Mil Symbology Renderer Web Service

Developer’S Guide

December 23th, 2013

Revision History

|  |  |
| --- | --- |
| Release | Date |
| Initial Document | 12/23/2013 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Contents

[1 Overview 1](#_Toc375141345)

[1.1 Features 1](#_Toc375141346)

[1.2 System Requirements 1](#_Toc375141347)

[1.2.1 Web Client Requirements 1](#_Toc375141348)

[1.2.2 Server Requirements 1](#_Toc375141349)

[2 Setting up the Mil Symbology Web application 1](#_Toc375141350)

[2.1 What's Included in the mil-symbology-renderer.war file 1](#_Toc375141351)

[2.2 Deploying the Mil Symbology Renderer Service on Tomcat 1](#_Toc375141352)

[3 Using the rest service for requesting icons 2](#_Toc375141353)

[3.1 Overview 2](#_Toc375141354)

[3.2 Single Coordinate Icon based symbology 2](#_Toc375141355)

[3.3 Additional Parameters for Images 2](#_Toc375141356)

[3.3.1 Size 2](#_Toc375141357)

[3.3.2 Alpha 2](#_Toc375141358)

[3.3.3 lineColor / fillColor 3](#_Toc375141359)

[3.3.4 keepUnitRatio 3](#_Toc375141360)

[3.3.5 meta 3](#_Toc375141361)

[3.3.6 Symbology Standard 4](#_Toc375141362)

[3.4 Additional Parameters for KML 4](#_Toc375141363)

[3.4.1 ID 4](#_Toc375141364)

[3.4.2 Name 4](#_Toc375141365)

[3.4.3 Longitude 4](#_Toc375141366)

[3.4.4 Latitude 5](#_Toc375141367)

[3.4.5 Altitude 5](#_Toc375141368)

[3.5 Getting Extent Information for Single Point Images 5](#_Toc375141369)

[4 Using the REST service for generating multipoint symbology KML 5](#_Toc375141370)

[4.1 Overview 5](#_Toc375141371)

[4.2 Multiple Coordinate KML based symbology 6](#_Toc375141372)

[4.3 Required Parameters for Multipoint Symbology 6](#_Toc375141373)

[4.3.1 ID (for 3D & 2D) 6](#_Toc375141374)

[4.3.2 name (for 3D & 2D) 6](#_Toc375141375)

[4.3.3 description (for 3D & 2D) 6](#_Toc375141376)

[4.3.4 control points (for 3D & 2D) 7](#_Toc375141377)

[4.3.5 altitude mode (for 3D) 7](#_Toc375141378)

[4.3.6 scale (for 3D) 7](#_Toc375141379)

[4.3.7 pixelwidth & pixelheight (for 2D) 7](#_Toc375141380)

[4.3.8 bounding box (for 3D & 2D) 7](#_Toc375141381)

[4.3.9 modifiers (for 3D & 2D) 7](#_Toc375141382)

[4.3.10 format (for 3D & 2D) 7](#_Toc375141383)

[4.3.11 symbology standard (for 3D & 2D) 7](#_Toc375141384)

[4.4 Additional Parameters for Multipoint Symbology 7](#_Toc375141385)

[4.4.1 lineColor / fillColor (for 3D & 2D) 7](#_Toc375141386)

# Overview

## Features

The Mil Symbology Renderer is both a developer’s toolkit as well as a ready to use deployable web application. The goal of this project is to provide a single distributable solution that can support as many use cases as possible for military symbology rendering. The current features available are:

* Use as a REST web service
* Single Point rendering as PNG
* Multi Point rendering as KML

## System Requirements

### Web Client Requirements

* Modern browser

### Server Requirements

* Java JRE or JDK 1.6 update 23 or newer
* Web server that supports Java Servlets and WAR file deployment

# Setting up the Mil Symbology Web application

## What's Included in the mil-symbology-renderer.war file

The Mil-symbology-service.war file contains:

1. Mil Symbology REST Service
2. Test pages multiPoints.html & singlePoints.html to demonstrate usage.

## Deploying the Mil Symbology Renderer Service on Tomcat

Place the mil-sym-service.war file into the webapps directory of your Tomcat web application server. When Tomcat runs, it will decompress the war file and a mil-sym-service directory with the source code will appear. At this point no further configuration is required and the REST web service can be used.

