Autodesk®
Scaleform®

DrawText API

This document describes the DrawText API available in Scaleform 4.4. This API can be used for C++ driven text rendering and formatting outside of the GFx::Movie and ActionScript sandbox.

Author: Artem Bolgar

Version: 2.0
Last Edited: April 22, 2011

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

How to Contact Autodesk Scaleform:

Document	DrawText API
Address	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

1. Introduction 1

1.1. GFx::DrawTextManager 2

1.1.1. Creating GFx::DrawText. 3

1.1.2.Text Rendering 5

1.1.3. Measuring Text Extents 5

1.2. GFx::DrawText 7

1.2.1. Setting and Getting Plain Text Methods 7

1.2.2. Setting and Getting HTML Text Methods 7

1.2.3. Setting Text Format Methods 8

1.2.4. Aligning Text Methods 9

1.2.5. Changing Text Position and Orientation 9

2. Overview of DrawText Sample Code 11

1. Introduction

The Scaleform 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 Scaleform 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, Scaleform 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 Scaleform based menu system with a 3D engine rendered HUD, it is still desirable to use the same font system in both cases.

Scaleform 4.0 includes a C++ driven DrawText API, exposing the Scaleform font rendering and text formatting engines outside of the movie view sandbox. The text API still uses the same Render::Renderer2D interface as the rest of Scaleform player; however, it does not require creation of the GFx::Movie 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 GFx::MovieDef (using the GFx::Loader::CreateMovie method) and then the created MovieDef could be passed as a parameter to GFx::DrawTextManager constructor.

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

- *GFx::DrawTextManager* - Creates instances of the DrawText class. It is used to initialize text rendering and viewport, configure fonts, and to start/stop rendering the text.

- *GFx::DrawText* – 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. GFx::DrawTextManager

The DrawTextManager class instance manages instances of DrawText class; it creates, renders and measures text extents.

There are several ways of creating the DrawTextManager class instance:

1. `DrawTextManager();`

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

Example:

```
Ptr<GFx::DrawTextManager> pDm1 = *new DrawTextManager();
```

2. `DrawTextManager(MovieDef* pmovieDef);`

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

Example:

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

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

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

Example:

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

1.1.1. Creating GFx::DrawText.

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

- `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);`

The methods above create DrawText instances using plain text (null-terminated UTF-8 string, null-terminated Unicode/UCS-2 and Scaleform::String respectively).
`putf8str` : specifies a null-terminated UTF-8 string.
`pwstr` : specifies a null–terminated Unicode/UCS-2 string.
`str`: specifies a Scaleform::String.
`depth`: Specifies the order of drawing.

`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
{
    Color                TextColor;
    DrawText::Alignment  HAlignment;
    DrawText::VAlignment VAlignment;
    DrawText::FontStyle  FontStyle;
    float                FontSize;
    String               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: DrawText::Align_Left (default), DrawText::Align_Right, DrawText::Align_Center, DrawText::Align_Justify.

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

`FontStyle` : specifies font style, such as bold, italic, normal or bolditalic. Possible values are: DrawText::Normal, DrawText::Bold, DrawText::Italic, DrawText::BoldItalic (and DrawText::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 DrawTextManager 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 DrawTextManager’s methods – CreateHtmlText - are similar to CreateText ones described above, but they create the DrawText instances from HTML:

```
// Creates and initialize a DrawText object using specified HTML.
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);
```

Examples:

```
// Creation of DrawText object using plain text with parameters.
```

```
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);

// Creation of DrawText object from HTML.
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.Text Rendering

Text rendering is done in a way similar to rendering GFx::Movie. It is necessary to obtain DisplayHandle (GFx::DrawTextManager) and set the viewport (DrawTextManager::SetViewport). Please refer to GFx documentation for the details about Viewport.

1.1.3. Measuring Text Extents

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

```
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 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 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
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 text extents
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

The DrawText 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 Scaleform::String 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 String& 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.

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

1.2.2. Setting and Getting HTML Text Methods

SetHtmlText method parses UTF-8, UCS-2 or Scaleform::String 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 String& 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.

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

1.2.3. Setting Text Format Methods

There are several methods for setting and getting text format.

`void SetColor(Color 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 DrawText class has various methods for positioning and orienting the text object.

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

Sets view rectangle, coordinates are in pixels.

```
RectF  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.

2. Overview of DrawText Sample Code

This section explains the DrawText API sample implementation that is included in Scaleform 4.1. The DrawText API has been included in Scaleform 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, loading the movie data, and adding a display handle to capture and 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. FxPlayerApp inherits from the FxPlayerAppBase class which is the base class where the window, render thread and graphics are created and initialized.

OnInit method of the FxPlayerApp initializes the DrawText instances, creates the text and applies various text transformations. OnUpdateFrame method applies transformation matrix on the text to create animation.

```
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;
};
```

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

Method 1

In this example, DrawTextManager instance is created using a pointer to GFx::Loader to inherit all states from the loader.

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

Here, we use the system fonts for the text to be displayed and thus GFx::FontProvider is used for the purpose. The FontProvider is created in FxPlayerAppBase::OnInit.

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

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 Scaleform::String.

```
// Create text using plain text and default text parameters.
DrawTextManager::TextParams defParams =
     pDm1->GetDefaultTextParams();
defParams.TextColor = Color(0xF0, 0, 0, 0xFF); // red, alpha = 255
defParams.FontName  = "Arial";
defParams.FontSize  = 16;
pDm1->SetDefaultTextParams(defParams);
```

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

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

The GetTextExtent method of DrawTextManager class measures the dimensions of the text rectangle. And DrawText class can be availed for manipulations on the text such as formating the font, aligning the text or changing the position of text object.

```
// Create text with using String and TextParams
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);
```

Once the text is created using the system fonts, set viewport for the DrawText, capture current DrawText state and add DrawText DisplayHandle to the `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);
```

Method 2

This example creates an instance of DrawTextManger which uses a MovieDef pointer to share the fonts contained in MovieDef. The font for the text is obtained from the SWF file loaded in MovieDef. Please refer to Scaleform Integration Tutorial for an understanding of Scaleform and loading movie objects.

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

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 **DrawTextManager::GetHtmlExtent method**.

```
      // Create HTML text, using fonts from ScaleformMovieDef
      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>";

      Scaleform::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);
```

We also need to create text to demonstrate text animation:

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

and filters:

```
SizeF sz;
Ptr<Scaleform::DrawText>  pglowTxt
pglowTxt = *pDm2->CreateText("Glow", RectF(800, 350, SizeF(200, 220)));
pglowTxt->SetColor(Render::Color(120, 30, 192, 255));
pglowTxt->SetFontSize(72);
Scaleform::DrawText::Filter glowF(Scaleform::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);
```

Set viewport for the DrawText, capture current DrawText state and add DrawText DisplayHandle to the RenderThread:

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

As an example, rotate and change color of text created by using SetMatrix and SetCxform methods of DrawText. We do it inside the FxPlayerApp::OnUpdateFrame:

```
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);
```

```
    }
```

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: