Geomajas WMS client plug-in guide

Geomajas Developers and Geosparc

Geomajas WMS client plug-in guide

by Geomajas Developers and Geosparc

Version 2.3.0

Table of Contents

Chapter 1. Introduction

This plugin provides client side WMS (Web Map Service) support. It defines services to communicate with WMS servers and it also provides a client-side layer definition.

Note

In the Geomajas server project there is also a WMS layer plugin that provides a server-side WMS layer. This plugin however provides a client-side WMS layer.

1. Client-side WMS versus server-side WMS

As mentioned, Geomajas also has the Geomajas-Layer-WMS plugin in the server project. That plugin defines a server-side WMS layer. The main difference is that the Geomajas-Layer-WMS plugin defines the WMS layers in the backend map composition, making it available to all users. This client-side WMS plugin on the other hand is able to define client WMS layers. These are available only on the client that adds them, making them more flexible.

2. Plugin structure

This plugin contains multiple artifacts. It provides a basic set of functionalities that do not require the Geomajas server project, and an extension that does require a dependency to the Geomajas server project. In short, the server extension provides support for the WMS GetFeatureInfo request.

- *wms*: The basic WMS client functionality. It contains a WmsLayer and support for GetCapabilities, GetMap and GetLegendGraphic.
- *example-jar*: Jar that contains showcase examples.
- *documentation*: This documentation.

3. Dependencies

Basic WMS client artifact:

```
<dependency>
    <groupId>org.geomajas.plugin</groupId>
    <artifactId>geomajas-client-gwt2-plugin-wms</artifactId>
    <version>2.3.0</version>
</dependency>
```

4. Versions

This plugin has support for WMS version 1.1.1 and 1.3.0.

Chapter 2. Configuration

1. Getting started

As most Geomajas plugins, the WMS client plugin too has a single class that provides a starting point for most functionalities the plugin supports. This starting point is the `WmsClient` class:

`org.geomajas.gwt2.plugin.wms.client.WmsClient`

From here on it is possible to create new WMS layers or execute WMS GetCapabilities requests, or ...

1.1. Using the WmsService

Before we explain how to create a client-side WMS layer, let us first delve into the inner workings of the WMS client plugin. It provides a client WMS service definition that supports WMS requests: `org.geomajas.gwt2.plugin.wms.client.service.WmsService`

`This WmsService provides support for the following WMS requests:`

- *GetCapabilities*: Executes a WMS GetCapabilities and parses the result into Java objects. This capabilities object allows you to read the WMS contact information, the list of supports coordinate reference systems, the list of known layers, etc.
- *GetMap*: A GetMap request gets an image for a certain location. This request is used by the layer renderer.
- *GetLegendGraphic*: A GetLegendGraphic request gets an image for the legend of a certain layer. This legend displays styling information.

The WmsService is a singleton service that can be acquired as follows:

```
WmsService wmsService = WmsClient.getInstance().getWmsService();
```

Using this WmsService it is possible to directly execute the supported WMS requests. There is no need for a WMS layer.

1.2. Automatic URL transformation

One of the most common problems when executing Ajax requests from within the browser is the security regarding cross domain requests. The WMS server you want to communicate with, will most likely be located on a different server that your web application.

For this purpose, the `WmsService` allows you to specify a `WmsUrlTransformer` object that will transform WMS request URLs to your liking. It can go as far as to specify different transformations for different types of WMS request. For example, you may want the `GetCapabilities` and `GetFeatureInfo` requests to make use of a proxy servlet, while your `GetMap` and `GetLegendGraphic` requests must remain untouched (`GetMap` and `GetLegendGraphic` URLs point to images, we apply them directly onto an images href attribute, where the cross-domain problem does not exist).

The following example will add a proxy servlet for the `GetCapabilities` and `GetFeatureInfo` requests, while the other WMS requests are left untouched:

```
wmsService.setWmsUrlTransformer(new WmsUrlTransformer() {

    public String transform(WmsRequest request, String url) {
        switch (request) {
```

```
            case GetCapabilities:
            case GetFeatureInfo:
                return "/proxy?url=" + url;
            default:
        }
        return url;
    }
});
```

2. WMS layers

2.1. Configuration objects

Before we go into the actual creating of a WMS layer, let us first cover the required configuration objects.

The first is the `WmsTileConfiguration` object. It provides the point of origin and the size of the tiles in pixels:

- *tileOrigin*: The origin for the layer (coordinate that provides the minimum X and Y values). If you're unsure, it's best to take a margin.
- *tileWidth*: The width in pixels for an individual tile.
- *tileHeight*: The height in pixel for an individual tile.

The other is the `WmsLayerConfiguration` object. It provides all the parameters used in the `GetMap` requests for a specific layer. These are the following:

- *baseUrl*: The base URL to the WMS server (without any WMS params!)
- *format*: The format for the images. The default value is "image/png".
- *layers*: The GetMap "layers" parameter. It provides the layer(s) to display.
- *styles*: The GetMap "styles" parameter. It provides the style for the requested layer.
- *filter*: Optional CQL parameter. This is not part of the WMS specification, but some WMS servers do support it.
- *transparent*: The GetMap "transparent" parameter. The default value is "true".
- *version*: The WMS version. The default value is 1.3.0.

