Geomajas pureGWT face

Geomajas Developers and Geosparc

Geomajas pureGWT face

by Geomajas Developers and Geosparc

1.0.0-M4

Table of Contents

List of Tables

Chapter 1. Introduction

The PureGWT face is an AJAX client for the Geomajas Server, based upon the Google Web Toolkit. The goal of this client is to be as light as possible, so it can easily run on mobile devices.

Contrary to the GWT face, this face provides only one widget: the map. It does not come with a widget library either. It focuses entirely on providing a clean GIS API. In a sense this face is somewhat similar to the Google Maps API, except that it still relies on the Geomajas server.

1. Thin client

Since the Geomajas Server is perfectly capable of safeguarding your domain model or executing complex tasks, this client can focus more on the map presentation than business logic. This already makes the PureGWT face a real thin client. Take also the use of the Google Web Toolkit (GWT) into account and this client becomes even thinner.

The Google Web Toolkit is a Java to JavaScript compiler, that compiles only the code that you're actually using. So instead of having to download entire JavaScript libraries of which you're only using a small portion, a GWT application contains only code that is actually used. Secondly, GWT provides a lot of tools for further optimizing the way your applications are constructed. It provides automatic obfuscation, resource bundling (combining resources so that less requests are needed to loads them all), etc.

As a result of all this optimization, the memory consumption within the browser is also kept at a minimum, while performance is kept at maximum.

2. Mobile vs desktop

Although aimed towards mobile devices, the PureGWT face is perfectly suitable for both desktop and mobile environments. As mentioned above, the initial footprint is as small as possible (easily a factor 10 smaller than other frameworks). On top of this, the PureGWT face is also designed to keep client-server bandwidth to a minimum, by providing caching mechanisms, lazy loading, etc.

Chapter 2. PureGWT face architecture

This part of the documentation describes the main architecture and API of the PureGWT face for Geomajas. For actual code examples or more detail on how to use them, visit the "HowTo" guide.

1. The central Map API

This section describes the interfaces of the most crucial object definitions within the API: those that make up the central map model. Central in all this, is the `MapPresenter`. This is the interface that represents a map in Geomajas. This `MapPresenter` in turn provides 2 very important interfaces through getters: the `ViewPort` and the `LayersModel`.

Quickly speaking, the `ViewPort` provides methods and events for changes in the viewing area of the map. In other words, it determines the map's navigation. The `LayersModel` on the other hand is what it's name implies: a model that governs a set of layers.

1.1. MapPresenter

The MapPresenter represents the central map interface and as such it determines the map's functionalities. It provides support for many of the topics that are discussed in the following sections, such as MapControllers or an EventBus.

1.2. ViewPort

The `ViewPort` can be seen of as the navigator behind the map. It manages the map's navigation (by making it zoom or pan) and sends the required events. On top of that, the `ViewPort` also has a second responsibility in providing transformations between "WorldSpace" and "ViewSpace" (visit the WorldSpace vs ScreenSpace section for more information).

1.3. LayersModel

Also part of the central map model, is a separate interface for managing all the layers of the map. As is typically the case in GIS, people work not just with one type of data, but with many different types that are all represented by "layers" in a map. These layers always have a certain order in which they are drawn, as they lie on top of each other.

This model provides methods to add and remove layers, to change layer order, to retrieve layer information etc. Most changes in the layer model will trigger some event that can always be captured.

1.4. Layers

As many different types of layers exist all with their own specific set of functionalities, we have decided to reflect this diversity in the layer interface, by splitting it up in multiple 'functional' interfaces. There is still a main Layer interface, which must always be implemented, but layer implementation can decide for themselves which of the 'functional' interfaces they support and which they don't.

2. Events

Associated with the functionalities in the central map interfaces, are several events that signal changes in the model to all registered handlers. Note that the term `Handler` is used, not listener, as we try to follow the GWT naming conventions. As of GWT 2.x, the use of a central `EventBus` has been promoted to work together with an MVP approach. The PureGWT face has incorporated this train of thought, and provides a map-centric `EventBus`. In other words, every `MapPresenter` governs it's own `EventBus`.

This means that for `Handlers` that are registered on such an `EventBus`, only events that have originated within that map will reach it. Here is an example of how one can attach a Handler to an EventBus:

```
mapPresenter.getEventBus().addHandler(MapResizedEvent.TYPE, new MapResizedHandl

    public void onMapResized(MapResizedEvent event) {
        // The map has resized. Quick, do something meaningful!
    }
});
```

In the example above a `MapResizedHandler` was used that listens to `MapResizedEvents`. From the moment the `MapResizedHandler` has been registered, it will receive all events that indicate the map has been resized.