There is no known limitation to only use Tomcat and any Java servlet container should be able to run the WAR file, but this is currently only tested in Tomcat configurations

# Using the rest service for requesting icons

## Overview

The REST web service provided as part of the WAR file supports generation of PNG formatted image icons as well as KML representations of single coordinate MIL-STD-2525 symbols.

## Single Coordinate Icon based symbology

Once the WAR file has been placed on the web server, using the single point rendering REST service is as simple as calling a URL. There are two formats that are supported by the service: PNG images and KML.

You can add modifiers via URL parameters. When building the URL you must use the coded modifier names a described in the MIL-STD-2525 document (such as “T” or “T1”).

Sample URL for a KML request:

<http://[server]/mil-sym-service/renderer/kml/SFGPUCDM---M--->

Sample URL for an image request:

[http://[server]/mil-sym-service /renderer/image/SFGPUCDM---M---](http://[server]/mil-sym-service%20/renderer/image/SFGPUCDM---M---)

Sample URL with some modifiers as parameters:

[http://[server]/mil-sym-service /renderer/image/SFGPUCDM---M---?V=35&T=TFOO&H=HFOO](http://[server]/mil-sym-service%20/renderer/image/SFGPUCDM---M---?V=35&T=TFOO&H=HFOO)

## Additional Parameters for Images

You can also adjust properties such as:

### Size

“**Size** = 35”, (default size 35) size of the core symbol (not modifiers) will fit within a 35x35 pixel space.

### Alpha

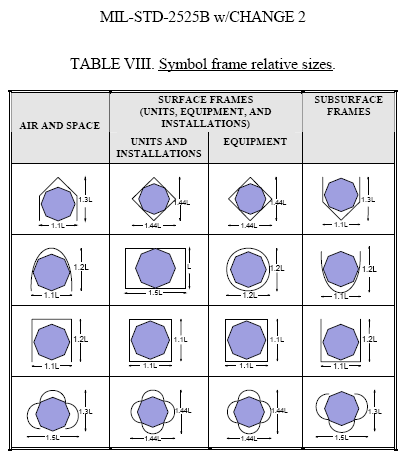
**alpha**=0-255, 0 is completely transparent, 255 is totally visible.

### lineColor / fillColor

**lineColor** and **fillColor**, if you want to override the default affiliation colors, do something like: “linecolor=0xFF0000”

### keepUnitRatio

**keepUnitRatio**=true, (defaults true) If true, the symbols will have proper proportions in relation to each other (see chart below). 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.



### meta

**meta**=true, (default is false) adds 4-6ms to image creation but embeds position info into the PNG.

If you can pull this information out of the header, it will look like this:  
  
<javax\_imageio\_png\_1.0>  
 <tEXt>  
 <tEXtEntry keyword="centerPoint" value="x=17,y=60"/>  
 <tEXtEntry keyword="bounds" value="x=1.0,y=1.0,width=32.0,height=60.0"/>  
 <tEXtEntry keyword="imageExtent" value="width=32,height=60"/>  
 </tEXt>  
</javax\_imageio\_png\_1.0>

Typically you would center an image on the related map point. However, not all graphics center on the center of the image. Check Point would center on the bottom middle of the image. This is where the information provided in the header of the PNG becomes useful. You know what point of the image should be centered on the location. The “bounds” value shows the bounding rectangle of the symbol (minus any modifiers) in the image which is potentially useful for hit testing. Also, modifiers can change the center of the image.

### Symbology Standard

**symStd**= “2525B” or “2525C”. This parameter (symbology standard) determines which rendering standard to use. “2525B” represents 2525Bch2 with USAS 13-14. “2525C” represents 2525C with USAS 13-14.

## Additional Parameters for KML

When requesting KML instead of a PNG icon some additional parameters are required to create valid KML. These have no effect the how the symbol looks. An example of a simple URL to request a KML representation is below:

[http://[server]/mil-sym-service /renderer/kml/SFGPUCDM---M---?id=1234&name=my](http://[server]/mil-sym-service%20/renderer/kml/SFGPUCDM---M---?id=1234&name=my) KML&lon=44.5&lat=33.5&alt=1000

Additional parameters for the MIL-STD-2525 symbol modifiers can be added to further adjust the visual representation of the symbol.

### ID

**Id** = unique ID for the symbol. This will populate the id parameter of the placemark node in the KML.

### Name

**Name** = a name for the symbol. This will populate the name node of the KML.

### Longitude

**Lon** = longitude in degrees decimal format (33.477543). This will be used in the geometry section of the placemark of the KML.

### Latitude

**Lat** = latitude in degrees decimal format (53.41943). This will be used in the geometry section of the placemark of the KML.

### Altitude

**Alt** = altitude as an integer representing meters (5000). This will be used in the geometry section of the placemark of the KML.

## Getting Extent Information for Single Point Images

It is possible to query the web service to get extents of the single point image via a POST request. You can look at the function batchInfoTest() in singlePoints.html inside the war for more detail but below is a code snippet.

var http = new XMLHttpRequest();

var url = "renderer/spbi";

var params = "iconURLs=" + encodeURIComponent("{\"iconURLs\":[\"SFGP------\*\*\*\*\*?size=35&T=Hello\",\"SHGPE-----\*\*\*\*\*?size=50\"]}");

http.open("POST",url,true);

http.setRequestHeader("Content-type","application/x-www-form-urlencoded; charset=UTF-8,");

http.setRequestHeader("Content-length",params.length);

http.onreadystatechange = function(){/\* your state change handling code where you receive your extent info as a JSON formatted string \*/};

http.send(params);

The returned JSON string will look something like this:  
{"singlepoints":[{"x":52.0,"y":13.0,"boundsx":35.0,"boundsy":0.0,"boundswidth":34.0,"boundsheight":26.0,"iconwidth":69.0,"iconheight":33.0},{"x":24.0,"y":24.0,"boundsx":0.0,"boundsy":0.0,"boundswidth":48.0,"boundsheight":48.0,"iconwidth":48.0,"iconheight":48.0}]}

Alternatively, you could use the JavaScript renderer to get the dimensions. There may be minimal differences in measurements based on what browser you use. Sometimes there’s a 1 pixel difference between IE and FireFox.

# Using the REST service for generating multipoint symbology KML

## Overview

The REST web service provided as part of the WAR file supports generation of KML to represent multipoint Mil-Std 2525 symbology for use on 3D maps.

## Multiple Coordinate KML based symbology

Once the WAR file has been placed on the web server, using the multipoint rendering REST service is as simple as making an http POST. There are two formats that are supported by the service: KML and JSON for use on 2D or 3D maps. The KML output can be fed directly to a 3D map that supports the format, like Google Earth.

In the POST URL you specify if you’d like a rendering based on a 2D or 3D map followed by the symbol ID.

Sample URL for a KML POST request:

[http://[server]/mil-sym-service /renderer/mp3d/GFFPAXS-------X](http://[server]/mil-sym-service%20/renderer/mp3d/GFFPAXS-------X) (3d map)  
[http://[server]/mil-sym-service /renderer/mp2d/GFFPAXS-------X](http://[server]/mil-sym-service%20/renderer/mp2d/GFFPAXS-------X) (2d map)

The JavaScript code snippet below shows how parameters can be sent. For more detail, look at “multipoints.html” in the WAR. When building the parameter string, you must use the coded modifier names a described in the MIL-STD-2525 document (such as “T” or “T1”). Modifiers “AM”, “AN” & “X” take values as an array. Other Mil-Std modifier values can be set as strings.

