

## DrawText API

This document describes the GfXDrawText API available in GfX 3.3. This API can be used for C++ driven text rendering and formatting outside of the GfXMovieView and ActionScript sandbox.

Author: Artem Bolgar  
Version: 1.03  
Last Edited: May 26, 2009

# Copyright Notice

## Autodesk® Scaleform® 3

© 2011 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries: 3DEC (design/logo), 3December, 3December.com, 3ds Max, Algor, Alias, Alias (swirl design/logo), AliasStudio, Alias|Wavefront (design/logo), ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Intent, Autodesk Inventor, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSnap, AutoSketch, AutoTrack, Backburner, Backdraft, Beast, Built with ObjectARX (logo), Burn, Buzzsaw, CAiCE, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design Web Format, Discreet, DWF, DWG, DWG (logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DXF, Ecotect, Exposure, Extending the Design Team, Face Robot, FBX, Fempro, Fire, Flame, Flare, Flint, FMDesktop, Freewheel, GDX Driver, Green Building Studio, Heads-up Design, Heidi, HumanIK, IDEA Server, i-drop, Illuminate Labs AB (design/logo), ImageModeler, iMOUT, Incinerator, Inferno, Inventor, Inventor LT, Kynapse, Kynogon, LandXplorer, LiquidLight, LiquidLight (design/logo), Lustre, MatchMover, Maya, Mechanical Desktop, Moldflow, Moldflow Plastics Advisers, MPI, Moldflow Plastics Insight, Moldflow Plastics Xpert, Moondust, MotionBuilder, Movimento, MPA, MPA (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, RiverCAD, Robot, Scaleform, Scaleform AMP, Scaleform CLIK, Scaleform GFx, Scaleform IME, Scaleform Video, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Softimage|XSI (design/logo), Sparks, SteeringWheels, Stitcher, Stone, StormNET, StudioTools, ToolClip, Topobase, Toxik, TrustedDWG, U-Vis, ViewCube, Visual, Visual LISP, Volo, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

All other brand names, product names or trademarks belong to their respective holders.

### Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS". AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

How to Contact Autodesk Scaleform:

Document	DrawText API
Address	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	<a href="http://www.scaleform.com">www.scaleform.com</a>
Email	<a href="mailto:info@scaleform.com">info@scaleform.com</a>
Direct	(301) 446-3200
Fax	(301) 446-3199

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1. GfxDrawTextManager .....	2
1.1.1. Creating GfxDrawText. ....	3
1.1.2. Text Rendering .....	5
1.1.3. Measuring Text Extents .....	5
1.2. GfxDrawText .....	8
1.2.1. Setting and Getting Plain Text Methods .....	8
1.2.2. Setting and Getting HTML Text Methods .....	8
1.2.3. Setting Text Format Methods .....	9
1.2.4. Aligning Text Methods .....	10
1.2.5. Changing Text Position and Orientation .....	10
1.2.6. Rendering Text Method .....	11
1.2.7. Using Custom Directives for Rendering Text .....	11
<b>2. Overview of DrawText Sample Code.....</b>	<b>12</b>

# 1. Introduction

The Scaleform GfX SDK implementation contains a powerful font rendering and text formatting engine, primarily used for rendering text displayed as a part of the Flash® based UI. Text rendering in GfX supports a number of advanced features such as on-the-fly rasterization and caching of screen-pixel aligned glyphs, scrollable text fields with paragraph alignment, and rich HTML-tag formatted text with fonts obtained from either system font providers or the SWF/GFX embedded file data.

In most cases, GfX text rendering system is used automatically when playing Flash files, with text fields arranged visually by the artist in the Flash Studio. For additional control, text formatting is exposed through the TextField and TextFormat ActionScript (AS) classes, which provide a preferred way of interfacing with text. With these APIs in place, it is rarely necessary to use the C++ APIs for this purpose.

There are, however, occasional situations when going through ActionScript for text rendering can be inconvenient, or the associated overhead undesirable. Rendering of name billboards over moving avatars on screen or text labels next to items on a radar can often be done more efficiently through C++ if your game can not afford having full-fledge Flash UI for those items. Furthermore, if your game combines the GfX based menu system with a 3D engine rendered HUD, it is still desirable to use the same font system in both cases.

Scaleform GfX 2.2 and higher versions includes a C++ driven DrawText API, exposing the GfX font rendering and text formatting engines outside of the movie view sandbox. The new text API still uses the same GRenderer interface as the rest of GfX player; however, it does not require creation of the GfXMovieView object, allowing developers to position and manipulate text fields directly within the viewport without the associated ActionScript overhead.

