Common Map Widget API

Version 1.1.0

Dated

12/20/2012

The intent of the Common Map Widget Application Program Interface (API) is to provide a common API for displaying and manipulating data on a map when the widget containing the map may be created by a different program than the ones creating the data widgets.

Common Map Widget API

Contents

Introduction

Background

Many programs and projects create widgets that search for or manipulate data then present the results on a map. The desire is to be able to combine data search/manipulation widgets from any provider with map widgets from other providers. In order to accomplish this, a standard way for the data search/manipulation widgets to be able to communicate with the map widget is necessary. This Application Program Interface (API) is the codification of that standard.

Overview

Using this API allows developers to focus on the problem domain rather than implementing a map widget themselves. It also allows the actual map implementation used to be chosen dynamically by the user at runtime rather than being chosen by the developer. Any map implementation that applies this API can be used. Currently, implementations using Google Earth, Google Maps V2, Google Maps V3, and OpenLayers APIs are available, and others can be written as needed.

Another benefit of this API is that it allows multiple widgets to collaboratively display data on a single map widget rather than forcing the user to have a separate map for each widget so the user does not have to learn a different map user interface for each widget.

The API uses the OZONE Widget Framework (OWF) inter-widget communication mechanism to allow client widgets to interact with the map. Messages are sent to the appropriate channels (defined below), and the map updates its state accordingly. Other widgets interested in knowing the current map state can subscribe to these messages as well.

It is worth noting that the map itself may publish data to these channels on occasion. For example, a map.feature.selected message may originate from a widget asking that a particular feature be selected or because a user has selected the feature on the map.

While in most instances the map will not echo back another message to confirm that it has performed an operation, the map will send a view status message whenever the map view (zoom/pan) has been changed, either directly by the user or by another widget that sent a view change message. This allows non-map widgets to be aware of the current viewport without having to process all the various messages that can change the non-map widget's state.

In addition, it is expected that subsequently opened widgets may wish to query the map for the current state. The API, therefore, supports widgets requesting the current status so they can configure themselves appropriately.

How to use the channels and messages

There are two ways that the map can be manipulated. The first is by other widgets via the Common Map Widget API. The second is by a user directly manipulating the map via the map Graphical User Interface (GUI) (for example, using a drawing tool on the map).

- To manipulate the map, widgets send messages in the map channels, and the map widget responds by modifying its current state. The current map state can be modified by a data source widget requesting that a Keyhole Markup Language (KML) file be loaded or that the view change to some other location. The map and any listening widget react to these change requests and modify their state accordingly.
- When the user manipulates the map, the map will post messages to the map channels, and any widget listening to the map channels can respond the changes.

In other words, if a feature is selected by a widget request or by a user, a map.feature.selected message is sent out to any widget configured to receive the message.

General Requirements

Overlays

By default, all data added by an individual widget is placed into a single overlay unique to that widget, which means one overlay per widget. However, the API supports specifying into which overlay data are inserted, so a widget can insert data in multiple overlays, and multiple widgets can insert data into the same overlay. To prevent unintended merging of data due to multiple widgets unintentionally using the same overlay ID, it is suggested that if a widget needs to use multiple overlays, and no sharing of those overlays with other widgets is intended, developers include the widget ID as part of the overlay ID. If a widget needs to share overlays with other widgets, developers should follow the guidelines in the OWF documentation regarding preference namespaces for shared overlays to avoid unintended collisions.

Features and Feature IDs

Features in the context of this document refer to the discrete pieces of data passed to the API. A feature may be a single marker, polygon, or a complex feature (for example, a KML Document containing many sub-features). The feature ID used by the API (featureId) refers to the feature ID given when plotting the entire feature. Sub-features also have IDs but their IDs are only used in the map.feature.selected message, which contains the ID of the lowest level feature selected (if available). Since feature IDs are required to be unique within an overlay, it is recommended to use an approach similar to the OWF channels: use a hierarchical naming pattern with the levels of the hierarchy separated by a dot (.). To form a unique feature ID, begin with the ID of the widget creating the feature ID. Then, the widget can generate a unique number to complete the feature ID. For example, if generating a feature ID from a widget with the name of “army.bccs.targeter”, the feature ID’s would begin with army.bccs.targeter.

Payloads

All Payloads can be either a single object or an array of like objects.