2.2. Creating a WMS layer

Once you have your configuration objects (WmsTileConfiguration and WmsLayerConfiguration) it's easy to create a new layer:

```
// First we define a WMS configuration object:
WmsLayerConfiguration layerConfig = new WmsLayerConfiguration();
layerConfig.setFormat("image/jpeg");
layerConfig.setLayers("bluemarble");
layerConfig.setVersion(WmsVersion.v1_1_1);
layerConfig.setBaseUrl("http://apps.geomajas.org/geoserver/wms");

// Then we define a Tile Configuration object:
Coordinate tileOrigin = new Coordinate(-360,-180);
WmsTileConfiguration tileConfig = new WmsTileConfiguration(256, 256, tileOrigin
```

```
// Now create the layer:
WmsLayer wmsLayer = WmsClient.getInstance().createLayer("Blue Marble", tileConfi
```

Once you have your layer, it is possible to add it to the map:

```
mapPresenter.getLayersModel().addLayer(wmsLayer);
```

Et voila! A newly created client-side WMS layer has been added to the map!

2.3. Changing the WMS layer configuration

As mentioned earlier, the WMS layer is build using a WmsLayerConfiguration object, which is used internally to build GetMap requests. Should you change any of the parameters in this configuration object, it's effects will immediately take place (or at least by refreshing the layer).

You can acquire the layer configuration objects from the layer itself. The following example takes this configuration to change the layer style on the fly:

```
WmsLayerConfiguration layerConfig = wmsLayer.getConfig();
layerConfig.setStyles("alternativeStyle");
```

Setting the styles parameter is a special case, because it will fire a `LayerStyleChangedEvent`, causing the layer to update automatically. No refresh is needed.

2.4. Using WMS GetCapabilities

Ofcourse one does not always know beforehand which WMS layers to add. Sometimes we need to ask a WMS Server for it's available layers, and present the user with a choice. This can be achieved through the GetCapabilities request, the result of which can than be used to create WMS layers.

A WMS GetCapabilities request returns an object of the type `WmsGetCapabilitiesInfo`. The following example executes a GetCapabilities request:

```
String wmsBaseUrl = "http://apps.geomajas.org/geoserver/demo_world/ows";
WmsClient.getInstance().getWmsService()
    .getCapabilities(wmsBaseUrl, WmsVersion.V1_1_1, new Callback<WmsGetCapabili

        @Override
        public void onSuccess(WmsGetCapabilitiesInfo capabilities) {
            // Do something with the result...
        }

        @Override
        public void onFailure(String reason) {
            Window.alert("We're very sorry, but something went wrong: " + reason
        }
    });
```

This object contains layer descriptions of the type `WmsLayerInfo`, which can be accessed as follows:

```
WmsGetCapabilitiesInfo capabilities; // Type of object in a GetCapabilities requ
List<WmsLayerInfo> layerInfo = capabilities.getLayers();
```

Now instead of having to set the `WmsTileConfiguration` and `WmsLayerConfiguration` parameters manually, we can use the values from the `WmsLayerInfo` object which came from the GetCapabilities request. Luckily we don't have to do this ourselves. The `WmsClient` helps out. Here is an example for creating a `WmsTileConfiguration` object:

```
// We need the tile origin in the correct coordinate system, so we use that of t
String crs = mapPresenter.getViewPort().getCrs();

// Now create a tile config object:
WmsTileConfiguration tileConfig = WmsClient.getInstance().createTileConfig(layer
```

Next up is the WmsLayerConfiguration object:

```
String wmsBaseUrl = "http://apps.geomajas.org/geoserver/demo_world/ows";
ViewPort viewPort = mapPresenter.getViewPort();

// Now create the layer config object:
WmsLayerConfiguration layerConfig = WmsClient.getInstance().createLayerConfig(
                        viewPort, layerInfo, wmsBaseUrl, WmsVersion.V1_1_1);
```

We now have our configuration objects from which to create a WMS layer. If we put it all together we get:

```
String wmsBaseUrl = "http://apps.geomajas.org/geoserver/demo_world/ows";
WmsClient.getInstance().getWmsService()
    .getCapabilities(wmsBaseUrl, WmsVersion.V1_1_1, new Callback<WmsGetCapabili

        @Override
        public void onSuccess(WmsGetCapabilitiesInfo result) {
            if (result.getLayers() != null && result.getLayers().size() > 0) {
                WmsLayerInfo layerInfo = result.getLayers().get(0);
                WmsTileConfiguration tileConfig = WmsClient.getInstance().create
                        mapPresenter.getViewPort().getCrs(), 256, 256);
                WmsLayerConfiguration layerConfig = WmsClient.getInstance().crea
                        mapPresenter.getViewPort(), layerInfo, WMS_BASE_URL, Wms
                WmsLayer layer = WmsClient.getInstance().createLayer(layerInfo.
                        tileConfig, layerConfig, layerInfo);
                mapPresenter.getLayersModel().addLayer(layer);
            }
        }

        @Override
        public void onFailure(String reason) {
            Window.alert("We're very sorry, but something went wrong: " + reason
        }
    });
```