There are two possible sources of fonts for the DrawText API: system fonts and fonts loaded from SWF/GFX files. In the first case, the system font provider should be used. In the second case, the SWF file with fonts should be loaded as a GfXMovieDef (using the GfXLoader::CreateMovie method) and then the created GfXMovieDef could be passed as a parameter to GfXDrawTextManager constructor.

The two C++ that classes that provide text rendering functionality are GfXDrawTextManager and GfXDrawText; their use is outlined below and then explained in detail through the rest of this document.

- *GfXDrawTextManager* - Creates instances of the GfXDrawText class. It is used to initialize text rendering and viewport, configure fonts, and to start/stop rendering the text.

- *GFxDrawText* – Represents an individual rectangular text field on screen, exposing text formatting and rendering capabilities. This class can either parse Flash HTML tags or use plain text with supplementary formatting data. Text field attributes can be changed through member functions such as *SetColor*, *SetFont*, and *SetFontStyle*.

## 1.1. *GFxDrawTextManager*

The *GFxDrawTextManager* class instance manages instances of *GFxDrawText* class; it creates, renders and measures text extents.

There are several ways of creating the *GFxDrawTextManager* class instance:

1. `GFxDrawTextManager();`

A default constructor. Will create internal instances of font cache manager, resource library and log. A system font provider should be used as font source (method *SetFontProvider*).

Example:

```
GPTr<GFxDrawTextManager> pdm = *new GFxDrawTextManager();
```

2. `GFxDrawTextManager(GFxMovieDef* pmovieDef);`

This constructor takes a pointer to *GFxMovieDef* as a parameter. The instance of the *GFxDrawTextManager* inherits all states from the passed *GFxMovieDef* instance, including font cache manager, font providers, etc. All fonts being contained in the *GFxMovieDef* instance become accessible for the *GFxDrawTextManager* (and by all *GFxDrawText* instances created by this *GFxDrawTextManager*).

Example:

```
GPTr<GFxMovieDef> pmd = *loader.CreateMovie("drawtext_fonts.swf");
GPTr<GFxDrawTextManager> pdm = *new GFxDrawTextManager(pmd);
```

3. `GFxDrawTextManager(GFxLoader* ploader);`

This constructor takes a pointer to *GFxLoader* as a parameter and inherits all the states set on the loader including the font library.

Example:

```
GFxLoader loader;
..
GPtr<GFxDrawTextManager> pdm = *new GFxDrawTextManager(loader);
```

### 1.1.1. Creating GFxDrawText.

There are several methods to create GFxDrawText instances in GFxDrawTextManager class. One instance of GFxDrawTextManager can be used to create as many GFxDrawText instances as necessary.

- GFxDrawText\* CreateText(const char\* putf8Str, const GRectF& viewRect, const TextParams\* ptxtParams = NULL);
- GFxDrawText\* CreateText(const wchar\_t\* pwstr, const GRectF& viewRect, const TextParams\* ptxtParams = NULL);
- GFxDrawText\* CreateText(const GString& str, const GRectF& viewRect, const TextParams\* ptxtParams = NULL);

The methods above create GFxDrawText instances using plain text (null-terminated UTF-8 string, null-terminated Unicode/UCS-2 and GString respectively).

putf8str : specifies a null-terminated UTF-8 string.

pwstr : specifies a null-terminated Unicode/UCS-2 string.

str: specifies a GString.

viewRect : specifies coordinates of text view area (in pixels), relatively to top-left corner of viewport (see BeginDisplay).

ptxtParams : Optional parameter. It specifies parameters of newly created text. TextParams has the following structure:

```
struct TextParams
{
    GColor          TextColor;
    GFxDrawText::Alignment HAlignment;
    GFxDrawText::VAlignment VAlignment;
    GFxDrawText::FontStyle  FontStyle;
    Float           FontSize;
    GString         FontName;
    bool            Underline;
    bool            Multiline;
    bool            WordWrap;
};
```

`TextColor` : specifies the color of the text, including the alpha channel.

`HAlignment`: specifies horizontal alignment. Possible values are: `GfxDrawText::Align_Left` (default), `GfxDrawText::Align_Right`, `GfxDrawText::Align_Center`, `GfxDrawText::Align_Justify`.

`VAlignment` : specifies vertical alignment. Possible values are:  
`GfxDrawText::VAlign_Top` (default), `GfxDrawText::VAlign_Bottom`, `GfxDrawText::VAlign_Center`.

`FontStyle` : specifies font style, such as bold, italic, normal or bolditalic. Possible values are:  
`GfxDrawText::Normal`, `GfxDrawText::Bold`, `GfxDrawText::Italic`, `GfxDrawText::BoldItalic` (and `GfxDrawText::ItalicBold` as a synonym).

`FontSize`: specifies size of font, in pixels. May be fractional.

`FontName`: specifies name of the font, for example “Arial”, “Times New Roman”. Do not put “Bold” or “Italic” suffixes in name; use `FontStyle` instead.

`Underline`: specifies whether underline should be used or not.

`Multiline`: toggles multiline/singleline modes for the text.

`WordWrap`: turns on/off wordwrapping; it is meaningful only for multiline text boxes.

If `ptxtParams` optional parameter is not specified then `GfxDrawTextManager` uses default text parameters. It is possible to set and get these default text parameters by using the following methods:

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

The following `GfxDrawTextManager`'s methods – `CreateHtmlText` - are similar to `CreateText` ones described above, but they create the `GfxDrawText` instances from HTML:

```
// Creates and initialize a GfxDrawText object using specified HTML.  
GfxDrawText* CreateHtmlText(const char* putf8Str, const GRectF& viewRect);  
GfxDrawText* CreateHtmlText(const wchar_t* pwstr, const GRectF& viewRect);  
GfxDrawText* CreateHtmlText(const GString& str, const GRectF& viewRect);
```

Examples:

```
// Creation of GfxDrawText object using plain text with parameters.  
GString str("String No 2");  
GfxDrawTextManager::TextParams params;  
params.FontName    = "Symbol";
```



```

params.FontSize      = 30;
params.FontStyle     = GFxDrawText::Italic;
params.Multiline     = false;
params.HAlignment    = GFxDrawText::Align_Right;
params.VAlignment    = GFxDrawText::VAlign_Bottom;

GPtr<GFxDrawText> ptxt;
ptxt = *pdm->CreateText(str, GRectF(20, 300, 400, 700), &params);

// Creation of GFxDrawText object from HTML.
GPtr<GFxDrawText> ptxt;
ptxt = *pdm->CreateHtmlText("<p><FONT size='20'>AB
                           <b>singleline</b><i> CD</i>O",
                           GRectF(20, 300, 400, 700));

```

### 1.1.2.Text Rendering

GFxDrawText::Display method is used to render the text. It is necessary to call GFxDrawTextManager::BeginDisplay before the first call to GFxDrawText::Display and to call GFxDrawTextManager::EndDisplay after the last call to GFxDrawText::Display:

```

GViewport vp(GetWidth(), GetHeight(), 0, 0, GetWidth(), GetHeight(), 0);
pdm->BeginDisplay(vp);

ptxt->Display();
ptxt2->Display();
ptxt3->Display();

pdm->EndDisplay();

```

GFxDrawTextManager::BeginDisplay has a parameter – reference to GViewport that specifies the viewport dimensions for GFxDrawText instances being displayed. Refer to GFx documentation for further details about GViewport.

### 1.1.3. Measuring Text Extents

GFxDrawTextManager provides functionality to measure text rectangle dimensions required to render the text:

```

GSizeF GetTextExtent(const char* putf8Str, Float width = 0,
                    const TextParams* ptxtParams = 0);
GSizeF GetTextExtent(const wchar_t* pwstr, Float width = 0,
                    const TextParams* ptxtParams = 0);
GSizeF GetTextExtent(const GString& str, Float width = 0,

```

```

        const TextParams* ptxtParams = 0);

GSizeF GetHtmlTextExtent(const char* putf8Str, Float width = 0,
                        const TextParams* ptxtParams = 0);
GSizeF GetHtmlTextExtent(const wchar_t* pwstr, Float width = 0,
                        const TextParams* ptxtParams = 0);
GSizeF GetHtmlTextExtent(const GString& str, Float width = 0,
                        const TextParams* ptxtParams = 0);

```

GetTextExtent method calculates dimensions of text rectangle using plain text and, optionally, using TextParams and desired width of text.

GetHtmlTextExtent method calculates dimensions of text rectangle using HTML text and, optionally, using TextParams and desired width of text.

