MilSym Documentation

1 OVERVIEW

1.1 FEATURES

The Android version of the Mil-Std Symbology Renderer supports rendering of single & multi point symbology. Single points as a BufferedImage or SVG. Multipoints as GeoJSON or a collection of points and other information that will enable the user to render.

Much of the API is the same between Android, Java and JavaScript. Differences will be noted.

1.2 System Requirements

Android Targeting Minimum Android API 24.
Java Targeting Java 8.

2 RENDERING

2.1 CONFIGURING THE RENDERER FOR YOUR NEEDS

The “RendererSettings” object will let you set some default rendering values. It is accessible at "armyc2.c5isr.renderer.utilities.RendererSettings”.

```
RendererSettings.getInstance().setCacheEnabled(true); //Does not apply to JavaScript or SVG Rendering

//Initialize the Icon Renderer.
C5Ren.initialize(path); //JavaScript only
MilStdIconRenderer mir = MilStdIconRenderer.getInstance();
mir.init(this); //Android only
```

2.2 SINGLEPOINT ICON SYMBOLOGY

Singlepoint rendering is provided by the MilStdIconRenderer object
You would typically render a single point via the code snippet below:

```
//Android and Java ports only:

ImageInfo ii = armyc2.c5isr.renderer.MilStdIconRenderer.Render("110310000012080000000000000000", modifiers,
attributes);

//All 3 ports:

SVGSymbolInfo si = armyc2.c5isr.renderer.MilStdIconRenderer.RenderSVG("110310000012080000000000000000",
modifiers, attributes);
```

3.2.1 SETTING MODIFIERS

“modifiers” is a Map which can contain Mil-Std modifiers. The modifiers can be set like this:

```
import armyc2.c5isr.renderer.utilities.Modifiers;

Map<String, String> modifiers = new HashMap<String, String>();
modifiers.put(Modifier.C_QUANTITY, "10");
modifiers.put(Modifiers.H_ADDITIONAL_INFO_1, "H");
modifiers.put(Modifiers.H1_ADDITIONAL_INFO_2, "H1");
//or like this for Single Point Tactical Graphics:
modifiers.put(Modifiers.H2_ADDITIONAL_INFO_3, "H2")
//or like this for Multi Point Tactical Graphics:
//(comma delimited for modifiers with multiple values)
modifiers.put(Modifiers.AM_DISTANCE, "1000,2000,3000");
```

3.2.2 SETTING ATTRIBUTES

Attributes will override any defaults set in RendererSettings.

```
Map<String, String> attributes = new HashMap<String, String>();
attributes.put(MilStdAttributes.PixelSize, "60");
attributes.put(MilStdAttributes.KeepUnitRatio, "true");
```

MilStdAttributes.java contains all the modifier key constants for the attributes that can be applied to rendering.

3.2.2.1 SIZE

PixelSize = Default size 35. This is the size of the core symbol (not including modifiers) which will fit within a 35x35 pixel space.

Depending on screen size and resolution of your target platform, you may want to change this value.

3.2.2.2 LINECOLOR / FILLCOLOR

LineColor and FillColor, if you want to override the default affiliation colors, do something like: “#FF0000”. You can add transparency by using an alpha value: "#80FF0000".

3.2.2.3 KEEPUNITRATIO

KeepUnitRatio If true (defaults true), the symbols will have proper proportions in relation to each other (see table VIII in the MilStd 2525C). Hostile airspace, with a size of 35, will end up being 25.667 wide by 30.333 high, hostile unit will be 33.6x33.6 ((35/1.5)*1.44). If false, image will fill the pixel space as much as it can without distorting the shape.

3.2.2.4 ICON

ICON=true, will result in a symbol that is stripped of display & text modifiers so that you get just the core symbol. This is useful for use as tree-node icons. If you had all the detail, the symbols would be so small it would be hard for the user to determine what symbol they're looking at.

3.2.3 GETTING YOUR IMAGE (ImageInfo) (Android and Java)

Looking back at the call to render, you’ll see you get an object back. This object contains the image along with some information about the image.

```
ImageInfo ii = armyc2.c5isr.renderer.MilStdIconRenderer.RenderIcon("SUGDUSAT----***", modifiers, attributes);
```

What is returned is the ImageInfo object and it has the following functions available:

3.2.3.1 getImage()

Returns the image. (Android: Bitmap; Java: Bufferedimage)

3.2.3.2 getSymbolCenterPoint()

Returns a point object that represents where the image should be centered if rendered on a coordinate based map. (Android: android.graphics.Point; Java: java.awt.Point; JavaScript Pont2D)

3.2.3.3 getSymbolBounds()

Returns a rectangle object that represents to area in the image that the core symbol (not including any modifiers) exists. (Android: Rect; Java: Rectangle2D; JavaScript Rectangle2D)

3.2.3.4 getImageBounds()

Returns a rectangle object that represents the size of the entire image. (Android: Rect; Java: Rectangle2D; JavaScript Rectangle2D)

3.2.3.6 getSquareImageInfo()

Returns an ImageInfo object where space is added as necessary to make the image a square so that it will fit nicely in a tree node or for any other situation where consistent image size is important. So if you draw a friendly ground unit and the resultant image is 16x10, this function will return a version that is 16x16.

3.2.4 GETTING YOUR IMAGE (SVGSymbolInfo) (Java and JavaScript)

Looking back at the call to render, you’ll see you get an object back. This object contains the SVG string along with some information about the image.

```
SVGSymbolInfo si = armyc2.c5isr.renderer.MilStdIconRenderer.RenderSVG("110310000012080000000000000000",
modifiers, attributes);
```

3.2.4.1 getSVG()

Returns the string value of the SVG.
The <desc> tag is formatted like: <desc> centerX centerY symbolBoundsX symbolBoundsY symbolBoundsWidth symbolboundsHeight imageBoundsX imageBoundsY imageBoundsWidth imageboundsHeight </desc>

The same information is also available in the more formal <metadata> tag.

```
sample output:
```

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="50" height="33" viewBox="0 0 50 33">
<desc>25 16 0.0 0.0 50.0 33.561641693115234 0.0 0.0 50.0 33.0</desc>
<metadata>
<symbolID>11031000001204010000000000000000</symbolID>
<anchor>25 16</anchor>
<symbolBounds>0.0 0.0 50.0 33.561641693115234</symbolBounds>
<imageBounds>0.0 0.0 50.0 33.0</imageBounds>
</metadata>
<g transform="translate(-16.92904107009099,-37.46575314551592) scale(0.1369863,0.1369863)"><g id="0_310_0"><rect fill="#80E0FF" height="240" id="_x3C_path_x3E_" stroke="#000000" stroke-width="5" width="360" x="126.082" y="276"/></g><g id="10120401_1"><polyline fill="none" id="symbol" points="126.5,515.5 306,280.5 485.5,515.5 " stroke="#000000" stroke-width="5"/><path d="M246,349.705c-60,0-60,92.545,0,92.545h120c60,0,60-92.545,0-92.545H246 L246,349.705z" fill="none" stroke="#000000" stroke-width="5"/></g></g>
</svg>

NOTE: For SVG output, the default san-serif font on an Android device will be different than san-serif on a windows device. Since these fonts have different measurements, it's possible an SVG generated on Android may not look quite right on a windows machine.

3.2.4.2 getSVGDataURI()

Returns a base64 string representation of the SVG ("[base64 string]")

3.2.4.3 getSymbolCenterPoint()

Returns a point object that represents where the image should be centered if rendered on a coordinate based map. (Android: android.graphics.Point; Java: java.awt.Point; JavaScript Point2D)

3.2.4.4 getSymbolBounds()

Returns a rectangle object that represents to area in the image that the core symbol (not including any modifiers) exists. (Android: Rect; Java: Rectangle2D; JavaScript Rectangle2D)

3.2.4.5 getImageBounds()

Returns a rectangle object that represents the size of the entire image. (Android: Rect; Java: Rectangle2D; JavaScript Rectangle2D)

3.3 MULTIPOINT SYMBOLOGY

Multipoint rendering is provided by the armyc2.c5isr.web.render.WebRenderer object
KML (deprecated, use GeoJson going forward) & MilStd Symbol rendering example:

(Also, symbology standard will be going away as a parameter as that value is baked into the 2525D symbol code)

```
//SECTOR RANGE FAN EXAMPLE//////////////////////////////

            String id = "id";
        String name = "name";
        String description = "description";
        String symbolCode = "10032500002422000000";
        String controlPoints = "8.40185525443334, 38.95854638813517 15.124217101733166, 36.694658205882995
18.49694847529253, 40.113591379080155 8.725267851897936, 42.44678226078903 8.217048055882143,
40.76041657400935";
        String altitudeMode = "absolute";
        double scale = 5869879.2;
        String bbox = "5.76417051405295, 34.86552015439102, 20.291017309471272, 45.188646318100695";

        Map<String, String> modifiers = new HashMap<String, String>();
        Map<String, String> attributes = new HashMap<String, String>();
        attributes.put(MilStdAttributes.LineColor, "ffff0000");


MilStdSymbol flot = WebRenderer.RenderMultiPointAsMilStdSymbol(id, name, description, symbolCode, controlPoints,
altitudeMode, scale, bbox, modifiers, attributes);

//[MilStdSymbol]() is an object that contains the information necessary to render.

//Info needed for rendering/////////////////
//ArrayList<ShapeInfo> sShapes = flot.getSymbolShapes();
//ArrayList<ShapeInfo> mShapes = flot.getModifierShapes();
//message.append("Symbol Shape Count: " + String.valueOf(sShapes.size()));
//message.append("Modifier Shape Count: " + String.valueOf(mShapes.size()));

//line info
//sShapes.get(0).getPolylines();//Arraylist<Point2D>
//sShapes.get(0).getLineColor();
//sShapes.get(0).getFillColor();
//sShapes.get(0).getStroke().getLineWidth();
//sShapes.get(0).getStroke().getDashArray();
//Renderer must be set to use dash array for dashed lines.
//Otherwise, the renderer will break the lines into multiple pieces to create the dashed line effect.
//ANDROID
//sShapes.get(0).getShader()//returns a BitmapShader that should be used to tile fill the shape.
//JAVA
//sShapes.get(0).getTexturePaint()//contains a BufferedImage. Use this, if present, to fill the shape assuming
you can't use the BitmapShader in your implementation.\
//ANDROID & JAVA
//sShapes.get(0).getPatternFillImage()//returns a Bitmap or Buffered Image depending on the platform.
////typically used for weather graphics.
//JavaScript
//sShapes.get(0).getPatternFillImage()//returns an SVG String.

//Modifier info
//RendererSettings.getMPModifierFont();//returns a Paint object
//flot.getLineColor (typically you want the modifier text to match the line color of the symbol)
//mShapes.get(0).getModifierString();
//mShapes.get(0).getModifierPosition();
//mShapes.get(0).getModifierAngle();

//Modifier shapes can also contain a Bitmap instead of text for cases line Mined Areas that show a mine icon in
the center of the area.
//In this case the getModifierString() method will return null.
//mShapes.get(0).getModifierImage();//JavaScript will be an SVG string.
/////////////////////////////////////////

//You can also get the symbol as GeoJSON with the RenderSymbol function. The second to last parameter is the
format for the string output to return.
//0 will return KML, while 2 will return GeoJSON.
String geoJSON = WebRenderer.RenderSymbol(id, name, description, symbolCode, controlPoints, altitudeMode, scale,
bbox, modifiers, attributes, 2);
```

3.3.1 REQUIRED PARAMETERS FOR MULTIPOINT SYMBOLOGY

3.3.1.1 ID (FOR 3D & 2D)

id = a unique identifier used to identify the symbol by Google map. The id will be the folder name that contains the graphic.

3.3.1.2 NAME (FOR 3D & 2D)

name = a string used to display to the user as the name of the graphic being created.

3.3.1.3 DESCRIPTION (FOR 3D & 2D)

description = a brief description about the graphic being made and what it represents.

3.3.1.4 CONTROL POINTS (FOR 3D & 2D)

controlPoints = the vertices of the graphics that make up the graphic. They are passed in the format of a string, using decimal degrees separating lat and lon by a comma, separating coordinates by a space. The following format shall be used "x1,y1 [xn,yn]...".

