实例类管理 DrawText 实例类，创建、渲染和测量文本范围。

有多重创建 DrawTextManager 类的方法：

1. `DrawTextManager();`

   一个默认的构造器。将创建字体缓存管理器的内部实例，资源库和日志。系统字体提供者将被用作字体资源（SetFontProvider 方法）。

实例：

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

2. `DrawTextManager(MovieDef* pmovieDef);`

   这个构造器获取一个指向 MovieDef 的指针作为参数。DrawTextManager 实例继承了已被传递的 MovieDef 实例的状态，包括字体缓存管理、字体提供者等。所有包含在 MovieDef 实例中的字体都可被 DrawTextManager 访问（所有 DrawText instances 由这个 DrawTextManager 创建）。

实例：

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

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

   该构造体拥有一个指向 Loader 的指针参数并在载入时候集成了所有的状态信息，包括字体库。

实例：

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

1.1.1 创建 GFx::DrawText

在 DrawTextManager 类中创建 DrawText 实例有多种方法。DrawTextManager 的例子之一为可以创建任意多的 DrawText 实例。

- `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);`

上面的方法中创建 DrawText 实例使用纯文本（空结束符 UTF8 字符，空结束符 Unicode/UCS-2 和 Scaleform::String）

`putf8str`：指定一个空结束符 UTF-8 字符。

`pwstr`：指定一个空结束符 Unicode/UCS-2 字符。

`str`:：指定一个 Scaleform::String.

`depth`: Specifies the order of drawing.

`viewRect`:指定匹配的文本可是范围（像素），视图中对应于左上角的视图位置（查看 BeginDisplay）。

`ptxtParams`:可选参数。指定新创建文本参数。文本参数 TextParams 具有以下结构：

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

`TextColor`：指定文本颜色，包括通道透明值。

`HAlignment`：指定水平对齐。可能的值为：DrawText::Align_Left (默认), DrawText::Align_Right, DrawText::Align_Center, DrawText::Align_Justify。

VAlignment：指定垂直对齐，可能值为：DrawText::VAlign_Top (默认), DrawText::VAlign_Bottom, DrawText::VAlign_Center。

FontStyle：指定字体类型，如粗体、下划线、常规或粗斜体。可能的值为： DrawText::Normal, DrawText::Bold, DrawText::Italic, DrawText::BoldItalic (DrawText::ItalicBold 意义相同)。

FontSize：指定字体大小，以像素为单位。可能有分片。

FontName: 指定字体名字，例如“Arial”，“Times New Roman”。名字中不要含有“Bold”或“Italic”后缀；使用字体类型代替。

Underline：指定是否用到下划线。

Multiline:多行/单行文本选择。

WordWrap：打开/关闭文本边框，只在多行文本框有用。

若 ptxtParams 选项参数没有指定，DrawTextManager 使用默认的文本参数。通过以下方法可以设置或者获取默认文本参数：

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

下列 DrawTextManager 方法 - CreateHtmlText 与前面已经描述的 CreateText 方法类似，但是这里是从 HTML 创建 DrawText 实例：

```
// 使用指定HTML创建和初始化一个GFx::DrawText对象。
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);
```

示例：

```
// 使用纯文本和参数创建GFx::DrawText对象。
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);

// 从HTML创建GFx::DrawText对象。
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 测量文本尺寸

DrawTextManager 提供了测量文本矩形尺寸的功能，便于渲染文本。

```
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 方法使用纯文本，也可以用 TextParams 参数和文本预期宽度值来获取文本框矩形尺寸。

GetHtmlTextExtent 方法使用 HTML 文本，也可用 TextParams 参数和文本预期宽度值来获取文本框举行尺寸。

选项‘width’参数在多行文本框和文字边框应用中用来指定文本矩形大小。在这种情况下，只计算出文本矩形的高度。

ptxtParams 参数指定文本参数，在计算文本尺寸的时候被用到。如果该参数没被指定则使用默认文本参数（查看 SetDefaultTextParams/GetDefaultTextParams）。在 HTML 版本中，ptxtParams 参数作为一套默认的文本参数，所以，所有从 HTML 解析而来的类型实际上应用在类型‘ptxtParams’参数之上。