The optional 'width' parameter specifies desired width of text rectangle in the case when multiline and word wrapping are used. In this case, only the height of text rectangle will be calculated.

The optional ptxtParams specifies text parameters, which will be used during text dimensions measurement. If it is not specified then the default text parameters will be used (see SetDefaultTextParams / GetDefaultTextParams). For HTML version, the ptxtParams parameter works as a set of default text parameters, so, all styles from parsed HTML virtually will be applied above the styles from the 'ptxtParams'.

Examples:

```

// Plain text extents
GString str("String No 2");
GfxDrawTextManager::TextParams params;
params.FontName    = "Symbol";
params.FontSize    = 30;
params.FontStyle   = GfxDrawText::Italic;
params.Multiline   = false;
params.HAlignment  = GfxDrawText::Align_Right;
params.VAlignment  = GfxDrawText::VAlign_Bottom;

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

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

```

```
// HTML text extents
const wchar_t* html =
    L"<p><FONT size='20'>AB <b>singleline</b><i> CD</i>0";
sz = pdm->GetHtmlTextExtent(html);

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

## 1.2. GfxDrawText

The GfxDrawText class provides functionality of setting text, parsing HTML, formatting and rendering text.

### 1.2.1. Setting and Getting Plain Text Methods

SetText method sets UTF-8, UCS-2 or GString text value to the text object. The optional parameter 'lengthInBytes' specifies number of bytes in the UTF-8 string; 'lengthInChars' specifies number of characters in wide character string. If these parameters are not specified then the UTF-8 and UCS-2 strings should be null-terminated.

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

GetText method returns currently set text in UTF-8 format. It returns plain text value; even if HTML is used, it returns the string with all HTML tags stripped out.

```
GString GetText() const;
```

### 1.2.2. Setting and Getting HTML Text Methods

SetHtmlText method parses UTF-8, UCS-2 or GString encoded HTML and initializes the text object by the parsed HTML text. The optional parameter 'lengthInBytes' specifies number of bytes in the UTF-8 string; 'lengthInChars' specifies number of characters in wide character string.

If these parameters are not specified then the UTF-8 and UCS-2 strings should be null-terminated.

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

GetHtmlText method returns currently set text in HTML format. If plain text is used with setting formatting by calling methods, such as SetColor, SetFont, etc, then this text will be converted to appropriate HTML format by this method.

```
GString GetHtmlText() const;
```

### 1.2.3. Setting Text Format Methods

There are several methods for setting and getting text format.

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

Sets color (R, G, B, A) to whole text or to the part of text in interval [startPos..endPos]. Both 'startPos' and 'endPos' parameters are optional.

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

Sets font to whole text or to the part of text in interval [startPos..endPos]. Both 'startPos' and 'endPos' parameters are optional.

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

Sets font size to whole text or to the part of text in interval [startPos..endPos]. Both 'startPos' and 'endPos' parameters are optional.

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

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

Sets font style to whole text or to the part of text in interval [startPos..endPos]. Both 'startPos' and 'endPos' parameters are optional.

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

Sets or clears underline to whole text or to the part of text in interval [startPos..endPos]. Both 'startPos' and 'endPos' parameters are optional.

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

Sets multiline (parameter 'multiline' is set to true) or singleline (false) type of the text.

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

Returns 'true' if the text is multiline; 'false' otherwise.

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

Turns wordwrapping on/off.

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

Returns state of wordwrapping.

### 1.2.4. Aligning Text Methods

Type definitions for Alignment and VAlignment:

```
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
};
```

The different methods for aligning the text methods are given below.

```
void SetAlignment(Alignment);
```

Sets horizontal text alignment (right, left, center).

```
Alignment GetAlignment() const;
```

Returns horizontal text alignment (right, left, center).

```
void SetVAlignment(VAlignment);
```

Sets vertical text alignment (top, bottom, center).

```
VAlignment GetVAlignment() const;
```

Returns vertical text alignment (top, bottom, center).

### 1.2.5. Changing Text Position and Orientation

The GfxDrawText class has various methods for positioning and orienting the text object.

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

Sets view rectangle, coordinates are in pixels.

```
GRectF GetRect() const;
```

Returns currently used view rectangle, coordinates are in pixels.

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

Sets transformation matrix to the text object.

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

Returns the currently using transformation matrix.

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

Set color transformation matrix to the text object.

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

Returns the currently using color transformation matrix.

### 1.2.6. Rendering Text Method

The Display method is used to render the text.

