Mil Symbology JavaScript Renderer

Developer’S Guide

February 24th, 2013

Revision History

|  |  |
| --- | --- |
| Release | Date |
| Initial Document | 12/18/2012 |
| Updated | 02/24/2014 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Contents

[1 Overview 1](#_Toc380999183)

[1.1 Features 1](#_Toc380999184)

[1.2 System Requirements 1](#_Toc380999185)

[1.2.1 Single Point Requirements 1](#_Toc380999186)

[1.2.2 Multi Point Requirements 1](#_Toc380999187)

[2 Setting up the JavaScript Renderer for Use 1](#_Toc380999188)

[2.1 Compile the mil-sym-js project 1](#_Toc380999189)

[2.2 Loading the JavaScript Renderer in your page 2](#_Toc380999190)

[3 Rendering 2](#_Toc380999191)

[3.1 Configuring the Renderer for your Needs 2](#_Toc380999192)

[3.2 Singlepoint Icon Symbology 2](#_Toc380999193)

[3.2.1 Setting Modifiers 2](#_Toc380999194)

[3.2.2 Setting Attributes 3](#_Toc380999195)

[3.2.3 Getting your Image (ImageInfo) 4](#_Toc380999196)

[3.2.4 Rendering Your Image via HTML5 Canvas 5](#_Toc380999197)

[3.2.5 Rendering Your Image via Data URL 5](#_Toc380999198)

[3.2.6 Rendering Your Image via jQuery plugin 6](#_Toc380999199)

[3.3 Multipoint Symbology 6](#_Toc380999200)

[3.3.1 Required Parameters for Multipoint Symbology 7](#_Toc380999201)

[4 Appendix A Single Point Modifiers 8](#_Toc380999202)

# Overview

## Features

The JavaScript versions of the Mil-Std Symbology Renderer supports rendering of single & multi point symbology. Single points as an html5 canvas object or a data url. Multi points as kml.

## System Requirements

### Single Point Requirements

IE 9+ or FireFox 22+ or Chrome 32+ or Opera 21+. (Chrome & Opera haven’t been fully tested but seem to work well)

### Multi Point Requirements

Multi Point rendering should work with most current browsers.

# Setting up the JavaScript Renderer for Use

## Compile the mil-sym-js project

This project builds with Ant resulting in two JavaScript files. One has all of the JS code combined into one file. The other is a minified version of the first file. (i.e. sm-bc.js & sm-bc.min.js)

Open a command prompt to the root folder of the repository on your local machine and enter:

ant clean

ant concat sm-bc minify samples

"sm-bc" represents what you rendering capabilities you want.

* s: singlepoint (jquery plugin for singlepoint rendering only available in singlepoint only builds)
* m: multipoints
* sm: both
* b: 2525B support
* c: 2525C support
* bc: both
* allFlavors: every variation will be generated

Running "ant concat sm-bc minify samples" would result in the following files being placed in the "dist" folder:

* fonts (contains fonts needed for singlepoint rendering)
* renderer.css (to load the font files)
* jquery-[version].min.js
* multiPointTester1.html (renders kml for a couple multipoint symbols)
* multiPointTester2.html (renders kml for a couple multipoint symbols in a loop to test performance)
* singlePointTester.html (renders a couple of singlepoint symbols to the page)
* single-point-plugin.html (show sample usage with jQuery)
* sm-bc.js (concatenated renderer code)
* sm-bc.min.js (concatenated & minified renderer code)

## Loading the JavaScript Renderer in your page

The top of your html page should look similar to the snippet below:

<!DOCTYPE html>

<html>

<head>

<!-- Meta line required for IE-->

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<title>single point tester</title>

<link rel="stylesheet" href="renderer.css" type="text/css" charset="utf-8" />

<script src="sm-bc.min.js" type="text/javascript" ></script>

This loads the required font files via the css file and pulls in the concatenated JavaScript renderer file.

# Rendering

## Configuring the Renderer for your Needs

The “RendererSettings” object will let you set some default rendering values. It is accessible at “armyc2.c2sd.renderer.utilities.RendererSettings”.

RendererSettings.setSymbologyStandard(RendererSettings.Symbology\_2525C);

RendererSettings.setTextOutlineWidth(1);

## Singlepoint Icon Symbology

You would typically render a single point via the code snippet below:

var ii = armyc2.c2sd.renderer.MilStdIconRenderer.Render("SUGDUSAT----\*\*\*",modifiers);

### Setting Modifiers

“modifiers” is an object which can contain Mil-Std modifiers and rendering attributes. The modifiers can be set like this:

var mu = armyc2.c2sd.renderer.utilities.ModifiersUnits

var modifiers = {};

modifiers[mu.C\_QUANTITY]=10;

modifiers[mu.H\_ADDITIONAL\_INFO\_1] = "H";

modifiers[mu.H1\_ADDITIONAL\_INFO\_2] = "H1";

//or like this for Single Point Tactical Graphics:

modifiers[armyc2.c2sd.renderer.utilities.ModifiersTG.H2\_ADDITIONAL\_INFO\_3] = "H2";

### Setting Attributes

Attributes will override any defaults set in RendererSettings.

var msa = armyc2.c2sd.renderer.utilities.MilStdAttributes;

modifiers[msa.PixelSize]=60;

modifiers[msa.KeepUnitRatio]=true;

modifiers[msa.SymbologyStandard] = RendererSettings.Symbology\_2525Bch2\_USAS\_13\_14;

#### Size

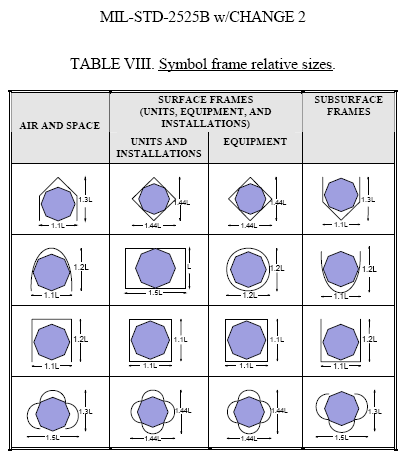
“**PixelSize** = 35”, (default size 35) size of the core symbol (not modifiers) will fit within a 35x35 pixel space.

#### 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.



#### 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.

### Getting your Image (ImageInfo)

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.

var ii = armyc2.c2sd.renderer.MilStdIconRenderer.Render("SUGDUSAT----\*\*\*",modifiers);

What is returned is the “ImageInfo” object (armyc2.c2sd.renderer.utilities.ImageInfo). It has the following functions available.

#### getImage()

“getImage()” returns an HTML5 canvas object which you can use to draw to another canvas.

#### toDataUrl()

“toDataUrl()” returns a base64 string that represents the image.

#### getCenterPoint()

“getCenterPoint()” returns a point object that represents where the image should be centered if rendered on a coordinate based map. {x:Number,y:Number}

#### getSymbolBounds()

“getSymbolBounds()” returns a rectangle object that represents to area in the image that the core symbol (not including any modifiers) exists. {x:Number,y:Number,width:Number,height:Number}

#### getImageBounds()

“getImageBounds()” returns a rectangle object that represents the size of the entire image. {x:Number,y:Number,width:Number,height:Number}

### Rendering Your Image via HTML5 Canvas

In your html you’ll need a canvas object.

<canvas id="preview" width="650" height="200"></canvas>

In your JavaScript, you can use the following code to render.

var preview = document.getElementById("preview");

var ctx = preview.getContext('2d');

ctx.drawImage(ii.getImage(), 0, 0);

### Rendering Your Image via Data URL

Have an image tag somewhere in your html:

<img id="urlexample" src="" alt="test" height="35" width="35">;

Modify the image tag with this JavaScript code:

//render via Data Url

modifiers = new Object();

modifiers[msa.PixelSize]=60;

ii = armyc2.c2sd.renderer.MilStdIconRenderer.Render("SHAPWMSA-------",modifiers);

var width = ii.getImageBounds().getWidth();

var height = ii.getImageBounds().getHeight();

var img = document.getElementById("urlexample");

img.src = ii.toDataUrl();

img.width = width;

img.height = height;

### Rendering Your Image via jQuery plugin

Note that this only works with the single point only compilations of the JavaScript renderer.

<html>

<head>

<title>Renderer jQuery Plugin Example</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<link rel="stylesheet" href="renderer.css" type="text/css" charset="utf-8" >

<script src="jquery-1.10.2.min.js" type="text/javascript"></script>

<script src="s-b.min.js" type="text/javascript"></script>

<script type="text/javascript">

$(window).bind("load", function() {

// Invoke render2525() on jQuery objects that match canvas tags

// when symbols on a canvas need to be drawn or re-drawn

$("canvas").render2525();

});

</script>

</head>

<body>

<!--

To display a symbol, canvas elements must contain a valid

15-character symbol code in the data-symbol-code attribute.

The optional data-pixel-size attribute controls the size of

the symbol.

-->

<canvas data-symbol-code="SHAPCF---------" data-pixel-size="64"></canvas>

</body>

</html>

## Multipoint Symbology

KML rendering example:

var formatJSON = 1;

var formatKML = 0;

var rendererMP = sec.web.renderer.SECWebRenderer;

var pixelWidth = 800;

var pixelHeight = 600;

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

var symbolCode3 = "GFFPAXS---\*\*\*\*X"; //sector range fan

var kml3=null;

var json3=null;

var controlPoints3 = "66.26700036208742,30.62755038706961";// point format “x,y x,y x,y…”

/\*bbox The viewable area of the map. Passed in the format of a string "lowerLeftX,lowerLeftY,upperRightX,upperRightY." Not required but can speed up rendering in some cases.//\*/

var bbox3 = "66.25,30.627,66.27,30.63";//whole symbol will be calculated

var scale3 = 50000.0;

//distance (AM), azimuth (AN), and altitudeDepth (X) can all have multiple values so they are enclosed in brackets.

//you can set with the string constants from “ModifiersTG”.

var mtg = armyc2.c2sd.renderer.utilities.ModifiersTG;

modifiers = {};

modifiers[mtg.AM\_DISTANCE] = [300,1000];//AM, AN, & X are the only modifiers that are passed as arrays.

modifiers[mtg.AN\_AZIMUTH] = [315,45]; //The rest of the modifiers are String or Number values.

modifiers[mtg.X\_ALTITUDE\_DEPTH] = [0];

var msa = armyc2.c2sd.renderer.utilities.MilStdAttributes;

modifiers[msa.LineWidth]=8;

format = formatKML;

//RenderSymbol for Google Earth. KML format recommended.

kml3 = rendererMP.RenderSymbol("ID","Name","Description", symbolCode3, controlPoints3, "clampToGround",scale3, bbox3, modifiers,format);

format = formatJSON;

//RenderSymbol2D for 2D maps, JSON Recommended for simpler output.

json3 = rendererMP.RenderSymbol2D("ID","Name","Description", symbolCode3, controlPoints3, pixelWidth, pixelHeight, bbox3, modifiers,format);

### 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.

# SymbolUtilities

armyc2.c2sd.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.

## getBasicSymbolID

“getBasicSymbolID(symbolID)” will be used often with the other objects in the renderer. It will take a symbolID like “SFGPUCI----K\*\*\*” and turn it into “S\*G\*UCI---\*\*\*\*\*”. Object like SymbolDefTable & UnitDefTable use the basic symbol ID for lookups.

## hasModifier

“hasModifier(symbolID, modifier)” will tell you if the modifier is valid for a specific symbolID.

symbolID – a 15 character symbol code.   
modifier – Can be one of the constants from the ModifiersUnits or ModifiersTG object.  
Returns true or false.

Use like:

Var SymbolUtilities = armyc2.c2sd.renderer.utilities.SymbolUtilities;

var result = SymbolUtilities.canUnitHaveModifier(symbolID, ModifiersUnits.B\_ECHELON);

## isMultiPoint

“isMultiPoint(symbolID, symStd)” will tell you if the modifier is valid for a specific symbolID.

symbolID – a 15 character symbol code.   
symStd – a Mil-Std symbology enumeration: 0 for 2525Bch2, 1 for 2525C. Will use renderer default if not provided.

Returns true or false.

# The Symbol Definition Table

The SymbolDefTable object will provide information about the tactical graphics that can be drawn.

## HasSymbolDef

armyc2.c2sd.renderer.utilities.SymbolDefTable.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.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 an 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 any 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 the SymbolDefTable.js. 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.