External interface description JavaScript client

Version 3.1

Last update 6/11/2015

Publisher:

alta4 AG

Fleischstraße 57
54290 Trier
Germany

Fon: +49.651.96626.0
Fax: +49.651.96626.26

www.alta4.com
info@alta4.com

Table of contents

1 General

The external programming interface of the atlasFX Javascript web client conduces to the programmatic control by external scripts. With the dojo utility atlasExternalInterface ("com/alta4/atlas/extensions/AtlasExternalInterface") functions are provided to control the behavior of the map application.

Note: In future you'll find the most recent version of this document here.

2 API Spezifikation

`setLayerFilter(layerId, whereExpression)`
Sets a filter for a feature layer, which is specified by a layerId. The syntax of the where-clause is given by each addressed ArcGIS-Server-API.

`getExtentWebMercator()`
Returns the extent of the map in WebMercator coordinates.

`getExtentGeographic()`
Returns the extent of the map in longitude and latitude.

`setCenter(x, y)`
Centers the map to the transferred point. The coordinates must be in the same coordinate system as the map.

`setCenterWgs84(lat, long)`
Centers the map to the transferred point. The coordinates must be specified as longitude/latitude.

`setScale(scale)`
Sets the scale of the map. If the map should be displayed at a scale of 1:50.000, a value of 50000 for the parameter scale has to be defined.

`getScale()`
Returns the current scale of the map.

`setCenterAndZoom(x, y, levelOrFactor)`
Combination of the functions above, while the coordinates are taken from the coordinate system of the map and optionally a zoom level or factor is configured.

`setCenterAndScale(mapPoint, scale)`
As above, but a scale factor has to be set.

`setInfoWindowLinkClickCallback(callback)`
When a link inside an info bubble is invoked, first the callback method is called.
`callback(url)`: Callback returns the pressed URL. The callback-method has to return false if the execution of the link should be suppressed.

`wgs84ToWebMercator(lat, lon)`
Transforms latitude/longitude into WebMercator coordinates.

`refreshLayers()`
Forces a redraw of all layers.

`registerGraphicCallback(layerId, callback)`
Registers a click-callback upon the given layer for graphics which are placed on it.

`removeGraphicCallback(layerId)`
Removes the callback-routines which have been registered via registerGraphicCallback before.

`showFeature(layerId, featureId, showInfoBubble, scale)`
Centers the map to the feature which is identified with the layerId and the featureId. With the boolean showInfoBubble it is possible to define whether the respective info window should be displayed, as long as the property of the feature is configured. The scale parameter is only defined for point features and is optional. Geometries containing an extent (line, polygon) are always displayed in the maximum zoom level, provided that the feature is fully visible.

`setMouseFeatureInteraction(layerTypes, event, procedure, handler)`
Extends or replaces the default mouse interaction with map features. Allowable parameter values:

`layerTypes`: One of the following strings or a combination of them as a n array.
POINT, LINE, POLYGON, CLUSTER, SINGLE_FEATURE_CLUSTER
The last two values are related to the clustered scale ranges. A distinction is made between "real" clusters and those which contain only a single feature and are shown as a normal point feature.

`event`: mouseOver, mouseOut, click
The treated mouse events.

`procedure`: before, after, replace
This parameter determines how the passed function `handler` is to be mounted. If the mounted handler is executed after the default function or replaces it, the mouse event is passed to the `handler`.

Note: In future you'll find the most recent version of this document here.

3 Optional Tools

In the following, functions are listed which are only available if the respective tools are loaded and activated. Not every tool belongs to the standard functionality and can be configured in the CMS.

Note: In future you'll find the most recent version of this document here.

3.1 Routing Tool

The routing tool extends, if loaded and configured, the external interface. Then, the following functions are available globally at `atlasExternalInterface.routingTool`.

`activate()`
Activates the click-routing-functionality, which means that waypoints can be added by clicking on the map. The points are characterized by three configurable symbols (start, stop, stopover).

`deactivate()`
Deactivates the click-routing-functionality in the map. The representation of the current route planning, both in the list view and in the map, are not affected.

`highlightPosition(geometry)`
At the coordinate, which is defined by geometry, a configurable highlighting symbol is drawn. It is placed on a special marker layer between the route and the waypoints. This function comes into play when hovering over the tabular route presentation to allow a fast acquisition of the waypoint on the map.

`clearHighlighting()`
Deletes the whole marker layer.

`addStop(geometry, title, force)`
Adds a new location to the route model which is defined by geometry. Here a point geometry is expected. If title is undefined, the address is automatically detected with a service for reverse geocoding and used as the title for the tabular representation. The parameter force forces an immediate transfer of the data into the model. In case of false, the synchronization of the route planner with the feature-click behavior is interposed. Namely, if a feature symbol is clicked, only the information window should be displayed first. With enabled click-routing a button will be available in the window allowing to add the feature to the route. In this case the title of the waypoint applies the configured title of the information window.

`addStopByAddress(address, plz, city)`
With a configured geoservice there is an attempt to georeference an address. If successful, the result is added as a waypoint to the route. The title then contains the candidate address of the geocoding result.
Adding a waypoint causes the route calculation of at most one route segment between the added location and its

predecessor. If successful, the route segment is drawn. If one of the points involved is not traversable, the segment is not applied to the model.

`removeStop(routeSegment)`
Removes the route segment (a point and the last route section which leads to him). If necessary, the segment between the remaining sub-routes will be recalculated.

`clearRoute()`
Deletes the whole route.

`ZoomToExtent()`
Zooms and moves the map so that the route can be seen entirely and nearly fills the full extent.

`ExportRouteJSON()`
Exports the route as a complex segment so that the reconstruction of the internal data model is possible.

`ExportSimplifiedRouteJSON()`
Exports a simplified form of the route, with start and end point as a geometry.

`closeRoute()`
Closes the route to a loop road.

Note: In future you'll find the most recent version of this document here.