var url = “renderer/mp3d/GFFPAXS-------X”;

var params = “ID=ID&NAME=Name&DESCRIPTION=Description &CONTROLPOINTS=66.26700036208742,30.62755038706961&ALTITUDEMODE=clampToGround&SCALE=50000&BBOX=66.25,30.627,66.27,30.63&MODIFIERS={\"modifiers\":{\"AM\":[3000,10000],\"AN\":[315,45],\"X\":[0]}}&FORMAT=0&SYMSTD=0”;

http.open("POST",url,true);

http.setRequestHeader("Content-type","application/x-www-form-urlencoded; charset=UTF-8,");

http.setRequestHeader("Content-length",params.length);

http.onreadystatechange = function(){/\* your state change handling code where you receive your rendered KML \*/};

http.send(params);

## Required Parameters for Multipoint Symbology

### 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.

### name (for 3D & 2D)

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

### description (for 3D & 2D)

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

### 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[,z1] [xn,yn[,zn]]...".

### 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).

### 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.

### pixelwidth & pixelheight (for 2D)

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

### 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"

### modifiers (for 3D & 2D)

**modifiers** = a JSON string representing all the possible symbol modifiers represented in the MIL-STD-2525C. Format of the string will be {"modifiers": {"attributeName":"value"[,"attributeNameN":"valueN"]...}}. The quotes are literal in the above notation. Example: {"modifiers": {"C":"4","Z":"300","AN":[100,200]}}

### format (for 3D & 2D)

**format** = an enumeration: 0 for KML, 1 for JSON.

### symbology standard (for 3D & 2D)

**symStd** = a Mil-Std symbology enumeration: 0 for 2525Bch2, 1 for 2525C.

## Additional Parameters for Multipoint Symbology

### lineColor / fillColor (for 3D & 2D)

**lineColor** and **fillColor** = are used to override the default affiliation colors. Example: “lineColor=0xFF0000”

# Running the Web Service as a local desktop service

The local service mirrors the behavior of the web service. But instead of calling "http://[server]/mil-sym-service/renderer/kml/SFGPUCDM---M---", you'd call "http://127.0.0.1:6789/mil-sym-service/renderer/image/SFGPUCDM---M---".

## Starting the local service

The local service can be started with the following command:  
"java -cp [location of jar files] sec.web.renderer.portable.PortableWrapper [args]".

So assuming your’re starting in the folder with all the jars and java.exe is on the path, you could use:  
“java –cp \* sec.web.renderer.portable.PortableWrapper -spport:6789 -mpport:6790”

It can also be started using the included JNLP file.

## Arguments

Modifier values can be set as strings.

-? print the help message.

-spon starts the single point service. (default true)

-spport desired port for the single point service. (default 6789)

-spbacklog desired backlog for the single point service. (default0, lets system decide)

-spon starts the multi point service. (default true)

-mpport desired port for the multi point service. (default 6790)

-mpbacklog desired backlog for the single point service. (default0, lets system decide)

Usage like: java -cp [location of jar files] sec.web.renderer.portable.PortableWrapper -spport:#### -spbacklog:### -mpport:#### -mpbacklog:###

Or, if launching using the JNLP file, these values can be configured from within that file.

# Using the Rendering Librarues in an application

You can reference "mil-sym-renderer-jar-with-dependencies.jar" to get all the classes in one jar file or you can reference them individually:

JavaRendererUtils-#.#.#.jar  
JavaLineArray-#.#.#.jar  
JavaRendererServer-#.#.#.jar  
JavaRenderer-#.#.#.jar  
RendererPluginInterface-#.#.#.jar  
mil-sym-renderer.jar

A code sample is available here:  
<https://github.com/missioncommand/mil-sym-java/tree/master/samples>

## Configuration

[ArmyC2.C2SD.Utilties.RendererSettings](https://github.com/missioncommand/mil-sym-java/blob/master/core/JavaRendererUtils/src/main/java/ArmyC2/C2SD/Utilities/RendererSettings.java) is home to most of your rendering options. You can set them as follows:

RendererSettings.getInstance().setSymbologyStandard(RendererSettings.Symbology\_2525C);  
RendererSettings.getInstance().setTextOutlineWidth(1);

## Rendering Single Point Images

Below is a usage snippet. Refer to [DrawUnit()](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java#L211) and [DrawTG()](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java#L265) in the [sample code](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java) for more detail.

SECRenderer sr = sec.web.renderer.SECRenderer.getInstance();

String symbolID = "SFGP-----------";

Map<String,String> params = new HashMap<String, String>();

params.put(ModifiersUnits.T\_UNIQUE\_DESIGNATION\_1, "T modifier");

//If Tactical Graphic

//params.put(ModifiersTG.T\_UNIQUE\_DESIGNATION\_1, "T modifier");

params.put(MilStdAttributes.PixelSize,"50");

PNGInfo pi = sr.getMilStdSymbolImage(symolID, params);

Once you have a [PNGInfo](https://github.com/missioncommand/mil-sym-java/blob/master/renderer/mil-sym-renderer/src/main/java/sec/web/renderer/utilities/PNGInfo.java) object, you can get the BufferedImage object, save the image to a file, get a squared version of the image (good for icons or tree nodes) and get the point of the image that should be the anchor when placing it on a map.

Other objects of interest are [MilStdAttributes](https://github.com/missioncommand/mil-sym-java/blob/master/core/JavaRendererUtils/src/main/java/ArmyC2/C2SD/Utilities/MilStdAttributes.java), [ModifiersUnits](https://github.com/missioncommand/mil-sym-java/blob/master/core/JavaRendererUtils/src/main/java/ArmyC2/C2SD/Utilities/ModifiersTG.java), [ModifiersTG](https://github.com/missioncommand/mil-sym-java/blob/master/core/JavaRendererUtils/src/main/java/ArmyC2/C2SD/Utilities/ModifiersTG.java), and [SECRenderer](https://github.com/missioncommand/mil-sym-java/blob/master/renderer/mil-sym-renderer/src/main/java/sec/web/renderer/SECRenderer.java).

## Rendering Multi Point Images

Refer to [generateAmbushKML()](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java#L304) and [generateSectorRangeFanKML()](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java#L377) in the [sample code](https://github.com/missioncommand/mil-sym-java/blob/master/samples/rendering-sample-1/src/main/java/sec/web/renderer/samples/Gui.java) for more detail.

## The Symbol Definition Table

The [SymbolDefTable](https://github.com/missioncommand/mil-sym-js/blob/master/web/js/src/armyc2/c2sd/renderer/utilities/SymbolDefTable.js) object will provide information about the tactical & METOC graphics that can be drawn. (Appendix B in the MilStd 2525)

### hasSymbolDef

armyc2.c2sd.renderer.utilities.SymbolDefTable.getInsatnce().getSymbolDef(symbolID, symStd) This function takes the following parameters: symbolID – a basic 15 character symbol ID code stripped of affiliation, status & modifiers.  
symStd – 0 or 1 depending if you want a definition based on 2525B or 2525C. Returns true if a symbol definition exists for the passed basic symbolID and symbology standard.

### getSymbolDef

armyc2.c2sd.renderer.utilities.SymbolDefTable.getInsatnce().getSymbolDef(symbolID, symStd)

#### Parameters

This function takes the following parameters: symbolID – a basic 15 character symbol ID code stripped of affiliation, status & modifiers. symStd – 0 or 1 depending if you want a definition based on 2525B or 2525C.

#### Return Object

This function will return a SymbolDef object with the following properties:

##### symbolID

symbolID – a basic 15 character symbol ID code stripped of affiliation, status & modifiers.

##### minPoints

minPoints – the minimum # of points the symbol can have.

##### maxPoints

maxPoints – the max # of points the symbol can have. Assume and value over 100 means infinite.

##### modifiers

modifiers – a string like “T.T1.W.” which shows which modifiers the symbol can have.

##### drawCategory

drawCategory – an integer representing the type of symbol that is being drawn. These constants can be found in [SymbolDef.java](https://github.com/missioncommand/mil-sym-java/blob/master/core/JavaRendererUtils/src/main/java/ArmyC2/C2SD/Utilities/SymbolDef.java#L34). Based on these constants, you’ll know what symbols have required modifiers. For instance, DRAW\_CATEGORY\_TWO\_POINT\_RECT\_PARAMETERED\_AUTOSHAPE: 20, is a symbol that requires one AM value. These requirements are described in SymbolDefTable.js as well.