Latitude and Longitude

All latitudes and longitudes are in decimal degrees.

Errors

Any message sent that is missing a required attribute should result in the map widget publishing an error message on the error channel. An error is also published if the map widget is unable to find an object based on the given identifier. In general, any time the map is unable to complete a requested operation, an error will be published (if possible).

API

The channels associated with the Common Map Widget API are grouped according to the following:

map.overlay channels – Messages associated with creating and manipulating overlays.

map.feature channels – Messages associated with loading feature data onto the map.

map.view channels – Messages associated with manipulating the map view.

map.status channels – Messages associated with obtaining the current map state.

map.error channels – Error messages.

map.overlay channels

Create Overlay

Purpose:	Create an overlay into which data can be aggregated.
Channel:	`map.overlay.create`
Payload:	`{name: (optional), overlayId: (optional), parentId: (optional)}`
name:	The name of the overlay. If not included, the ID is used as the name. Note that overlay names do not have to be unique and are intended for display purposes only.
overlayId:	The unique ID of the new overlay. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed. If an overlay with the given ID already exists, this message will have no effect. Note that overlay IDs must be unique even across multiple parent overlays.
parentId:	The ID of the parent overlay in which to create this overlay.
Example:	`{"name": "Test", "overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

Remove Overlay

Purpose:	Remove entire overlay from the map.
Channel:	`map.overlay.remove`

Payload:	`{overlayId: (optional)}`
overlayId:	The ID of the overlay to be removed. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

Hide Overlay

Purpose:	Hide existing overlay on the map.
Channel:	`map.overlay.hide`
Payload:	`{overlayId: (optional)}`
overlayId:	The ID of the overlay to be hidden. If no overlayId is included, default overlay with ID equal to sending widget’s ID is assumed.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

Show Overlay

Purpose:	Show existing overlay on the map.
Channel:	`map.overlay.show`
Payload:	`{overlayId: (optional)}`
overlayId:	The ID of the overlay to be shown. If no overlayId is included, default overlay with ID equal to sending widget’s ID is assumed.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

Update Overlay

Purpose:	Update an existing overlay
Channel:	`map.overlay.update`
Payload:	`{ name: (optional), overlayId: (optional), parentId: (optional) }`
name:	The new name of the overlay. Note that overlay names do not have to be unique and are intended for display purposes only.
overlayId:	The unique ID of the overlay being updated. If no overlayId is included, default overlay with ID equal to sending widget’s ID is assumed. If an overlay with the given ID already exists, this message will update that overlay. If an overlay with the given ID does not exist, an error is generated. Note that overlay IDs must be unique even across multiple parent overlays.

parentId:	The ID of the parent overlay that is associated with this overlay. If no ID is provided, the overlay will keep its existing parentage. If a parentId is provided, the overlay will be changed to the new parentId.
Example:	`{"name": "New Name", "overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

map.feature channels

Plot Feature

Purpose:	Plots feature data on the map.
Channel:	`map.feature.plot`
Payload:	`{overlayId: (optional), featureId: (required), name: (optional), format: (optional), feature: (required), zoom: (optional)}`
overlayId:	The ID of the overlay this feature should be loaded into. If an overlay with this ID already exists, the new feature is merged into existing overlay; otherwise, a new overlay is created. If no overlayId is included, default overlay with ID equal to sending widget's ID is used. If an overlay exists, it will retain its status (whether visible or hidden). If an overlay is created, it will be made visible.
featureId:	Unique identifier for the given feature data. Note that feature IDs must be unique within a given overlay. Reusing a feature ID will be considered a reload, with the original feature data being removed and replaced by the new feature data.
name:	Name for the given feature data. Note that feature names do not have to be unique and are intended for display purposes only.
format:	Data format of the given feature. If no format is specified, the format defaults to "kml." A list of formats supported by a particular map implementation may be obtained by querying the map using the map.status channel (see map.status). Note that for this version of the Common Map Widget API, the only format that all map implementations *must* support is KML.
feature:	Feature data to be loaded into the map.
zoom:	*true* if map should zoom to newly loaded feature data, *false* if not. Default is false.