示例：

```
// 纯文本扩展
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 文本扩展
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 GFx::DrawText

DrawText 类提供文本设置、HTML 解析、格式化和文本渲染功能。

1.2.1 设置和获取纯文本方法

SetText 方法设置 UTF8，UCS2 或文本对象的 Scaleform::String 文本值。可选参数‘lengthInBytes’指定 UTF8 字符的字节数；‘lengthInChars’指定字符串字符数。如果这些参数没有指定，则 UTF-8 和 UCS-2 字符串应该为空结束符。

```
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 格式的当前设置文本。返回纯文本值；即使使用了 HTML 也如此，其返回了所有 HTML 标签的字符串。

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

1.2.2 设置和获取 HTML 文本方法

SetHtmlText 方法解析 UTF8,UCS2 或 String HTML 编码和用 HTML 文本初始化文本对。
可选参数‘lengthInBytes’指定 UTF8 字符的字节数；‘lengthInChars’指定字符串字符数。如果这些参数没有指定，则 UTF-8 和 UCS-2 字符串应该为空结束符。

```
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 格式返回当前设置文本。如果纯文本用调用方法设置格式使，如 SetColor, SetFont 等，然后这些文本将会通过本方法转换成对应的 HTML 格式。

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

1.2.3 设置文本格式方法

有多种方法设置和获取文本格式。

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

设置整个文本或以[startPos..endPos]为间隔的部分文本颜色 (R, G, B, A) 。

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

设置整个文本或者以[startPos..endPos]为间隔的部分文本字体。‘startPos’和‘endPos ’都为可选项。

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

设置整个文本或者以[startPos..endPos]为间隔的部分文本字体大小。‘startPos’和‘endPos ’都为可选项。

```
void SetFontStyle(FontStyle, UPInt startPos = 0, UPInt endPos = UPInt(-1));
    enum FontStyle
    {
        Normal,
        Bold,
        Italic,
        BoldItalic,
        ItalicBold = BoldItalic
    };
```

设置整个文本或者以[startPos..endPos]为间隔的部分文本字体类型。‘startPos’和‘endPos ’都为可选项。

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

设置或清楚整个文本或者以[startPos..endPos]为间隔的部分文本下划线。

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

设置多行（参数‘multiline’设置为 true）或单行（false）文本类型。

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

如果文本为多行则返回‘true’；否则返回‘false’。

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

文字边框打开/关闭。

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

返回文字边框状态。

1.2.4 文本对齐方法

Alignment 和 Valignment 类型定义：

```
    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
   };
```

文本对齐的不同方法如下所示：

```
void SetAlignment(Alignment);
```
设置水平文本对齐（右，左，居中）。

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

```
void SetVAlignment(VAlignment);
```
设置垂直文本对齐（置顶，置底，居中）。

```
VAlignment  GetVAlignment() const;
```
返回垂直文本对齐（置顶，置底，居中）。

1.2.5 改变文本位置和方位

DrawText 类具有定位和定向文本对象的多种不同方法。

```
void SetRect(const RectF& viewRect);
```
设置试图矩形，以像素为坐标。

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

返回当前使用的试图矩形，以像素为坐标。

```
void SetMatrix(const Matrix& matrix);
```
设置文本对象的转换矩阵。

```
const Matrix GetMatrix() const;
```
返回当前使用的转换矩阵。

```
void SetCxform(const Cxform& cx);
```
设置文本对象颜色转换矩阵。

```
const Cxform& GetCxform() const;
```
返回当前使用的颜色转换矩阵。

2 DrawText 实例代码总览

本节介绍了 Scaleform 4.1 中包含的 DrawText API 实现例子。在 Scaleform SDK 中包含的 DrawTextAPI 函数可以用来在屏幕上易于绘制简单文本对象，而不会消耗更多内存，降低标准性能。在 DrawText API 函数中的实例代码描述类用来绘制 文本并能方便地包含在文本引擎中来显示文本。

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 实例描述了如何使用 DrawTextManager 类接口的不同实例完成文本的渲染。

方法 1

本例中，DrawTextManager 实例使用一个指向 GFx::Loader 的指针以从导入器继承所有的的状态。

```
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 函数中所提到的，有多种使用纯文本或者 HTML 文本创建 DrawText 实例的方法。本例中，使用纯文本和 Scaleform::String 来创建 DrawText 实例。

```
// 使用纯文本和默认文本参数来创建文本
DrawTextManager::TextParams defParams =
     pDm1->GetDefaultTextParams();
defParams.TextColor = Color(0xF0, 0, 0, 0xFF); // red, alpha = 255
defParams.FontName  = "Arial";
defParams.FontSize  = 16;
pDm1->SetDefaultTextParams(defParams);
```

通过调用 CreateText 方法创建文本，通过预先调用 `SetDefaultTextParams` 使用默认的文本参数设置。

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

DrawTextManager 类的 GetTextExtent 方法测量文本矩形的尺寸。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创建文本
  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

本实例创建一个 DrawTextManger 实例，使用 GFx::MovieDef 指针共享 GFx::MovieDef 中包含的字体。从 MovieDef 导入的 SWF 文件包含了文本字体。请参考 Scaleform Integration Tutorial 获得更多关于导入视频对象的信息。

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

与之前的实例不同，我们使用 HTML 文本替代纯文本来显示文本（当然 HTML 也会在前面的例子中用到）。文本矩形的尺寸由 **DrawTextManager::GetHtmlExtent method** 方法来计算。

```
// 创建HTML文本，使用GFx::MovieDef中的字体
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'>     /     </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);
```

我们也需要创建文本来证明文本动画：

```
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);
```

作为例子，旋转和改变由DrawText的SetMatrix和SetCxform方法创建的文本颜色。我们是在 FxPlayerApp::OnUpdateFrame 内完成的：：

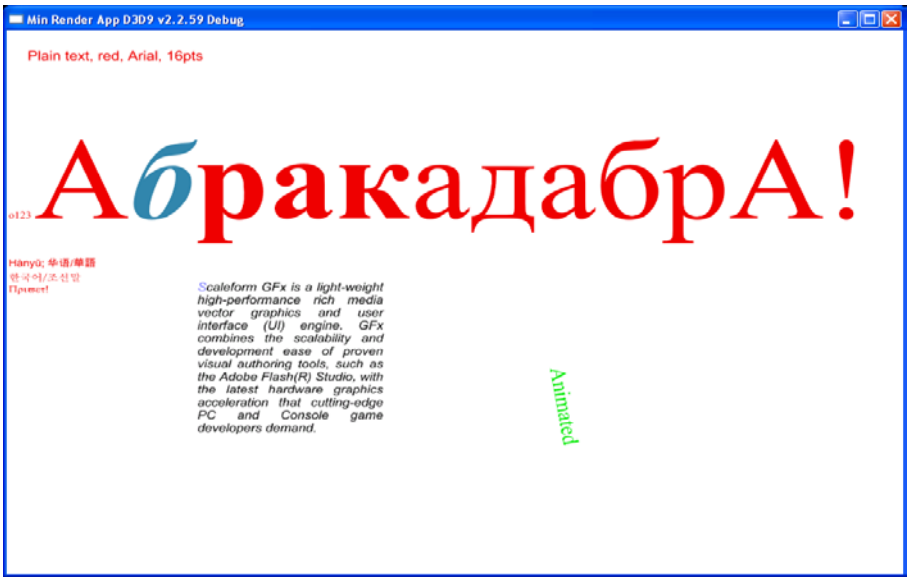
```
void FxPlayerApp::OnUpdateFrame( bool needRepaint )
{
    DrawText::Matrix txt21matrix;
    Angle += 1;
    RectF r = ptxt21->GetRect();
    txt21matrix.AppendTranslation(-r.x1, -r.y1);
    txt21matrix.AppendRotation(Angle*3.14159f / 180.f);
    txt21matrix.AppendScaling(2);
    txt21matrix.AppendTranslation(r.x1,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

    pDm2->Capture();

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

注意：从 Bin/Samples 目录拷贝 drawtext_fonts.swf 文件到执行文件所在的目录（如果手动运行可执行程序）或者到工程目录中去（例如：Projects/Win32/Msvc90/Samples/DrawText，若从 Visual Studio 2008 运行程序）。否则，只能看到字体轮廓。

截图：


Min Render App D3D9 v2.2.59 Debug
Plain text, red, Arial, 16pts
АбракадабрА!
Hànyǔ; 华语/華語
Ερμηνεί
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