Time to go over all supported events and explain their purpose. Note that every event in this list is part of the PureGWT API within Geomajas. All event and handler classes can be found in the following package: `org.geomajas.puregwt.client.map.event`.

Map events:

Table 2.1. Map Events

Event	Handler	Description
MapInitializationEvent	MapInitializationHandler	Event that is fired when the map has been initialized. Only after this point will layers be available.
MapResizedEvent	MapResizedHandler	Event that is fired when the map widget has changed size.

ViewPort events:

Table 2.2. ViewPort Events

Event	Handler	Description
ViewPortChangedEvent	ViewPortChangedHandler	Event that is fired when the map both zooms and pans at the same time.
ViewPortDraggedEvent	ViewPortChangedHandler	Event that is fired when the map is being panned.
ViewPortScaledEvent	ViewPortChangedHandler	Event that is fired when the map zooms is or out keeping the same center.
ViewPortTranslatedEvent	ViewPortChangedHandler	Event that is fired when the map is being translated.

Layer events:

Table 2.3. Layer Events

Event	Handler	Description
LayerAddedEvent	MapCompositionHandler	Event that is fired when a new layer has been added to the map.
LayerRemovedEvent	MapCompositionHandler	Event that is fired when a layer has been removed from the map.

Event	Handler	Description
LayerSelectedEvent	LayerSelectionHandler	Event that is fired when a layer is selected. Only one layer can be selected at any time, so these events often go together with layer deselect events.
LayerDeselectedEvent	LayerSelectionHandler	Event that is fired when a layer has been deselected.
LayerHideEvent	LayerVisibilityHandler	Event that is fired when a layer becomes invisible.
LayerShowEvent	LayerVisibilityHandler	Event that is fired when a layer becomes visible.
LayerLabelHideEvent	LayerLabeledHandler	Event that is fired when the labels for a layer have been turned off.
LayerLabelShowEvent	LayerLabeledHandler	Event that is fired when the labels for a layer have been turned on.
LayerOrderChangedEvent	LayerOrderChangedHandler	Event that is fired when the layer order has changed.
LayerStyleChangedEvent	LayerStyleChangedHandler	Event that is fired when a layer has a new style.

Feature events:

Table 2.4. Feature Events

Event	Handler	Description
FeatureSelectedEvent	FeatureSelectionHandler	Event that is fired when a feature has been selected.
FeatureDeselectedEvent	FeatureSelectionHandler	Event that is fired when a selected feature has been deselected again.

3. Graphics & Rendering

This section will handle all rendering related topics, explaining the different render spaces (WorldSpace versus ScreenSpace), and how to make full advantage of them when trying to render objects on the map. Finally, the MapGadget will be introduced, which is an experimental interface for defining functional gadgets at a fixed location on the map.

3.1. WorldSpace vs ScreenSpace

Before trying to render anything on a map, it is crucial you understand the difference between WorldSpace and ScreenSpace. Both represent render spaces wherein the user can render his objects.

- *WorldSpace*: World space describes a rendering space where all objects are expressed in the coordinate reference system of the map they are drawn in. As a result, all objects within world space move about with the view on the map.

  Let's say for example that a rectangle is rendered on a map with CRS lon-lat. The rectangle has origin (118,34) and width and height both equal to 1. Than this rectangle will cover the city of Los Angeles.

- *ScreenSpace*: Screen space describes a rendering space where all objects are expressed in pixels with the origin in the top left corner of the map. Objects rendered in screen will always occupy a fixed position on the map. They are immobile and are not affected by changes in the map view.

Caution

Beware that drawing a great many objects in WorldSpace can slow down an application, as their positions need to be recalculated every time the map navgates.

3.2. VectorObjectContainers & VectorObjects

For vector object rendering, the PureGWT face makes use of the Vaadin GwtGraphics library. This library provides all the necessary methods for standard SVG and VML rendering. The main interfaces to note are the `VectorObjectContainer` and the `VectorObject`.

The VectorObjectContainer is a container object as the name implies and provides methods for storing and managing VectorObjects. These VectorObjects in turn are the individual objects (such as Rectangle, Circle, Path, ...) that can be drawn on the map.

The PureGWT face has extended the VectorObjectContainer to provide a ScreenContainer and a WorldContainer. As you might have guessed, the ScreenContainer allows you to render objects in ScreenSpace while the WorldContainer allows you to render objects in WorldSpace.

In order to acquire a ScreenContainer or WorldContainer, all one has to do is request such a container with the MapPresenter. This can be done by calling one of the following lines:

```
// Getting a WorldContainer:
WorldContainer worldContainer = mapPresenter.getWorldContainer("myWorld");

// Getting a ScreenContainer:
ScreenContainer screenContainer = mapPresenter.getScreenContainer("myScreen");
```

When acquiring such a container, you have to specify a name (see above 'myWorld' or 'myScreen'). If a container with such a name does not yet exist a new container is created and returned.
TODO: How about removing these containers again? Also add architectural images here.

