Interpreting GeoJSON Output

1 Included Data

1.1 Top level output

Element	Value
type	"FeatureCollection"
feature	List of shape, text and image features detailed below

1.2 Shape Feature

Element		Type	Required	Value
type		string	Required	"Feature"
properties		object	Required	
	label	string	Required	""
	strokeColor	string	Optional	Hex string such as "#000000"
	lineOpacity	number	Optional	Number between 0 and 1 inclusive
	fillColor	string	Optional	Hex string such as "#000000"
	fillOpacity	number	Optional	Number between 0 and 1 inclusive
	fillPattern	string	Optional	base64 encoded image
	strokeDasharray	string	Optional	Example: "2, 2"
	lineCap	number	Required	0 1 or 2
	strokeWidth	number	Required	Pixel width of lines
	strokeWeight	number	Required	Same value as strokeWidth
style		object	Required	Same data as "properties" element in SVG style
	stroke	string	Optional	Hex string such as "#000000"
	line-opacity	number	Optional	Number between 0 and 1 inclusive
	fill	string	Optional	Hex string such as "#000000"
	fill-opacity	number	Optional	Number between 0 and 1 inclusive
	stroke-dasharray	string	Optional	Example: "2 2"
	stroke-linecap	string	Required	"square" "round" or "butt"
	stroke-width	number	Required	Pixel width of lines
geometry		object	Required	Points to plot
	type	string	Required	"Polygon" or "MultiLineString"
	coordinates	number[][][]	Required	Any array of polylines where each polyline is an array of coordinate points.

1.3 Text Feature

Element		Type	Value
type		string	"Feature"
properties		object	
	label	string	Text modifier string
	pointRadius	number	0
	fontColor	string	Hex string such as"#000000"
	fontSize	string	Example: "12 pt"
	fontFamily	string	Example: "arial, sans-serif"
	fontWeight	string	"bold" or "normal"
	labelAlign	string	"left" "center" or "right"
	labelBaseline	string	"alphabetic"
	labelXOffset	number	0
	labelYOffset	number	0
	labelOutlineColor	string	Hex string such as "#000000"
	labelOutlineWidth	number	4
	rotation	number	Text rotation in degrees
	angle	number	Same value as rotation
geometry		object	
	type	string	"Point"
	coordinates	[number, number]	Number array of format [x, y]

1.4 Image Feature

Element		Type	Value
type		string	"Feature"
properties		object	
	image	string	base64 encoding of image
	rotation	number	Text rotation in degrees
	angle	number	Same value as rotation
geometry		object	
	type	string	"Point"
	coordinates	[number, number]	Number array of format [x, y]

1.5 Metadata Feature

The last feature in the list contains metadata about the symbol

Element	Type	Value
type	string	"Feature"

geometry		object	
	type	string	"polygon"
	coordinates	array	Empty array
properties		object	
	id	string	User defined from input
	name	string	User defined from input
	description	string	User defined from input
	symbolID	string	20-30 digit code corresponding to a symbol
	wasClipped	boolean	True if the rendered symbol fit inside the bounding box

2 Example Output

```
WebRenderer.RenderSymbol2D("id", "Base Camp", "Control Measure / Command and Control Areas",
"110325000012050000000000000000", "49.3,19.8 49.2,19.4 49.6,19.4 49.7,19.8", 1200, 900, "49.0,19.0,50.0,20.0",
new HashMap<>(), new HashMap<>(), WebRenderer.OUTPUT_FORMAT_GEOJSON);
```


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "label": "",
        "strokeColor": "#000000",
        "lineOpacity": 1,
        "lineCap": 2,
        "strokeWidth": 3,
        "strokeWeight": 3
      },
      "style": {
        "stroke": "#000000",
        "line-opacity": 1,
        "stroke-linecap": "square",
        "stroke-width": 3
      },
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [49.3, 19.8],
            [49.2, 19.4],
            [49.6, 19.4],
            [49.7, 19.8],
            [49.3, 19.8]
          ]
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "label": "BC",
        "pointRadius": 0,
        "fontColor": "#000000",
        "fontSize": "12pt",
        "fontFamily": "arial, sans-serif",
        "fontWeight": "bold",
        "labelAlign": "center",
        "labelBaseline": "alphabetic",
        "labelXOffset": 0,
        "labelYOffset": 0,
```

```
        "labelOutlineColor": "#FFFFFF",
        "labelOutlineWidth": 4,
        "rotation": 0,
        "angle": 0
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          49.45,
          19.59333333
        ]
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": []
      },
      "properties": {
        "id": "id",
        "name": "Base Camp",
        "description": "Control Measure / Command and Control Areas",
        "symbolID": "11032500001205000000000000000",
        "wasClipped": "false"
      }
    }
  ]
}
```

3 Plotting GeoJSON symbols in Cesium

Cesium does not include colors, labels, fills and image modifiers from GeoJSON. The following function adds any missing features to each entity.

