TinyLine for Google App Engine

Version 2.5

Table of Contents

1. Introduction

The popularity of Google App Engine Java platform is on the rise. There are lot more applications being deployed on this platform. At the same time, the graphics support on Google App platform is very limited.

There is a need for the richer graphics on Google App Engine. Especially if we are talking about cloud computing when data is being collected and processed in the cloud, there should be an option to visualize this data in the cloud.

2. Google App Engine Java

2.1 Overview

Google App Engine Java is cloud computing technology. It allows Java developers to build scalable web applications using standard Java technologies and run them on Google's infrastructure [R1].

Google App Engine Java provides:

- Java 6 JVM
- Java Servlets
- JDO, JPA, JavaMail, and JCache standard interfaces to the App Engine datastore and services

2.2 Java Runtime Environment

Google App Engine runs Java applications using the Java 6 JVM. Google App Engine uses the Java Servlet standard for web applications.

A web application has a standard WAR directory structure that can have servlet classes, JavaServer Pages (JSPs), static files and data files, the deployment descriptor and other configuration files.

The Java 6 JVM runs web applications in a Sandbox environment. The Sandbox ensures a web application does not interfere with the performance and scalability of other web applications.

A web application, for example, cannot create new threads, write to local file system, make network connections, use JNI or load native libraries.

2.3 Datastore and Services

Google App Engine provides scalable services that web applications can use to store persistent data, access over the network, and perform other tasks.

Google App Engine provides User service API for user authentication. A user that already has a Google account (such as a GMail account) can use that account with an application.

An application can detect if the current user is signed in, and can access the user's email address. Google App Engine also provides the low-level API to generate sign-in and sign-out URLs.

User service API can return the current user's information as a User object. User objects could be saved in the datastore.

The Google App Engine Memcache provides fast, transient distributed storage for caching the results of datastore queries and calculations. It implements JCache API (JSR 107).

The Google App Engine Datastore is a reliable, scalable persistent storage of data. It supports standard Java interfaces: JDO 2.3 and JPA 1.0. These interfaces are implemented using DataNuclueus Access Platform.

The Google App Engine URL Fetch service allows to access resources over the network and to communicate with other hosts using the HTTP and HTTPS protocols.

The Google App Engine Mail service allows sending email messages on behalf of the application user or application administrators. It uses JavaMail API for sending email messages.

The Google App Engine Images service allows transforming and manipulating image data in several formats.

2.4 Sandbox

Google App Engine runs Java web application using Java 6 JVM in a Sandbox environment.

When Google App Engine receives a request for a web application, it invokes the servlet that corresponds to the URL. It uses the Java Servlet API to provide the request data to the servlet, and accept the response data.

Google App Engine uses many web servers to run web application. Any request may be routed to any server. It could be not the same server that handled a previous request from the same user.

The Sandbox environment allows Google App Engine to route requests for applications across various web servers, and to prevent one application from interfering with another [R2].

Restriction	Alternative
Threads	Modify the thread state
Direct network connections	URLConnection
Direct file system writes	Use memory, memcache, datastore
Java2D	Images API, Software rendering
Native code	Pure Java libraries

2.5 Server side graphics

Google App Engine supports client side graphics via Google Web Toolkit (GWT) and Web Browser capabilities (like Canvas, JavaScript or SVG native support).

The server side graphics in GAE is very limited. It supports some Images API to manipulate image data while Java2D is not allowed.

There is a need for the server side graphics on Google App Engine. Especially if we are talking about cloud computing when data is being collected and processed in the Cloud, there should be an option to visualize this data in the Cloud.

There are a lot of people that feel comfortable with static data visualization. They would like to copy and paste their charts, diagrams or over graphics as still images in PNG or JPEG format into their MS Office Word, Excel or Power Point program.

They would like to place their own caption or memos using simple raster graphics editor, etc.

They would like to see a static preview or capture of a dynamic presentation before they click on it.

Because Java2D is not allowed software rendering should be used for server side graphics in Google App Engine. Software rendering it is exactly what TinyLine does!

3. TinyLine

TinyLine includes TinyLine 2D and TinyLine SVG [R3].

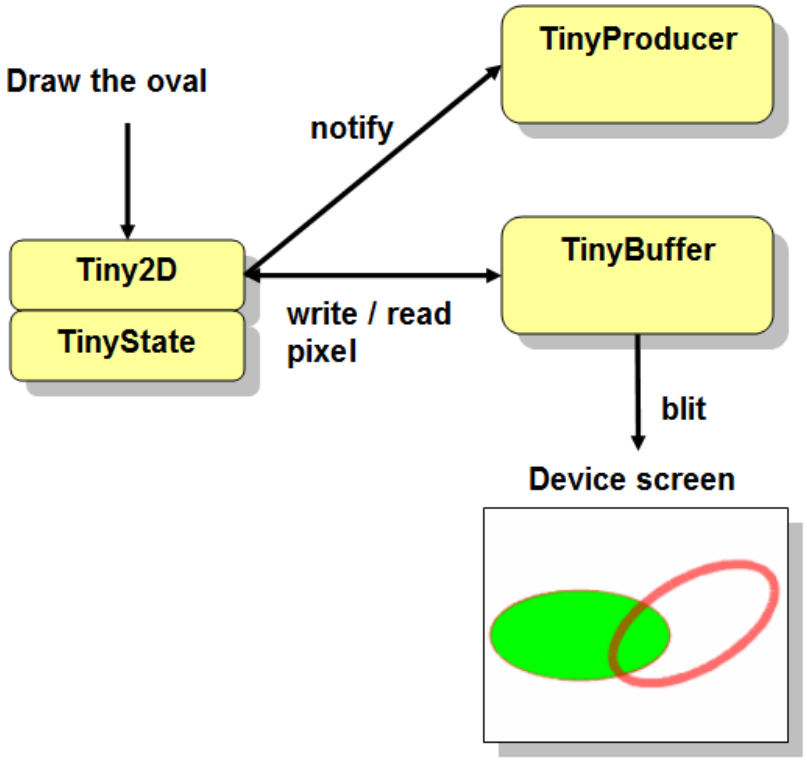
3.1 TinyLine 2D

TinyLine 2D implements a mobile 2D graphics engine for Java platform.

Key features:

- Small footprint
- Fast fixed-point numbers mathematics
- Paths, basic shapes and texts drawings
- Hit tests for paths and texts
- Solid color, bitmap, pattern, gradient (radial and linear) paints
- Fill, stroke and dash
- Affine transformations
- Outline fonts
- Left-to-right, right-to-left and vertical text layouts
- Antialiasing
- Opacity

Drawing pipeline



The TinyLine 2D drawing pipeline has four components: Tiny2D, TinyState, TinyBuffer and TinyProducer.

The TinyLine 2D rendering process is controlled through the Tiny2D object and its state attributes. The state attributes, such as line styles and transformations are applied to graphic objects when they are rendered. The collection of state attributes associated with a Tiny2D is referred to as the graphics state TinyState.

The basic drawing process is the same for any graphics element:

1. Specify the target surface TinyBuffer.
2. Specify the appropriate attributes for the graphics by setting the graphics state attributes in the TinyState object.
3. Define the shape or text you want to draw.
4. Use the Tiny2D object to render the shape, paths or text, by calling one of the Tiny2D rendering methods.

During the rasterization, the Tiny2D object reads the graphics state attributes TinyState and applies them to the drawing process of the graphics primitives. The rasterization target is the TinyBuffer object (also named as "canvas") where all is drawn. When the drawing process has completed the TinyProducer is

notified that new pixels on the TinyBuffer "canvas" are ready to be send onto a device screen.

The painters model

TinyLine 2D uses the painter's model for its imaging. In the painter's model, each successive drawing operation applies a layer of "paint" to an output "canvas" - TinyBuffer object. The paint on the pixels buffer (TinyBuffer object) can be modified by overlaying more paint through additional drawing operations. This model allows you to construct sophisticated images from a small number of primitives.

When the paint is not completely opaque the result on the pixels buffer is defined by the (mathematical) rules of the *simple alpha blending*.

Coordinate spaces

TinyLine 2D supports three coordinate spaces: character space, user space and device space. Transformations between coordinate spaces are defined by transformation matrices, which can specify any linear mapping of two-dimensional coordinates.

Graphics elements

TinyLine 2D supports two basic types of graphics elements that can be rendered onto the TinyBuffer object:

- Shapes, which represent some combination of straight line and curves
- Text, which represents some combination of character glyphs

As about raster images, a raster image is presented via the TinyColor class as array of values that specify the paint color and opacity (alpha).

Shapes and text can be filled and stroked. Each fill and stroke operation has its own opacity settings; thus, you can fill and/or stroke a shape with a semi-transparently drawn solid color, with different opacity values for the fill and stroke operations.

