

RAGE

Tutorials

Font/Text Rendering

Software & Documentation © The RAGE Team 2005

Version Relevance: JGE Release 0.2

CONTENTS

| | |
|--------------------------------------|---|
| Are you reading the right tutorial? | 3 |
| Required Imports | 3 |
| Required Properties | 3 |
| Font Basics | 4 |
| Using Text Elements | 6 |
| Creating a FontPlatform | 7 |
| Creating your own Font with Textures | 8 |

ARE YOU READING THE RIGHT TUTORIAL?

This tutorial is intended to show you how to use the font and text classes to render single-line text on a 2D or 3D layer. It assumes you have read the tutorials for 2D rendering basics.

It will also demonstrate how to construct your own font's using textures, recommended ways of using getting fonts from the system or file into the FontManager.

This document does not cover the inner workings of the font system.

REQUIRED IMPORTS

Font Package: `jge.widget.font`

Text Package: `jge.widget.text`

Render Packages used in examples:

`jge.hal.render.core`

`jge.hal.render.core2d`

Resource Packages used in examples:

`jge.kernel.resource`

REQUIRED PROPERTIES

| <i>Property</i> | <i>Description</i> | <i>Example</i> |
|---------------------------------|--|---|
| <code>class.FontPlatform</code> | Implementation of <code>jge.widget.font.FontPlatform</code> to use in JGE. | <code>plugin.widget.font.ExampleFontPlatform</code> |

FONT BASICS

USING FONTS AND GLYPHS

Fonts are a group of glyphs, one for each character supported by the font. A glyph stores an image representation of the character in the image format used by the renderer.

A glyph also stores the bounding area of the character as well as the area it would take up if it was to be rendered. The bounding area is typically smaller than the space it renders to, so that characters sit comfortably next to each other. In some fonts characters bounding areas will cause characters to overlap – common in scripting fonts.

A string of text is typically created by merging the images of different glyphs together to form a single image. The image would be rendered by placing it on a single sprite. If you found it to be more suitable, you could also render the text by placing each glyph on its own sprite, and rendering them side-by-side. This latter technique is not recommended due to the performance difference.

See example code on the next page.

USING THE FONT MANAGER

The font manager provides an interface through to the platform's source of fonts. Upon asking the FontPlatform for a font, if it cannot find it in its register, it will ask the FontPlatform for the font. The FontPlatform is required to ALWAYS return a Font object, and to use a default font if the requested is not available.

If you register your own Font with the FontManager and another Font in the FontPlatform exists with that name, your Font will be used instead of the other.

The FontManager also contains a set of characters, which are not used directly in the FontManager. This character set is available to loaders/platforms as a base for creating fonts. The programmer can change the character set to limit the glyphs created, which can improve performance when only a small set of characters are required. (It is not required that a platform implements using this feature.)

See example code on the next page.

EXAMPLE CODE

```
// setup the font manager for our use
// we only need a limited number of characters
FontManager fm = FontManager.getInstance();
fm.setCharacterSet("helloworld");

// get the Font we want,
// if it doesn't exist we will get a default font
Font arial = fm.getFont("Arial");

// get a texture for the string 'helloworld'
Texture helloworld =
    arial.getTextureString("helloworld");

// we can now use the texture for rendering on a sprite,
// 3D object, etc
```

USING TEXT ELEMENTS

The Text element allows for simpler rendering of text using fonts.

A Text element stores the font, the text to be rendered and what height to render the text at. The Text class itself is an interface, providing all the functionality except the actual rendering of the text. Currently only a single implementation of Text is provided. It renders the string as a single texture on a single sprite.

A Text element does not have a concept of width, only height. Width is calculated automatically based off the length of the string.

Functions are provided in the Text class to align text automatically. See the JavaDocs for more information on these.

Note: Since most text uses alpha areas in their textures, alpha blending must be enabled for text to appear without a background.

EXAMPLE CODE

```
FontManager fm = FontManager.getInstance();
Renderer r = Renderer.getInstance();
RenderPlatform rp = r.getPlatform();
LayerManager lm = r.getLayerManager();

// setup the font manager for our use
// we only need a limited number of characters
fm.setCharacterSet("helloworld");

// get the Font we want,
// if it doesn't exist we will get a default font
Font arial = fm.getFont("Arial");

// create the text object with the font
Text hwText = new StringText(Font arial);

// set the text we want rendered
hwText.setText("helloworld");

// enable alpha blending so the text doesn't have a
// background
rp.setAlphaBlendFunction(SrcBlendOp.SRC_ALPHA,
    DstBlendOp.ONE_MINUS_SRC_ALPHA);
rp.setAlphaBlendMode(true);

// setup the 2D layer to put the text on
Layer layer = new Layer2d();
layer.addElement(hwText);
lm.addLayer(layer);
lm.addLayer(new ClearLayer(0.0f, 0.0f, 0.0f, 1.0f));
```

CREATING A FONTPLATFORM

A FontPlatform provides access to a platforms library of fonts. It needs to provide three basic functions:

```
+getFont(name:String):Font
```

Gets the Font on the system with the specific name. If the font does not exist, this function MUST return a default font. The Font should be setup with all the characters that are available.

```
+getFont(name:String, characters:String):Font
```

Gets the Font on the system with the specific name. If the font does not exist, this function MUST return a default font. The Font should be setup with only the characters specified in the characters String..

```
+getFontNames():String[]
```

Get the names of fonts available on the platform.

CREATING YOUR OWN FONT WITH TEXTURES

It is possible for you to create your own Font using a series of Textures for each Glyph. You can design your own font in a graphics application and load each character.

Step 1: Create the graphics or acquire them. Each character needs to be in its own Image.

Step 2: Load them into your application using a resource loader.

```
ResourceManager rm = ResourceManager.getInstance();
Image imageA = (Image)rm.getResource("a.tga");
```

Step 3: Create a Font object that has a name, but not any glyphs yet.

```
Font myFont = new Font("myFont");
```

Step 4: Create a Glyph object for each character. You need to assign each Glyph object a character, Image and bounding area on construction. The bounding area will in most cases be (0,0) to (width, height), where width and height are the size of the image. (You can get the size of the image by calling `getSizeX` and `getSizeY` on the Image object.)

```
Glyph a = new Glyph(
    'a',
    imageA,
    new Rectangle(
        0, imageA.getSizeX(),
        0, imageA.getSizeY()
    )
);
```

Step 5: Add each glyph to the Font object.

```
myFont.setGlyph(a);
```

Step 6: Register the Font with the FontManager.

```
FontManager fm = FontManager.getInstance();
rm.registerFont(myFont);
```

You can now use the font by calling `fm.getFont("myFont")`.