```
async function renderGeojson(geo_json_str: string, viewer: Cesium.Viewer) {
  const geo_json = JSON.parse(geo_json_str)

  if (geo_json.type == "error") {
    console.log(geo_json.error);
    return;
  }

  let geoJsonDataSource = await Cesium.GeoJsonDataSource.load(geo_json, {
    stroke: Cesium.Color.BLACK,
    fill: Cesium.Color.TRANSPARENT,
    clampToGround: true
  })

  for (let entity of geoJsonDataSource.entities.values) {
    if (entity.polyline) {
      entity.polyline!.width = entity.properties['strokeWidth'].getValue();
      if (entity.properties['strokeColor']) {
        entity.polyline.material = new Cesium.ColorMaterialProperty(Cesium.Color.fromCssColorString(entity.
properties['strokeColor'].getValue()));
      }
    } else if (entity.polygon) {
      if (entity.properties) {
        if (entity.properties['fillPattern']) {
          entity.polygon.material = new WallPaperMaterialProperty(viewer.scene, entity.properties
['fillPattern'].getValue(), entity.polygon.hierarchy.getValue(viewer.clock.currentTime));
        }
        if (entity.properties['strokeColor']) {
          // Polygon outline requires height to be defined. Height = 0 might place polygon below terrain.
Height > 0 raises polygon above other entities.
          // Adding outline as polyline allows polygon to remain clamped to ground
          // https://community.cesium.com/t/polygon-clamp-to-ground-when-terrain-provider-is-used/22798
          // Note outline and fill might not be perfectly aligned
          // https://community.cesium.com/t/unaligned-polygon-and-polyline/9059
```

```
          entity.polyline = new Cesium.PolylineGraphics();
          entity.polyline.positions = entity.polygon.hierarchy.getValue(viewer.clock.currentTime).positions;
          entity.polyline.material = new Cesium.ColorMaterialProperty(Cesium.Color.fromCssColorString(entity.
properties['strokeColor'].getValue()));
          entity.polyline.width = entity.properties['strokeWidth'].getValue();
          entity.polyline.clampToGround = new Cesium.ConstantProperty(true);
        }
        if (entity.properties['fillColor']) {
          entity.polygon.fill = new Cesium.ConstantProperty(true);
          if (entity.properties['fillOpacity']) {
            entity.polygon.material = new Cesium.ColorMaterialProperty(Cesium.Color.fromCssColorString(entity.
properties['fillColor'].getValue()).withAlpha(entity.properties['fillOpacity'].getValue()));
          } else {
            entity.polygon.material = new Cesium.ColorMaterialProperty(Cesium.Color.fromCssColorString(entity.
properties['fillColor'].getValue()));
          }
        }
      }
    } else if (entity.billboard) {
      if (entity.properties['label']) {
        entity.billboard.color = new Cesium.ColorMaterialProperty(Cesium.Color.TRANSPARENT);
        entity.billboard.show = new Cesium.ConstantProperty(false);

        const fontColor: Cesium.Color = entity.properties['fontColor']
          ? Cesium.Color.fromCssColorString(entity.properties['fontColor'].getValue())
          : Cesium.Color.BLACK;

        const outlineColor: Cesium.Color = entity.properties['labelOutlineColor']
          ? Cesium.Color.fromCssColorString(entity.properties['labelOutlineColor'].getValue())
          : null;
        const outlineWidth: number = entity.properties['labelOutlineWidth'] ? entity.properties
['labelOutlineWidth'].getValue() : 0;

        const fontSize: string = entity.properties['fontSize'] ? entity.properties['fontSize'].getValue() :
"12pt";
        const fontFamily: string = entity.properties['fontFamily'] ? entity.properties['fontFamily'].getValue()
: "Helvetica";

        const align: string = entity.properties['labelAlign'] ? entity.properties['labelAlign'].getValue() :
"left";
        var horizontalOrigin: Cesium.HorizontalOrigin = Cesium.HorizontalOrigin.LEFT;
        if (align == "center") {
          horizontalOrigin = Cesium.HorizontalOrigin.CENTER;
        } else if (align == "right") {
          horizontalOrigin = Cesium.HorizontalOrigin.RIGHT;
        }

        entity.label = new Cesium.LabelGraphics({
          text: entity.properties['label'].getValue(),
          horizontalOrigin: horizontalOrigin,
          font: fontSize + ' ' + fontFamily,
          fillColor: fontColor,
          outlineColor: outlineColor,
          outlineWidth: outlineWidth,
          style: Cesium.LabelStyle.FILL_AND_OUTLINE,
          disableDepthTestDistance: Number.POSITIVE_INFINITY,
        });
      } else if (entity.properties['image']) {
        entity.billboard.image = entity.properties['image'];
        entity.billboard.disableDepthTestDistance = new Cesium.ConstantProperty(Number.POSITIVE_INFINITY);
        entity.billboard.rotation = new Cesium.ConstantProperty(-entity.properties['rotation'].getValue() / 360
* 2 * Math.PI);
      }
    }
  }

  viewer.dataSources.add(geoJsonDataSource);
};
```