```
void Display();
```

Displays the text.

**Note:** GfxDrawTextManager's BeginDisplay method should be called before the first call to this method, and EndDisplay method should be called after the last Display method call.

### 1.2.7. Using Custom Directives for Rendering Text

Custom render implementations are possible by using SetRendererFloat and SetRendererString methods of GfxDrawText. These methods set the user specific data (float or string value) to the GRenderer::PushUserData virtual function for renderer implementations.

```
void SetRendererFloat(float f);  
void SetRendererString(const GString& str);
```

GetRendererFloat and GetRendererString return the float and string values respectively which are used in custom renderer implementations.

```
float GetRendererFloat();  
GString GetRendererString();
```

## 2. Overview of DrawText Sample Code

This section explains the DrawText API sample implementation that is included in Gfx 2.2 and higher versions.

DrawText API has been included in Gfx SDK to provide ease in drawing simple text objects on the screen without overheads on memory and performance criteria. The sample code describes classes used in DrawText API to draw text and can be easily included in your text engines to display text.

The basic steps include setting up the window, creating a render object GRenderer and GfxRenderConfig object to render the text.

FxPlayerApp class is the player application class which defines all the properties and functions for setting up the window, loading the movie and rendering the text. The entry point to the FxPlayer is the **main** function which initializes the settings like the height and width of the window, rendering states and the anti-aliasing mode type. **FxPlayer::Run** method contains the program logic for loading the movie object and rendering it.

Note, remember to initialize the width and height of FxPlayer as these change when **FxPlayerApp::OnSizeEnter** method is called. **OnSizeEnter** method is called whenever the viewport size is changed and the renderer is reinitialized.

Create renderer and GfxRenderConfig objects:

```
if (!CreateRenderer())
    return 1;
GPtr<GfxRenderConfig> pRenderConfig = *new GfxRenderConfig(GetRenderer(),
    GfxRenderConfig::RF_StrokeNormal);
pRenderConfig->SetRenderFlags(pRenderConfig->GetRenderFlags() |
    GfxRenderConfig::RF_EdgeAA);
```

The DrawText sample describes how to render text using two different instances of GfxDrawTextManager class interface.

### Method 1

In this example, GfxDrawTextManager instance is created using a pointer to GfxLoader to inherit all states from the loader.

As the first step in drawing a text object using this method, create GfxLoader object and pass the tracing information to **SetLog** which handles the log stream for debugging.



```
GFxLoader loader;
loader.SetLog(GPTr<GFxLog>(*new GFxPlayerLog()));
```

Now, create the GFxDrawTextManager:

```
GPTr<GFxDrawTextManager> pdml = *new GFxDrawTextManager(loader);
```

Associate the renderer with GFxDrawTextManager instance.

```
pdml->SetRenderConfig(pRenderConfig);
```

Here, we use the system fonts for the text to be displayed and thus GFxFontProvider is used for the purpose.

```
GPTr<GFxFontProviderWin32> fontProvider =
    *new GFxFontProviderWin32(::GetDC(0));
pdml->SetFontProvider(fontProvider);
```

As mentioned in the API, there are several ways of creating DrawText instance by using either plain text or HTML text. In this example, DrawText instances are created using plain text and GString.

```
// Create text using plain text and default text parameters.
GFxDrawTextManager::TextParams defParams =
    pdml->GetDefaultTextParams();
defParams.TextColor = GColor(0xF0, 0, 0, 0xFF); // red, alpha = 255
defParams.FontName = "Arial";
defParams.FontSize = 16;
pdml->SetDefaultTextParams(defParams);
```

The text is created by calling CreateText method and uses the default text parameters set by the previous call to SetDefaultTextParams.

```
GPTr<GFxDrawText> ptxt11 = *pdml->CreateText("Plain text, red, Arial,
16pts",GRectF(20, 20, 500, 400));
```

A beneficial function of GFxDrawTextManager is the GetTextExtent method that measures the dimensions of the text rectangle. And GFxDrawText class can be availed for manipulations on the text such as formatting the font, aligning the text or changing the position of text object.

```
// Create text with using GString and TextParams
GString str(
    "Scaleform Gfx is a light-weight high-performance rich media vector
    graphics and user interface (UI) engine.");
```

```

GfxDrawTextManager::TextParams params;
params.FontName      = "Arial";
params.FontSize      = 14;
params.FontStyle     = GfxDrawText::Italic;
params.Multiline     = true;
params.WordWrap      = true;
params.HAlignment    = GfxDrawText::Align_Justify;
sz = pdm1->GetTextExtent(str, 200, &params);
GPtr<GfxDrawText> ptxt12 = *pdm1->CreateText(str, GRectF(200, 300, sz),
                                           &params);
ptxt12->SetColor(GColor(0, 0, 255, 130), 0, 1);

```

Once the text is created using the system fonts, call the **BeginDisplay** method of GfxDrawTextManager which has a pointer to the viewport and then call **Display** method of GfxDrawText to render the text on the viewport. After the text is rendered, use the **EndDisplay** method of GfxDrawTextManager to end the display.

```

pdm1->BeginDisplay(vp);
ptxt11->Display();
ptxt12->Display();
pdm1->EndDisplay();

```

## Method 2

This example creates an instance of GfxDrawTextManger which uses a GfxMovieDef pointer to share the fonts contained in GfxMovieDef. The font for the text is obtained from the SWF file loaded in GfxMovieDef. Please refer to [Gfx Integration Tutorial](#) for an understanding of Gfx and loading movie objects.

```

GPtr<GfxMovieDef> pmovieDef = *loader.CreateMovie("drawtext_fonts.swf");
GPtr<GfxDrawTextManager> pdm2 = *new GfxDrawTextManager(pmovieDef);

```

Associate the renderer with GfxDrawTextManager instance.

```

pdm2->SetRenderConfig(pRenderConfig);

```

In differing from the previous example, we display the text using HTML text instead of plain text (but, of course, the HTML might be used in the previous example as well). The dimensions of the text rectangle is calculated by using the **GfxDrawTextManager::GetHtmlExtent** method.

```

// Create HTML text, using fonts from GfxMovieDef
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>адабpA!</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>" ;
```

```
GfxDrawTextManager::TextParams defParams2 =
    pdm2->GetDefaultTextParams();
defParams2.TextColor = GColor(0xF0,0,0,0xFF); //red,alpha = 255
defParams2.Multiline = true;
defParams2.WordWrap = false;
GSizeF htmlSz = pdm2->GetHtmlTextExtent(html, 0, &defParams2);
GPtr<GfxDrawText> ptxt22 =
    *pdm2->CreateHtmlText(html, GRectF(00,100, htmlSz), &defParams2);
```

We also need to create text to demonstrate text animation (rotation and changing color matrix):

```
GSizeF sz;
sz = pdm2->GetTextExtent("Animated");
GPtr<GfxDrawText> ptxt21 = *pdm2->CreateText("Animated",
    GRectF(600, 400, sz));
ptxt21->SetColor(GColor(255, 255, 255, 255));
Float angle = 0;
UInt32 color = 0xFF00;
```

Display the text on the viewport by calling **GfxDrawText::Display**

```
pdm2->BeginDisplay(vp);
ptxt21->Display();
ptxt22->Display();
pdm2->EndDisplay();
```

Make use of the different functions available in GfxDrawText to change color or position the text object.

As an example, rotate and change color of text created by using **SetMatrix** and **SetCxform** methods of GfxDrawText. We do it at the end of the main loop inside the **FxPlayer::Run:**

```
while(1)
{
    . . . .
    // Rotate and animate color for ptxt21
    GfxDrawText::Matrix txt21matrix;
    angle += 1;
    GRectF r = ptxt21->GetRect();
    txt21matrix.AppendTranslation(-r.Left, -r.Top);
    txt21matrix.AppendRotation(angle*3.14159f / 180.f);
```

```

txt2lmatrix.AppendScaling(2);
txt2lmatrix.AppendTranslation(r.Left, r.Top);
ptxt2l->SetMatrix(txt2lmatrix);
GfxDrawText::Cxform txt2lcx;
txt2lcx.M_[0][0] = Float(color & 0xFF) /255.f;
txt2lcx.M_[1][0] = Float((color >> 8) & 0xFF) /255.f;
txt2lcx.M_[2][0] = Float((color >> 16) & 0xFF) /255.f;
color += 32;
ptxt2l->SetCxform(txt2lcx);
}

```

NOTE: Copy the drawtext\_fonts.swf from the Bin/Samples directory into the same directory where the executable is located (if you run the executable manually) or into the project directory (e.g.: Projects/Win32/Msvc90/Samples/DrawText, if run from the Visual Studio 2008). Otherwise, instead of most of the glyphs you will just see rectangles.

Screenshot:

