Geomajas geomajas-plugin-wmsclient plug-in guide

Geomajas Developers and Geosparc

Geomajas geomajas-plugin-wmsclient plug-in guide

by Geomajas Developers and Geosparc

1.0.0-M1

Table of Contents

List of Examples

Chapter 1. Introduction

This plugin provides client side WMS (Web Map Service) support. It defines services to communicate with backend WMS services and it also provides a client-side layer definition. The main difference with the Geomajas-Layer-WMS plugin is that the layers are defined client-only. That means that they are not part of the backed map composition. That in turn means .....

1. Client-side WMS versus server-side WMS

As you may know, Geomajas also has the Geomajas-Layer-WMS plugin. 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.

Note that if your WMS server is located on another domain, the browser will prohibit you from accessing it directly. In that case you will need to go through the server anyway, although a proxy servlet should suffice. If you need extra security measures, this proxy servlet would be the place to do it. The serverside WMS layer already has security measures built in.

2. Supported functionalities

2.1. WMS Requests

This plugin provides a client WMS service definition that supports WMS requests: `org.geomajas.plugin.wmsclient.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.
- *GetFeatureInfo*: The GetFeatureInfo request is used to request feature information for a specific location. Depending on the return type, this information can be parsed as a list of Feature objects.

2.2. Layer definitions

This plugin provides not one, but two layer definitions:

- *WmsLayer*: The basic WMS layer definition. This will display a WMS layer on the map, and also supports the GetLegendGraphic to display the legend.
- *FeaturesSupportedWmsLayer*: An extension of the basic WMS layer definition that also support the GetFeatureInfo request. Not all WMS layer will support this feature.

2.3. Map Controllers

By default this plugin provides the following MapControllers:

- *WmsGetFeatureInfoController*: Controller that executes GetFeatureInfo requests on mouse clicks. This controller does not extend the NavigationController, and can best be used as a MapListener next to the default NavigationController.

2.4. GIN

This plugin extends the default Geomajas injection definitions, by providing it's own GIN module definition and GIN injector interface. Through this GIN injector, one can create new WMS Layers or acquire the WmsService singleton.

3. Versions

This plugin has support for WMS version 1.1.1 and 1.3.0.

Chapter 2. Configuration

1. Dependencies

Make sure your include the plug-in in your project. If you are using Maven, add the following dependency to your pom:

Example 2.1. Plug-in dependency

```
<dependency>
    <groupId>org.geomajas.plugin</groupId>
    <artifactId>geomajas-plugin-wmsclient</artifactId>
    <version>1.0.0</version>
</dependency>
```

Chapter 3. How-to

1. How to define a GIN injector that used both PureGWT face and the WMS client plugin

This plugin extends the default Geomajas injection definitions, by providing it's own GIN module definition and GIN injector interface. If you have an application wherin you want to define a GIN injector that supports the GIN modules of both the PureGWT face and the WMS client plugin, you can define it as follows:

```
@GinModules({ GeomajasGinModule.class, WmsClientGinModule.class })
public interface MyApplicationGinjector extends WmsClientGinjector, GeomajasGin
}
```

2. How to create a new WMS Layer and add it to the map

Creating a new WmsLayer instance, is done through the GIN injection framework. You need to define a GIN injector (see previous paragraph), and have this injector create the instance for you:

```
// It is best to define your own injector, and make it a global static variable
// (you need only one for your app)
WmsClientGinjector injector = GWT.create(WmsClientGinjector.class);

// We create a new map:
MapPresenter mapPresenter = injector.getMapPresenter()
mapPresenter.initialize("puregwt-app", "mapEmpty"); // Map with an empty config
mapPresenter.getEventBus().addMapInitializationHandler(new MapInitializationHan

    // When the map has been initialized, we can start adding WMS layers:
    public void onMapInitialized(MapInitializationEvent event) {

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

        // Then we define a Tile Configuration object:
        Coordinate tileOrigin = new Coordinate(mapPresenter.getViewPort().getMa
                mapPresenter.getViewPort().getMaximumBounds().getY());
        WmsTileConfiguration tileConfig = new WmsTileConfiguration(256, 256, ti

        // Create the WMS layer and add it to the map:
        WmsLayer wmsLayer = injector.getWmsLayerFactory().createWmsLayer("bluem
                wmsConfig, tileConfig);
        mapPresenter.getLayersModel().addLayer(wmsLayer);
    }
});
```

3. How to read a GetCapabilities file and get the list of layers

For parsing a GetCapabilities file, we need the WmsService. This service must be acquired through the GIN injector:

```
// It is best to define your own injector, and make it a global static variable
// (you need only one for your app)
WmsClientGinjector injector = GWT.create(WmsClientGinjector.class);

// Get the WMS Service:
WmsService wmsService = injector.getWmsService();

wmsService.getCapabilities("http://apps.geomajas.org/geoserver/wms", WmsVersion
        new Callback<WmsGetCapabilitiesInfo, String>() {

    public void onFailure(String reason) {
        Window.alert(reason);
    }

    public void onSuccess(WmsGetCapabilitiesInfo result) {
        // Get the list of layers:
        List<WmsLayerInfo> layers = result.getLayers(); // These are the WMS la

        // Let's add the first layer to the map:
        WmsLayerInfo layerInfo = layers.get(0);

        WmsLayerConfiguration wmsConfig = new WmsLayerConfiguration();
        wmsConfig.setBaseUrl(urlBox.getValue());
        wmsConfig.setLayers(layerInfo.getName());

        Coordinate origin = new Coordinate(layerInfo.getBoundingBox().getX(),
                layerInfo.getBoundingBox().getY());
        WmsTileConfiguration tileConfig = new WmsTileConfiguration(256, 256, or

        // Create a WMS layer and add it to the map:
        WmsLayer layer = injector.getWmsLayerFactory().createWmsLayer(layerInfo
                wmsConfig, tileConfig);
        mapPresenter.getLayersModel().addLayer(layer);

        // Make sure we enable animation for our newly add layer - it's just ni
        mapPresenter.getMapRenderer().setNrAnimatedLayers(mapPresenter.getLayer
                getLayerCount());
    }
});
```

4. How to make cross-domain WMS requests

Often your WMS server will have a different domain than the application you're creating. In such cases the browser will not allow cross-domain requests, and as a result you will most likely use a proxy servlet that takes care of this problem. The trick is to have the `WmsService` make use of your proxy servlet.

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 are URLs point to images, 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;
    }
});
```