Example:

```
{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1",
"featureId": "example.mapWidget.1", "feature": "<kml
xmlns="http://www.opengis.net/kml/2.2"
xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:kml="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom"><Placemark
id="example.mapWidget.1.1"><name>World Trade
Center</name><description><![CDATA[Site of World Trade
Center]]></description><Style><IconStyle><Icon><href>https://loca
lhost/widgets/mapWidget/images/blu-
circle.png</href></Icon><hotSpot x="0.5" y="0" xunits="fraction"
yunits="fraction"></hotSpot></IconStyle></Style><Point><coordinat
es>-74.01324033737183,40.71149172571141,0
</coordinates></Point></Placemark></kml>", "name": "World Trade
Center", "zoom": true}
```

Notes: If using the channel shouter to send a feature, embedded quotes in KML must be escaped with a backward slash (that is, use \" instead of ").

Plot URL

Purpose: Have the map plot feature data from a Uniform Resource Locator (URL).

Channel: `map.feature.plot.url`

Payload: `{overlayId: (optional), featureId: (required), name: (optional), format: (optional), url: (required), params:(optional), zoom: (optional)}`

overlayId: The ID of the overlay this feature should be loaded into. If an overlay with this ID already exists, new feature is merged into existing overlay; otherwise, a new overlay will be created. If no overlayId is included, default overlay with ID equal to sending widget's ID is used. If overlay exists, it will retain its status (whether visible or hidden). If overlay is created, it will be made visible.

featureId: Unique identifier for the given feature data. Note that feature ids must be unique within a given overlay. Reusing a feature id will be considered a reload with the original feature data being removed and replaced by the new feature data.

name: Name for the given feature data. Note that feature names do not have to be unique and are intended for display purposes only.

format: Data format of the given feature. If no format is specified, the format defaults to “kml.” A list of formats supported by a particular map implementation can be obtained by querying the map using the map.status channel (see map.status). Note that for this version of the Common Map Widget API, all map implementations *must* support KML and WMS (GetMap only).

url: URL from which to retrieve the feature data to load onto the map

params: A JSON object containing a list of parameters to be passed to the server along with the URL when loading WMS data. Params object is ignored unless “format” is set to “wms”. Note that request, exceptions, SRS/CRS, width, height, and bbox params should not be passed in as they are determined by the map as needed and will be ignored if passed. Params as passed will be concatenated to the URL and are expected to follow the WMS specification. All parameters passed in must *not* be URL encoded (the map widget implementation will URL encode all passed in params).

zoom: *true* if map should zoom to newly loaded feature data, *false* if not. Default is false. Ignored when loading WMS data.

Example:

KML:

```
{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1",
"featureId": "example.mapWidget.2", "url":
"https://developers.google.com/kml/documentation/KML_Samples.kml"
, "name": "Samples", "zoom": true}
```

WMS:

```
{"overlayId":"xyz", "featureId":"def", "name": "Bodies of Water”,
"format":"wms", "url":"http://demo.opengeo.org/geoserver/wms",
"params": {"layers": "topp:tasmania_water_bodies", "transparent":
true, "format": "image/gif"}}
```

Notes: For version 1.1.0 of the API, “featureName” was changed to “name” for consistency with feature.plot. In order to maintain backwards compatibility with version 1.0.* of the API, it is suggested that developers on the receiving side of these messages should look for the old “featureName” if “name” is not found in the message

Unplot Feature

Purpose: Remove feature data from the map.

Channel: `map.feature.unplot`

Payload: `{overlayId: (optional), featureId: (required)}`

overlayId: The ID of the overlay where the feature to be removed is found. If no overlayId is included, default overlay with an ID equal to sending widget's ID is assumed.

featureId: The ID of the feature to be removed.

Example:

```
{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1",
"featureId": "example.mapWidget.2"}
```

Hide Feature

Purpose: Hide existing feature on the map.