3.3.1.5 ALTITUDE MODE (FOR 3D)

altitudeMode = indicates whether the symbol should interpret altitudes as above sea level or above ground level. Options are "clampToGround", "relativeToGround" (from surface of earth), "absolute" (sea level), "relativeToSeaFloor" (from the bottom of major bodies of water).

3.3.1.6 SCALE (FOR 3D)

scale = a number corresponding to how many meters one meter of our map represents. A value "50000" would mean 1:50K which means for every meter of our map it represents 50000 meters of real world distance.

3.3.1.7 PIXELWIDTH & PIXELHEIGHT (FOR 2D)

pixelWidth & pixelHeight = represents the width & height in pixels of the visible map area of a 2D map.

3.3.1.8 BOUNDING BOX (FOR 3D & 2D)

bbox = the viewable area of the map. Passed in the format of a string "lowerLeftX, lowerLeftY, upperRightX, upperRightY." example: "-50.4, 23.6, -42.2, 24.2"

3.3.1.9 MODIFIERS (FOR 3D & 2D)

Map<String, String> - use like:
```
modifiers.put(Modifiers.T_UNIQUE_DESIGNATION_1, "T");
```
Or:
```
modifiers.put(Modifiers.AM_DISTANCE, "1000,2000,3000");
```

3.3.1.10 ATTRIBUTES (FOR 3D & 2D)

Map<String, String> - use like:
```
modifiers.put(MilStdAttributes.LineWidth, "3");
```
Or:
```
modifiers.put(MilStdAttributes.LineColor, "#00FF00");
```

3.3.1.11 FORMAT (FOR 3D & 2D)

format = an enumeration: 3 for GeoSVG, 2 for GeoJSON (recommended), 1 for JSON **(deprecated)**, 0 for KML **(deprecated)**,

4 SYMBOLUTILITIES

armyc2.c5isr.renderer.utilities.SymbolUtilities is an object with various utility functions used to process symbolIDs. The two a user is most likely to use are below. Please look at the class for other functions that may be of use.

4.1 GETBASICSYMBOLID

Takes a Symbol ID like “10030500001107000000” and returns the 2-character symbol set + the 6-character entity code, which looks like "05110700".

4.2 HASMODIFIER

“hasModifier(symbolID, modifier)” will tell you if the modifier is valid for a specific symbolID.
symbolID – a 30-character symbol code.
modifier – Can be one of the constants from the Modifiers.
Returns true or false.

Use like: `SymbolUtilities.hasModifier(symbolID, Modifiers.B_ECHELON);`

5 MSLookup

The MSLookup object will provide information about all the 2525D symbols that can be drawn.

5.1 getMSInfo

`armyc2.c5isr.renderer.utilities.MSLookup.getMSInfo(String symbolID)`
symbolID being a 20- to 30-character symbol code

`armyc2.c5isr.renderer.utilities.MSLookup.getMSInfo(String basicID, int version)`
basicID being a String concatenating the 2-character Symbol Set value with the 6-character entity code of the symbol.
version being 11 for 2525Dch1 or 13 for 2525E (not yet supported).

This MSInfo object provides information such as:

- Version
- Symbol Set
- Name
- Path
- Geometry ("point", "line", "area")
- DrawRule (armyc2.c5isr.renderer.utilities.DrawRules)
- Min Point Count
- Max Point Count
- Modifiers (An ArrayList of the modifier constant ID that the symbol has)

6 Accessing data from the Symbol Picker

6.1 Android

Pressing the **PICK '<symbol name>'** button in the Symbol Picker creates an android.content.Intent object, then the SymbolPickerActivity is closed and the Intent is propagated back to the launching application via onActivityResult(). The Intent contains three kinds of data about the symbol to be rendered, which are mapped in key–value pairs:

Class containing the key constants: `armyc2.c5isr.renderer.symbolpicker.SymbolPickerActivity`

Key: `selectedSymbolIdKey`

The 30-digit Symbol ID Code as a String.

Key: `modifiersKey`

The symbol modifiers as a HashMap<String, String>. The modifier keys are constant Strings specified in `armyc2.c5isr.renderer.utilities.Modifiers`.

Key: `attributesKey`

The symbol attributes as a HashMap<String, String>. The attribute keys are constant Strings specified in `armyc2.c5isr.renderer.utilities.MilStdAttributes`.