Text and fonts

In TinyLine 2D texts are rendered just like shapes. Therefore, coordinate system transformations, painting - all TinyState graphics state attributes apply to text in the same way as they apply to shapes or paths.

Additional text attributes include such things like the writing direction (text layout), font specification (TinyFont) and painting attributes which describe how exactly to render the characters.

The TinyFont class includes the information necessary to map characters to glyphs, to determine the size of glyph areas and to position the glyph area.

Colors

TinyLine 2D supports the following built-in types of paint that can be used in fill and stroke operations using the ARGB color space:

- Single color
- Gradients (linear and radial)
- Patterns (raster images)

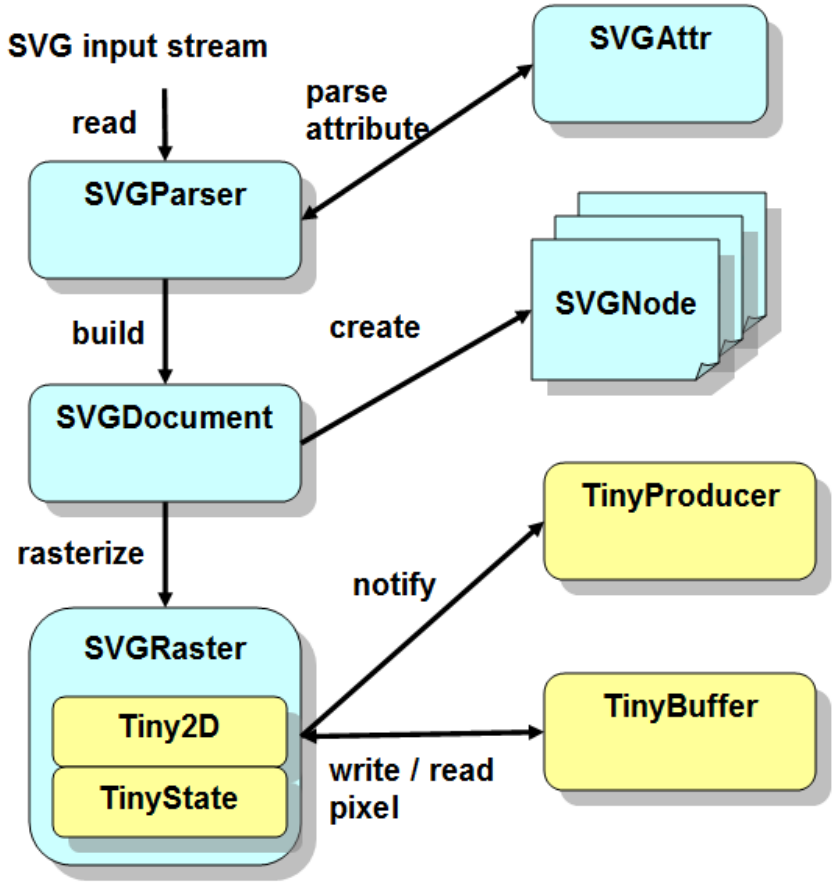
3.2 TinyLine SVG

TinyLine SVG uses (includes) TinyLine 2D rendering capabilities and implements SVG Tiny 1.1+ engine for Java platform.

Key features:

- SVG Tiny 1.1+ engine
- Supports SVG fonts, raster image and text elements, paths
- Supports SMIL animations and events
- Allows both textual and gzipped SVG streams
- Compact code
- Easy to use API

SVG pipeline



During the parsing process SVGParser callbacks build internal OM (SVGNode tree) from the input SVG stream. The SVGDocument has the SVGNode tree root as its member.

The SVGRaster translates SVGNode objects into graphics primitives and draws them (rasterization process) using Tiny2D onto the TinyBuffer object.

The TinyProducer interface gets a notification that new pixels of the TinyBuffer object are ready to be send onto physical screen.

The SVGRaster class rasterizes the tree of SVGNodes onto the TinyBuffer. Each SVGNode "paint" operation draws over some area of the pixel buffer. When the area overlaps a previously painted area the new paint partially or completely obscures the old. When the paint is not completely opaque, the result is defined by the simple alpha blending rules.

Nodes in the SVG tree have an implicit z drawing order. Subsequent nodes are painted on top of previously painted nodes.

When a graphic object is rendered, the geometry, image, and attribute information are combined to calculate which pixel values must be changed.

3.3 TinyLine for GAE

TinyLine is extremely portable across Java flavors because it uses a small set of standard Java SDK classes

- java.lang.Exception
- java.io.InputStream
- java.lang.Object
- java.lang.System
- java.lang.Throwable

It made TinyLine run on Google App Engine platform without any efforts.
Because the Google App Engine Sandbox environment does not allow AWT Image class instead of drawing TinyBuffer on an AWT Image, we should serialize TinyBuffer object into PNG image data.
For SVG Tiny support, we should be able to load (read) PNG image data into TinyBuffer object.

Raster images

Neither Google App Engine nor TinyLine provide reading and writing raster images API.

SVG Tiny should support at least one of the following formats PNG or JPEG. It seems that PNG support is a lot easier than JPEG.

The PNG reading and writing classes are in com.tinyline.appengine.image package:

```
com.tinyline.appengine.ImageWriter
com.tinyline.appengine.AppImageLoader
```

Here ImageWriter converts a TinyBuffer object into a stream of bytes in PNG image format. AppImageLoader reads an input stream in PNG image format and converts it to a TinyBuffer object.

2D and SVG Canvases

TinyLine for Google App Engine classes includes

```
com.tinyline.appengine.AppTiny2DProducer
com.tinyline.appengine.AppTiny2DCanvas
com.tinyline.appengine.AppViewerCanvas
```

AppTiny2DProducer is a Google App Engine implementation of the TinyProducer interface. TinyProducer provides an interface for objects which can produce the image data from the TinyBuffer object (pixels buffer)

```
/**
 * Determine if there is a consumer that is currently
 * interested in data from this TinyProducer.
 */
public boolean  hasConsumer();

/**
 * Send a rectangular region of the buffer of pixels to any
 * consumer that is interested in the data from
 * this TinyProducer.
 */
public void     sendPixels();

/**
 * The imageComplete method is called when the TinyProducer is
 * finished delivering all of the pixels to a consumer.
 */
public void     imageComplete();
```

AppTiny2DCanvas is the Google App Engine implementation of the Tiny2D Canvas.

AppViewerCanvas is the Google App Engine implementation of the SVG Tiny Canvas. It implements ImageLoader interfaces

```
/**
 * Returns a TinyBuffer for the given image URL or path.
```

```
     */
    public TinyBuffer createTinyBuffer(TinyString imgRef);
    /**
     * Creates a TinyBuffer which decodes the image stored in the
specified
     * byte array, and at the specified offset and length.
     */
    public TinyBuffer createTinyBuffer(byte[] imageData, int
imageOffset, int imageLength);
```

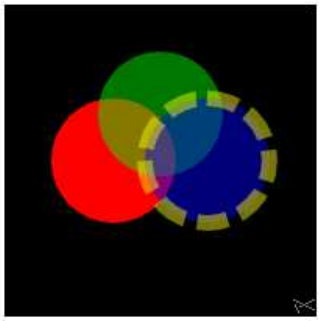
4. Basic demos

4.1 TinyLine 2D Examples

TinyLine for Google App Engine comes with a list of 2D examples in com.tinyline.appengine.demos.tinylinegae.tiny2d package.

They show how Java developers could use TinyLine and Google App Engine for the server side 2D graphics. See [R4]. Here are just some of them.

The Colors example shows how to draw with a single ARGB color.



The Patterns example shows how to draw with patterns (bitmaps) colors. Also, it shows how to draw on different targets and store and restore the graphics state object.

The Gradients example shows how to draw with gradients colors.

4.2 SVG Thumbnail images

TinyLine brings SVG server side rendering to Google App Engine platform.

This example demonstrates how to build a very basic SVG Thumbnail service using TinyLine and Google App Engine. See [R4].

Thumbnails are used to help users in recognizing and organizing images. Most up to date operating systems, desktop environments, image-organizing programs and search engines use thumbnails.

Thumbnail images are convenient way to show SVG files previews to users before they decide to download or play real SVG files.

Thumbnail images could be used for SVG listings, SVG file directories or for SVG Result Lists of search engines.

SVG drawing service fetches an SVG Tiny document from the Internet and exports a PNG image of the given size - a Thumbnail image.

Function generateImage () in com.tinyline.appengine.demos.tinylinegae.TinyLineServlet does the job:

```
    /**
```

```
     * Generates an image that corresponds to the request.
     */
    private byte[] generateImage(HttpServletRequest request)
                throws IOException, ServletException
    . . .

    /* Initialize SVG canvas */
    viewerCanvas = new AppViewerCanvas(w, h);
    /* Load the default SVG font if needed */
    if (SVGDocument.defaultFont == null)
    {
        SVGDocument doc = viewerCanvas

        .loadSVG("http://tinyline.com/svgt/download/guide/example/images
/helvetica.svg");
        SVGFontElem font = SVGDocument.getFont(doc,
                        SVG.VAL_DEFAULT_FONTFAMILY);
        SVGDocument.defaultFont = font;
    }
    /* Load the SVGT image */
    viewerCanvas.goURL(urlStr);
    /* Return the corresponding PNG image */
    return imageWriter.generateImage(viewerCanvas.t2d.getTarget());
```

Here are the sample Thumbnail images in generated from the same SVG file:

4.4 Barcode example

TinyLine for Google App Engine helps to port other graphical Java applications to the Google App Engine platform.

It is particularly easy to port Java applications that are already using SVG. As example, we selected Barcode4J library [R5].
Barcode4J is has a wide range of barcodes it supports, it is written in Java and it has a modular design. Therefore, we expected to spend fewer hours to port it.

Here are some basic facts about barcodes. A barcode is an optical representation of data. Barcodes that present data in lines and spaces between them are called 1D barcodes. There are also 2D barcodes that represent data as squares, dots and other patterns.

Barcode example classes:

```
com.tinyline.appengine.demos.barcode.BarcodeServlet
com.tinyline.appengine.demos.barcode.SVGTinyCanvasProvider
```

BarcodeServlet is the main entry of the Barcode example application. It reads parameters from HTTP requests, creates all Barcode4J classes and sends HTTP responses.

SVGTinyCanvasProvider class implements SVG Tiny producer for Barcode4J library.

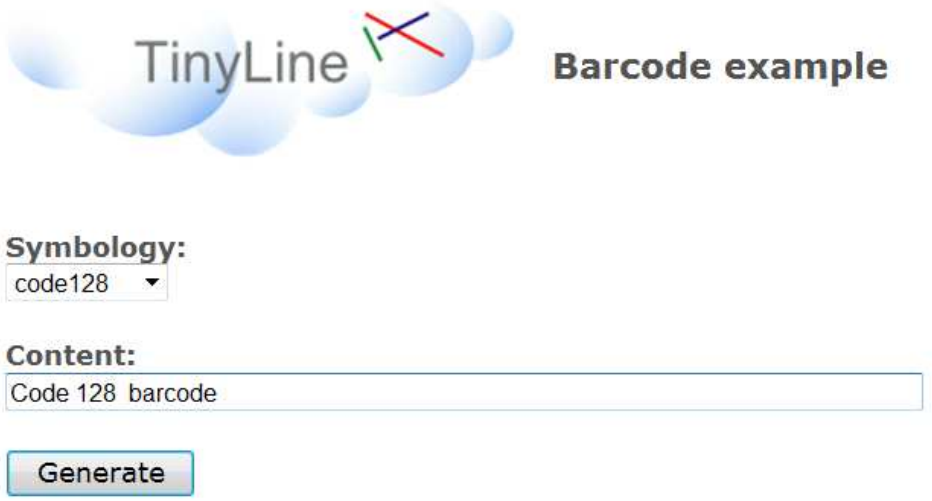
SVGTinyCanvasProvider supports the SVG Tiny requirements:

- SVG Tiny does not support styling with CSS
- SVG Tiny supports presentation attributes
- SVG Tiny supports user units only, except for the 'width' and 'height' attributes on the outermost 'svg' element where percentage units are also supported.

For example, deviceFillRect () function that draws 'bars' uses presentation attributes and converts CSS units to user units by calling toTiny() function:

```
public void deviceFillRect(double x, double y, double w, double h)
{
    Element el = createElement("rect");
    el.setAttribute("x", toTiny(x));
    el.setAttribute("y", toTiny(y));
```

```
        el.setAttribute("width", toTiny(w));
        el.setAttribute("height", toTiny(h));
        detailGroup.appendChild(el);
    }
```



Here are the sample barcodes generated by the Barcode example.
(Code128 barcode, PDF417 barcode and UPCA barcode)

5. Resources

[R1] *Google App Engine Java Overview*
http://code.google.com/appengine/docs/java/overview.html

[R2] *Google Developer Days Brazil 2009 - Java Appengine*
http://www.slideshare.net/chanezon/google-developer-days-brazil-2009-java-appengine

[R3] *TinyLine* http://www.tinyline.com

[R4] *TinyLine GAE: Basic Demos* http://tinylinegae.appspot.com/

[R5] *Barcode4J library* http://barcode4j.sourceforge.net