Channel:	`map.feature.hide`
Payload:	`{overlayId: (optional), featureId: (required)}`
overlayId:	The ID of the overlay where the feature to be hidden is found. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed.
featureId:	The ID of the feature to be hidden.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1", "featureId": "example.mapWidget.2"}`

Show Feature

Purpose:	Have the map show previously hidden feature data.
Channel:	`map.feature.show`
Payload:	`{overlayId: (optional), featureID: (required), zoom: (optional)}`
overlayId:	The ID of the overlay where the feature to be shown is found. If no overlayId is included, default overlay with ID equal to sending widget’s ID is assumed.
featureId:	The ID of the feature to be shown.
zoom:	*true* if map should zoom to the shown feature, *false* if not. Default is false.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1", "featureId": "example.mapWidget.2"}`

Feature Selected

Purpose:	“Select,” or report that object was selected.
Channel:	`map.feature.selected`
Payload:	`{selectedId: (optional*), selectedName: (optional*), featureId: (required), overlayId: (optional)}`
selectedId:	The ID of the actual clicked object (may be a sub-feature contained within the aggregate feature data with the given featureId).
selectedName:	The name of the clicked object.
featureId:	The ID of the feature that contains the clicked object.
overlayId:	The ID of the overlay which contains the clicked object. If no overlayId is included, default overlay with ID equal to sending widget’s ID is assumed.

Example:	`{"selectedId": "example.mapWidget.1.1", "selectedName": "World Trade Center", "featureId": "example.mapWidget.1", "overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`
Notes:	Although both selectedId and selectedName are optional, one must be passed in if a sub-feature is to be identified. Generally, selectedId is preferred and selectedName is used when no selectedId is available.

Update Feature

Purpose:	Update an existing feature on the map.
Channel:	`map.feature.update`
Payload:	`{overlayId: (optional), featureId: (required), name: (optional), newOverlayId: (optional)}`
overlayId:	The ID of the overlay where the feature to be updated is currently found. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed.
featureId:	The ID of the feature to be updated.
name:	If the name provided differs from the feature's current name, the display value will be changed to show the new value. If no value is provided, the feature name will remain unchanged.
newOverlayId:	This represents the ID of an overlay to move the feature to. If this attribute is provided, the feature should be removed from its current overlay and added to the overlay with this ID.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1", "featureId": "example.mapWidget.2", "name": "New Name"}`
Notes:	In order to update the feature data itself, use the map.feature.plot or map.feature.plot.url message and use the existing featureId.

map.view channels

Zoom

Purpose:	Zoom the map to a particular range.
Channel:	`map.view.zoom`
Payload:	`{range: (required)}`
Range:	The distance in meters from the map (note that most 2-D map renderers do not support infinite zoom and the range will be translated into the nearest supported zoom level).

Example:	`{"range": 100000}`

Center on Overlay

Purpose:	Center the map on a particular overlay. The map may also be zoomed to show the entire overlay (if possible) or to show a given range.
Channel:	`map.view.center.overlay`
Payload:	`{overlayId: (optional), zoom: (optional)}`
overlayId:	The ID of the overlay to center on. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed.
zoom:	If "auto," zoom will adjust to best fit the overlay in the user's viewable area. If a number, map will zoom to specified range in meters. If no zoom attribute is included, no zoom is performed.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1"}`

Center on Feature

Purpose:	Center the map on a particular feature. The map may also be zoomed to show the entire feature (if possible) or to show a given range.
Channel:	`map.view.center.feature`
Payload:	`{overlayId: (optional), featureId: (required), zoom: (optional)}`
overlayId:	The ID of the overlay where the feature to be centered on is found. If no overlayId is included, default overlay with ID equal to sending widget's ID is assumed.
featureId:	The ID of the feature to center on.
zoom:	If "auto," map will adjust to best fit the feature in the user's viewable area. If a number, map will zoom to specified range in meters. If no zoom attribute is included, no zoom is performed.
Example:	`{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1", "featureId": "example.mapWidget.1"}`

Center on Location

Purpose:	Center the map on a particular location. The map may also be zoomed as close as possible to the location or to a given range.
Channel:	`map.view.center.location`
Payload:	`{location: {lat: (required), lon: (required)} (required), zoom: (optional)}`

location: Location to be centered in map.

- lat: The latitude value of the point.
- lon: The longitude value of the point.

zoom: If "auto," map will adjust to zoom as close as possible to the given location in the user's viewable area.
If a number, map will zoom to specified range in meters.
If no zoom attribute is included, no zoom is performed.

Example: `{"location": {"lat": 38.8708, "lon": -77.0558}, "zoom": 1000}`

Center on Bounds

Purpose: Center the map on a particular bounding box. The map may also be zoomed to show the entire bounds (if possible) or to show a given range.

Channel: `map.view.center.bounds`

Payload: `{bounds: {southWest: {lat: (required), lon: (required)} (required), northEast: {lat: (required), lon: (required)}} (required), zoom: (optional)}`

bounds: Bounding box of area to be centered in map.

southwest:

- lat: The latitude value of the southwest point.
- lon: The longitude value of the southwest point.

northeast:

- lat: The latitude value of the northeast point.
- lon: The longitude value of the northeast point.

zoom: If "auto," map will adjust to best fit the bounds in the user's viewable area.
If a number, map will zoom to specified range in meters.
If no zoom attribute is included, no zoom is performed.

Example: `{"bounds": {"southWest": {"lat": 24.5, "lon": -124}, "northEast": {"lat": 50.5, "lon": -79}}, "zoom": 3000000}`

Map Clicked

Purpose: "Click," or report that map was clicked.

Channel: `map.view.clicked`

Payload:	`{lat: (required), lon: (required), button: (required), type: (required), keys: (required)}`
lat:	The latitude of the location that was clicked.
lon:	The longitude of the location that was clicked.
button:	Which mouse button was clicked. Allowable values are “right,” “left,” and “middle.” For backwards compatibility, if this attribute is not populated it should be treated as a left mouse click the same as if it were populated with “left.”
type:	The type of click event. Allowable values are “single” and “double.” For backwards compatibility, if this attribute is not populated it should be assumed to be a single mouse click and treated the same as if it were populated with “single.”
keys:	An array of keys pressed during the click event. Allowable values are “alt,” “ctrl,” “shift,” and “none.” For backwards compatibility, if this attribute is not populated it should be assumed that no additional keys were pressed and behave the same way as if it were populated with “none.”
Example:	`{"lat": 40.195659093364654, "lon": -74.28955078125, "button": "right", "type": "single", "keys": ["shift","ctrl"]}`

map.status channels

Request Map Status

Purpose:	Request current status from the map. Map will send out requested map.status messages in response.
Channel:	`map.status.request`
Payload:	`{types: (optional)}`
types:	An array of status types being requested. Currently only status types of “about,” “format,” and “view” are supported (future versions are expected to support a larger family of statuses, perhaps including “overlay,” “feature,”or “selected”). If types attribute is not included, all status types will be generated.
Example:	`{"types": ["view", "about"]}`

Map View Status

Purpose:	Send out the current status of the map view.
Channel:	`map.status.view`
Payload:	`{requester: (optional), bounds: {southWest: {lat: (required), lon: (required)} (required), northEast: {lat: (required), lon:`

`(required)}}, center: { lat: (required), lon: (required)} (required), range: (required)}`.

requester: Client that requested this status message be sent (if any). If no requester, message is being sent due to a map view change.

bounds: Bounding box of area visible on map.

southwest:

- lat: The latitude value of the southwest point.
- lon: The longitude value of the southwest point.

northeast:

- lat: The latitude value of the northeast point.
- lon: The longitude value of the northeast point.

center: The current center of the map.

- lat: The latitude value of the point.
- lon: The longitude value of the point.

range: The current distance, in meters, map is zoomed out.

Example:
```
{"requester": "217c086f-719f-928f-5e75-972530cf0db6", "bounds":
{"southWest": {"lat": 39.46164364205549, "lon": -
75.6134033203125}, "northEast": {"lat": 40.97575093157534, "lon":
-73.10302734375}}, "center": {"lat": 40.2205, "lon": -74.3579},
"range": 137500}
```

Map Format Status

Purpose: Send out the list of data formats that the map widget supports; in other words, this map implementation supports the following feature formats.

Channel: `map.status.format`

Payload: `{formats: [] (required)}`

format: An array of the formats that this map supports. Note that for this version of the Common Map Widget API, all map implementations *must* support KML, and WMS. Additional map formats (for example, GeoJSON) may be supported

Example: `{"formats": ["kml", "geojson", "wms"]}`

Map About Status

Purpose:	Send out static information about this map implementation.
Channel:	`map.status.about`
Payload:	`{version: (required), type: (required), widgetName: (required)}`
version:	The version numbers of the Common Map Widget API that this map widget supports.
type:	The type of map in the map widget. Allowable values are “2-D,” “3-D,” or “other.”
widgetName:	The name of the map widget.
Example:	`{"version": "1.0.0", "type": "2D", widgetName: "Common Map Widget"}`

