SYMBOLOGY FRAMEWORK

V1.0

CM ID: 21360810-097

07 OCTOBER 2021

Dignitas Technologies, LLC
3626 Quadrangle Boulevard, Suite 100
Orlando, FL 3281

TABLE OF CONTENTS

1. OVERVIEW

The Symbology Framework is intended to provide a basis for creating and maintaining a new symbology library allowing clients to retrieve symbol assets and maintain a consistent look and feel while displaying objects to a user. This framework will define a schema for configuring a symbol library, allowing symbol assets to be quickly added, modified, or removed. It is intended to be used with SVG assets as a Node.js package and should be easily embeddable into web development projects or used as server with a RESTful interface. The current baseline contains an implementation of the NAPSG icon set from their website.

2. USE CASES

This section covers the standard use cases intended for the Symbology Framework and the procedures to perform them.

2.1 CREATE OR UPDATE A SYMBOL LIBRARY

1. Extract or checkout the symbology framework source. The existing baseline already includes a set of frames, sample decorators, and icons. The available icons were generated from the set of icons available on the NAPSG website.
2. Establish the symbol hierarchy. All assets reside under the `dhs-symbol/assets` directory. Icons can be hierarchical and the directory structure within the `icons` directory will establish that. Each directory within the `icons` directory structure should have either icon files or category directories at each level, not both. Create, update, or remove any category directories to establish the desired symbol hierarchy.
3. Create new assets in text-based SVG format and add them to the appropriate directories. Documentation on the SVG standard can be found at https://developer.mozilla.org/en-US/docs/Web/SVG. As well, assets should stick to using a viewbox of (0, 0, 100, 100). To avoid problems with some browsers, assets should use path elements when possible, avoid using the class attribute, and apply def/style attributes directory to paths versus using group. When naming asset files, avoid using digits as this will cause a dash to be inserted into the internal name at compile time.
    a. Frames represent an icon's surrounding border such as a circle or a square. The filenames for frames should be prepended with "frame-" and placed in the `dhs-symbol/assets/frames` directory.
    b. Icons represent the primary or center icon to be shown. The filenames for icons should be prepended with "icon-" and placed in the dhs-symbol/assets/icons directory. The icon's symbol identification code (SIDC) will be an appended string of the root category id followed by each ancestor subcategory id followed by the icon id. Ex. For a category with an id of "R", a subcategory with an id of "E", a further subcategory with an id of "B" and an icon id of "AR", the asset name would be "icon-REBAR.svg".
    c. Decorators represent icons than can be added as smaller surrounding icons to a primary/center icon to represent additional capabilities the corresponding object may

have or carry such as a cellphone. The filenames for decorators should be prepended with “dec-“ and placed in the dhs-symbol/assets/decorators directory.

4. Update, move, and remove any assets as needed following the same guidelines as step 3.
5. Edit the Symbol Map. The dhs-symbol/assets/symbol-map.json file defines the symbol library. The allowable schema for the symbol map can be found below in the Interfaces section. Each category, subcategory, and icon should be assigned a unique id within its hierarchical level. Note: the “dec” and “frame-“ prefixes are not used within the symbol map file itself when defining decName and frameName values.

2.2 INSTALL NODE.JS

1. Download the Node.js Windows Binary from https://nodejs.org/en/download/
2. Extract node-v14.17.6-win-x64.zip (or whichever version you choose) to C:\Node.js\ node-v14.17.6-win-x64 (or wherever is desired). May need to create Symbols directory first.
3. Add Node.js to PATH. Make sure to set to path used in Step 2. Note: this only sets from the current command line. Update system environment to make permanent.
   - set PATH=C:\Node.js\node-v14.17.6-win-x64

2.3 BUILD THE SYMBOL LIBRARY

1. Install Node.js using the previous steps.
2. Extract or checkout the symbology framework source and build using the previous steps in Section 2.2.
3. Build the updated library by running build.bat or use the following commands:
   - cd dhs-symbol
   - npm install
   - npm run build

2.4 RUN FROM NODE.JS SERVER

1. Install Node.js using the previous steps.
2. Extract or checkout the symbology framework source and build using the previous steps.
3. Build the Symbol Library using the previous steps in Section 2.3.
4. Build and Run the Server by running run.bat or use the following commands:
   - cd dhs-server
   - npm install
   - node build\index.js

2.5 EMBED LIBRARY INTO WEBPAGE

1. Install Node.js using the previous steps. Note: for this use case, Node.js is only used to compile as the server does not need to be run.
2. Extract or checkout the symbology framework source and build using the previous steps.
3. Build the Symbol Library using the previous steps in Section 2.4.
4. Place the output dhssymbol.js file in the directory with your html that will import it.
5. The following is example HTML that demonstrates using the symbol library without running the server:

```
<!DOCTYPE html>
<html>
<head>
	<title>Symbol Generator</title>
	<meta charset="UTF-8" />
</head>
<script src="dhssymbol.js"></script>
<script>
	function generateSymbol() {
		var style = { outline: true, outlineColor: 'yellow' };
		var badge = { badgeCount: 5, badgeFillColor: 'grey' };
		var decorators = [{ name: 'cellphone', frameColor: 'red' }];
		var symbol = new dhs.Symbol('BBA', style, badge, decorators).asSVG();
		console.log(symbol);
		var imageElement = document.getElementById("imageId");
		imageElement.src = "" + btoa(symbol);
	}
</script>
<body onload=generateSymbol()>
	<img id="imageId">
</body>
</html>
```

3. INTERFACES

3.1 SYMBOL MAP

The file *symbol-map.json* can be found in the assets directory of the symbol library. In the current baseline, this file has been created to represent the full set of NAPSG icons found on their website. This file contains the JSON definition of a symbology library. It defines the available frames, icons, categories, and colors. The symbol identification code (SIDC) is used to name the icon SVG files and is the mapping key for finding an icon. The following defines the schema for the JSON.

key	req/opt	type	description
symbologyName	required	string	Name of the symbology map/definition
version	required	string	Version of the symbology map
categories	required	Category[]	Set of categories
catIdLength	required	number	Number of characters used to represent the symbol id code (SIDC)
frames	optional	Frame[]	Set of defined frames
colors	optional	Color[]	Set of defined colors
decorators	optional	Decorator[]	Set of defined decorators

3.1.1 COLOR

A color object that allows for assigning a unique name to an RGB color. This makes it easier for symbol library developers to create a set of colors to be used by the library.

key	req/opt	type	description
colorName	required	string	Unique name of color
colorString	required	string	A standard CSS hex or RGB color string. ex. "red" or "#00FF00" or "rgb(0,0,255)"

3.1.2 FRAME

A frame object for defining a shape that borders an icon. Each frame object should correspond to an SVG asset found in the symbol assets folder. When naming assets, the corresponding SVG file should pre-pend "frame-" to the frameName from this definition. This definition also allows for custom frames to be defined that do not have a corresponding SVG asset, by adding a path string using the framePath key. The path string should adhere to the SVG standard for path.

key	req/opt	type	description
frameName	required	string	Unique frame name. This should match a frame asset or set framePath to define a dynamic frame. Ex. The frameName "square" would correspond to a file in assets with the name "frame-square.svg".
framePath	optional	string	SVG path string for dynamically creating a custom frame
frameColor	optional	string	Color of frame. Locally defined colors are checked first. If not present, then assume CSS Color syntax. ex. "red" or "#00FF00" or "rgb(0,0,255)"

3.1.3 CATEGORY

A nestable object that defines the hierarchy of the symbology set. Each category has a representative code and optionally a set of subcategories or icons associated with it. If either the subcategories or icons are defined for the category, then the character length of the code being used to uniquely identify the items in those sets must also be defined.

key	req/opt	type	description
catName	required	string	Displayable name
catDesc	required	string	Displayable description
catId	required	string	Set of unique characters to use in symbol id code (sidc). The number of valid characters is defined by the parent category or the symbol map itself.
frameName	optional	string	Frame assigned to this category. This corresponds to a frame defined in the symbology map and defaults to outline of viewbox if not set or or overridden by subcategory.
icons	optional	Icon[]	Set of this category's icons instances

iconIdLength	optional	number	Number of characters used to represent the symbol id code (sidc) of the icons in this category. Needs to be present if icons are used.
subcategories	optional	Category[]	Set of this category's sub-categories
subcatIdLength	optional	number	Number of characters used to represent the symbol id code (sidc) of the subcategories in this category. Needs to be present if subcategories are used.

3.1.4 ICON

An inner icon image object. Each icon object should correspond to an SVG asset found in the symbol assets folder. When naming assets, the corresponding SVG file should pre-pend "icon-" to the icon's SIDC from this definition and hierarchy ids. The icon's SIDC will be an appended string of the root category Id followed by each ancestor subcategory Id followed by the icon Id. Ex. For a category with an Id of "R", a subcategory with an Id of "E", a further subcategory with an Id of "B" and an icon Id of "AR", the asset name would be "icon-REBAR.svg".

key	req/opt	type	description
iconName	required	string	Unique icon name. A corresponding icon asset should exist. However, unlike frame, the asset name should include the full SIDC and be pre-pended with "icon-".
iconDesc	required	string	Icon description
iconId	required	string	Set of unique characters to use in symbol id code (sidc). The number of valid characters is defined by the parent category or the symbol map itself.
frameName	optional	string	Frame assigned to this icon. This corresponds to a frame defined in the symbology map and defaults to the category/subcategory frame.

3.1.5 DECORATOR

A decorator image object

key	req/opt	type	description
decName	required	string	Unique decorator name. This should match a decorator asset or set decPath to define a dynamic decorator. Ex. The decName "cellphone" would correspond to a file in assets with the name "dec-cellphone.svg".
decPath	optional	string	SVG path string for dynamically creating a custom decorator
decColor	optional	string	Color of decorator. Locally defined colors are checked first. If not present, then assume CSS Color syntax. ex. "red" or "#00FF00" or "rgb(0,0,255)"

3.2 URL PARAMETERS

3.2.1 STYLE

Style arguments are passed as URL parameters and allow the user to adjust visual aspects of the desired icon such as colors and sizing. The following defines the available style arguments.

parameter	type	default	description
decoratorLocation	(BOTTOM, TOP, BOTH)	BOTTOM	Location relative to symbol where decorators will appear, default is bottom
decoratorMaxCount	number	3	Maximum allowed decorators to display (0-6), default is 3
fill	boolean	true	Should the frame get filled, default is false
fillColor	string	white	Frame fill color if enabled, default is white
fillOpacity	number	1	Frame fill opacity (0-1) if enabled, default is 1 fully opaque
frame	boolean	true	Should the frame be displayed, default is true

frameName	string	empty string	Overrides default frame, this name should correspond to a frame definition in the symbology map
frameColor	string	black	Stroke color of frame, default is black
frameWidth	number	4	Frame stroke width, default is 4
icon	boolean	true	Show inner icon, default is true
iconScale	number	1	Scale the inner icon, default is 1
iconColor	string	empty string	Override icon stroke color, optional
iconStrokeWidth	number	1	Icon stroke width, default is 1
monoColor	string	false	Draw the icon as a single color, overrides all other colors except outline, optional
name	string	empty string	Name text to be shown to the right of the symbol, optional
nameBgColor	string	white	The text background color of the name, default is white
nameColor	string	black	The text color of the name, default is black
nameFontSize	number	28	The font size of the name text, default is 28
outline	boolean	false	Show an outline, default is false
outlineColor	string	white	Color of outline if enabled, default is white
outlineWidth	number	4	Outline stroke width if enabled, default is 4
padding	number	8	padding, default is 8
size	number	256	The size in pixels to draw, proportional, default is 256

3.2.2 BADGE

Badge arguments are passed as URL parameters and allow the user to overlay a number to icons that can be used to display the number of objects the icon represents. The following defines the available badge arguments.

parameter	type	default	description
badgeCount	number	0	Number to display within badge
badgeRadius	number	0	Radius of badge frame
badgeFontSize	number	0	Size of number to display within badge
badgeTextColor	string	empty string	Color of Badge Text. Locally defined colors are checked first. If not present, then assume CSS Color syntax. ex. "red" or "#00FF00" or "rgb(0,0,255)"
badgeFillColor	string	empty string	Fill Color of Badge. Locally defined colors are checked first. If not present, then assume CSS Color syntax. ex. "red" or "#00FF00" or "rgb(0,0,255)"
badgeBaselineAdjust	number	0.1	Baseline adjustment for badge text, a percentage of the diameter (0-1), positive value moves the text down, default is 0.1

3.2.3 DECORATOR

Decorator arguments are passed as URL parameters and allow the user to add smaller surrounding icons to an icon to represent additional capabilities the corresponding object may have or carry such as a cellphone. The following defines the available decorator arguments.

Decorator parameters example: &decorator={ "name": "cellphone", "frameColor": "red" }?decorator={ "name": "radio" }

parameter	type	req/opt	default	description
name	string	required		Unique name of decorator icon. This name should correspond to a frame definition in the symbology map.
scale	number	optional	TBD	Relative size of decorator icon in relation with base symbol

fill	boolean	optional	false	Should the frame get filled
fillColor	string	optional	white	Override frame fill color
fillOpacity	number	optional	1	Override frame fill opacity (0-1), 1 is fully opaque
frame	boolean	optional	true	Should the frame be displayed
frameColor	string	optional	black	Override frame stroke color
iconColor	string	optional	black	Override icon stroke color
monoColor	string	optional		Draw the icon as a single color, overrides all other colors