HERE Maps API - Managing map Features

This article explains how to use HERE Maps API class `Nokia.Features`.

Introduction

Since the version 2.0.0 of the HERE Maps API, it will offer the possibility to display, load and manage Maps features including e.g. “routing” and “search”.

Each of the features may exist in different implementations for different target environments. A feature implementation is a set of classes that enable certain functionality and which can be loaded to enrich the functionality.

By default the API loads all possible features in the optimal implementation for the environment in which it runs. However, the loading of certain features may also be deferred by explicitly disallowing certain or all features at page load time. This may reduce the initial page load time as not all of the API's code is transferred and executed during the page's start up.

The `Features` object makes it possible to load individual features of the API manually. It automatically resolves dependencies between different features and provides mechanisms to detect the best feature implementation for the current environment.

The `Features` object is a static object that is not to be instantiated, it is intended to be used directly instead.

Example use

In the example below we first need to have the HERE Maps instantiated. Please use the link to get a basic HERE Maps onto a webpage:

- Basic Map

You can then continue using either Firefox with Firebug, or Chrome's tools or whichever debugging tool you prefer.

After this you can just use the web development console for testing as follows:

For example, if we would like to see what is currently loaded, we could issue the following:

```
nokia.maps.Features.getFeatureMap();
```

Firebug Object Display shows:

```
Object { search=[1], routing=[1], positioning=[1], more...}
```

Based on this information, we can find out that all the components that are listed are currently loaded, and we will also be able to see how many instances there are of them, so "[1]" means there is one loaded.

You can reveal the rest with “more…”. Please note that it depends on your map implementation of course, how many features there are, this is just given as an example. After clicking "more..." you could see e.g.:

```
gfx
["canvas", "svg", "vml"]
kml
["kml"]
language
["en-GB", "en-US", "de-DE", 5 more...]
map
["js-p2d-canvas", "js-p2d-dom"]
positioning
["w3c"]
routing
["nokia"]
search
```


(C) Copyright Nokia 2014. All rights reserved.

```
["nokia"]
ui
["nokia_generic", "ovi_web"]
```

The method `getLoadedMap()` will return a map of all registered feature implementations per feature.

To use this information in our own code, we could do e.g. like this:

```
myFeatureMap = new nokia.maps.Features.getLoadedMap();
alert("You have the current UI feature loaded: "+myFeatureMap.ui);
```

An `Alert` window could display now : "You have the current UI feature loaded: nokia_generic"

Please note: In the version 2.0 of the API the old UI package "ovi_web" is still accessible, but will be deprecated in an oncoming release.

With the method `isLoaded()` we can find out if certain feature implementation has been successfully loaded. The method returns a Boolean value of "`true` / `false`". The query will result in `true` only if the queried implementation was **fully** loaded, meaning it was both transmitted and evaluated. The method will throw an exception if either feature or implementation are unknown.

The following code line would issue a query to check if the UI feature's ovi_web implementation is loaded.

```
ovi_web_feature_loaded = nokia.maps.Features.isLoaded("ui","ovi_web");
alert("UI feature ovi_web load status : "+ovi_web_feature_loaded);
```

In our case the query returned a value of `false`.

Loading a new set of specified features is done with the "load" method. Optionally you can set an `onSuccess` and an `onError` callback, plus define the target document into which the respective script tags should be appended. The feature request is expected to be specified as an object where the keys represent the features and their values represent the implementation name.

As in the previous code we had checked for the existence of the "ovi_web" UI component, and got back the result that it is not loaded (`false`) then we could attempt to load it as follows:

Loading e.g. both "ovi_web" UI and "search":

```
>>> nokia.Features.load({"ui":"ovi_web", "search":"auto"});
```

Check if the load was successful. Please note that in an online app you should use the callbacks to check for success.

```
>>> ovi_web_feature_loaded = nokia.maps.Features.isLoaded("ui","ovi_web");
    alert("UI feature ovi_web load status : "+ovi_web_feature_loaded);
```

In this case we had a result `true` for our query.

A working example which loads a map and requests all the features loaded can be found at:

http://rawgithub.com/heremaps/examples/master/maps_api_for_javascript/advanced-examples/reading-map-features.html

Summary

For more on `Features`, please see the HERE Maps API reference at :

http://developer.here.com/docs/maps_js/topics_api_pub/nokia.Features.html

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.