Map Error Channels

Error

Purpose:	Map Widget reports errors occurred when attempting to process any message.
Channel:	`map.error`
Payload:	`{sender: (required), type: (required), msg: (required), error: (required)}`
sender:	Sender of message that caused error.
type:	Type of message that caused error.
msg:	The message that caused the error
error:	A description of the error.
Example:	`{"sender": "29ff882d-4ef3-4ea2-852e-de799dd0699d", "type": "map.feature.hide", "msg": "{"overlayId": "2d882141-0d9e-59d4-20bb-58e6d0460699.1", "featureId": "example.mapWidget.doesntExist"}", "error": "No feature with id example.mapWidget.doesntExist found to hide."}`

Other

Drag and Drop

Purpose:	Drop an item on the map.
Channel:	NA – See OWF drag drop API for details

Payload:

```
dragDropData: {overlayId: (optional), featureId: (required),
name: (optional), zoom: (optional), marker: {details: (optional),
iconUrl: (optional)} (optional), feature: {format: (required),
featureData: (required)} (optional), featureUrl: {format:
(required), url: (required), params: (optional)}
```

overlayId: The ID of the overlay the dropped item should be loaded into. If overlay with this ID already exists, new item is merged into existing overlay; otherwise, new overlay will be created. If no overlayId is included, sending widget's ID is used.

featureId: Unique identifier for the dropped feature. Note that feature IDs must be unique within a given overlay. Reusing a feature ID will be considered a reload, with the original feature data being removed and replaced by the new feature data.

name: Name for the given feature data. Note that feature names do not have to be unique and are intended for display purposes only.

Zoom: *true* if map should zoom to newly loaded feature or marker data, *false* if not. Default is *false*.

marker: A JSON object containing information with which to create a new marker on the map.

- details: Detail text associated with the item that will appear in an info window.
- iconUrl: URL to an icon that represents this dropped item on the map. If no URL is included, a default icon is used.

feature: A JSON object containing feature data to be plotted on the map.

- format: Data format (http header content type) of the given feature. If no format is specified, the format defaults to "kml." A list of formats supported by a particular map implementation can be obtained by querying the map using the map.status channel (see map.status). Note that for this version of the Common Map Widget API, all map implementations *must* support KML
- featureData: Feature data to be loaded into the map.

featureUrl: A JSON object containing URL information for feature data to be plotted on the map.

- format: Data format of the given feature. If no format is specified, the format defaults to "kml." A list of formats supported by a particular map implementation can be obtained by querying the map using the map.status channel (see map.status). Note that for this version of the Common Map Widget API, all map implementations *must* support KML and WMS (GetMap only).
- url: URL from which to retrieve the feature data to load onto the map
- params: A JSON object containing a list of parameters to be passed to the server along with the URL when loading WMS data. Params object is ignored unless "format" is set to "wms." Note that request, exceptions, SRS/CRS, width,

height, and bbox params should not be passed in as they are determined by the map as needed and will be ignored if passed. Params as passed will be concatenated to the URL and are expected to follow the WMS specification. All parameters passed in must *not* be URL encoded (the map widget implementation will URL encode all passed in params).

Example:

```
{"dragDropData": {"overlayId": "Spot Reports", "featureId":
"2d882141-0d9e-59d4-20bb-58e6d0460699.1”, "name": "A Spot
Report", "zoom": "true", "marker": {"details": "Spot report
details here", "iconUrl":
“http://localhost/widgets/mapWidget/images/spotReport.gif”}}
```

NOTE: Although marker, feature, and featureUrl are optional, one must be present.

If marker is included, the marker will be placed at the location of the drop. Feature and featureUrl data will be placed at their natural location (equivalent to being loaded using map.feature.plot.* message).

Acronyms

API	Application Program Interface
GeoJSON	Geo JavaScript Object Notation
GUI	Graphical User Interface
KML	Keyhole Markup Language
Lat	Latitude
Lon	Longitude
OWF	OZONE Widget Framework
URL	Uniform Resource Locator

References

GeoJSON specification can be accessed at http://www.geojson.org/geojson-spec.html

KML specification and related documents can be accessed at https://developers.google.com/kml/documentation/

WMS specification can be accessed at http://www.opengeospatial.org/standards/wms