3.3. MapGadgets

The MapGadget is a standalone functional gadget that can be placed at a fixed position on the map. Examples are the navigation buttons and the scalebar. MapGadgets are notified of map navigation events, because they often require updating on those events.

The position of a MapGadget is maintained by the face, but you have to make sure that the container has a "position: absolute" style.

4. Geometry manipulation

The spatial package (`org.geomajas.puregwt.client.spatial`) contains a collection of math and geometry related classes and utilities to provide all the client-side calculations one should need. If really complex calculations need to be performed, it's best to let the server handle it anyway.

Next to the expected Geometry definitions (Point, LineString, Polygon, ...) this package also provides a few mathematical concepts such as a Vector, Matrix or LineSegment.

4.1. Geometry definitions

The geometry definitions on the client have been written to resemble the JTS (Java Topology Suite) geometry definitions. This definition in turn has been based upon the OGC Simple Feture

Specification. You will find the same classes with each roughly the same methods. The main difference (beside the package name) is that the methods for complex geometry manipulation are left out.

Supported geometries are:

- *Point*: a geometry representation of a single coordinate.
- *MultiPoint*: a geometry containing multiple Point geometries.
- *LineString*: a list of connected coordinates. Sometimes also called a polyline.
- *LinearRing*: an extension of the LineString geometry that expects the last coordinate to be equal to the first coordinate. In other words, a LinearRing is a closed LineString.
- *MultiLineString*: a geometry containing multiple LineString geometries.
- *Polygon*: a Polygon is a two-part geometry, consisting of an exterior LinearRing and a list of interior LinearRings. The exterior LinearRing, also called the shell, is the outer hull of the geometry, while the interior rings can be seen as holes in the exterior ring's surface area.
- *MultiPolygon*: a geometry containing multiple Polygon geometries.

4.2. Spatial services

5. User Interaction on the Map

This section will handle the basics of interacting with the map, by listening and responding to the mouse events that are generated from the map. Instead of dealing with those mouse events directly, 2 interfaces have been created to shield you from having to attach the correct handlers to the correct DOM elements in the HTML tree.

Note

If you want fine tuned control and attach custom mouse event handlers to custom objects on the map, have a look at the Graphics & Rendering section.

This section here handles the interface that help shield you from such fine tuned but cumbersome mouse event handling.

The next 2 sections will cover the following 2 basic interfaces: the MapController and the MapListener.

5.1. MapController

The MapController is an interface which listens to all mouse events that comes from the map (mouse up, mouse down, mouse move, ...) and has the possibility to react directly onto receiving such events. The MapController basically has all the freedom in the world to manipulate and manage the state of the map as events are being received.

For example, it could prevent the default mouse event behaviour or even prevent event bubbling. But because the MapController has such freedom only one can be active at a time.

An example of a MapController is the NavigationController which allows the user to navigate all around the map.

5.2. MapListener

The other way of interacting with the map, is through MapListeners. A MapListener resembles a MapController quite a lot, except that it does not receive the real mouse events, but imitations of it. It therefore can not interfere with the normal browser event flow, but takes on a more passive form.

As a result, multiple MapListeners can be active on a map at any one time. The MapListener is would therefore be the perfect tool for reporting or reacting without interfering, while a MapController is all about the interference.

6. Client/Server communication

Chapter 3. Configuration

In order to use the PureGWT face in your web application, there are a few steps you need to take. These include adding the correct libraries to your project, configuring the web.xml to set up the Geomajas Communication Service, and setting up the GWT module configuration.

1. Dependencies

2. web.xml

3. Build steps

Chapter 4. GWT widgets

...

Chapter 5. How-to

1. Adding a map to a classic GWT layout

The MapPresenter class not only acts as the Presenter in the MVP (Model-View-Presenter) model but also gives access to the View by implementing the IsWidget interface. As a widget, it can be added to any of the GWT layout classes with some precautions, however:

- The parent widget should implement ProvidesResize
- The size of the map has to be set explicitly

```
ResizeLayoutPanel panel = new ResizeLayoutPanel();
mapPresenter.setSize(100,100);
panel.setWidget(mapPresenter);
```

If the panel is resizable, a custom resize handler should be registered to set the size whenever a resizing event occurs.

2. Adding a map to a GWT 2.0 layout

A special MapLayout class is provided for adding a map to a GWT 2.0 layout class (class that have Layout in their name and follow a sizing hierarchy). No special resize handling is needed for this case:

```
MapLayoutPanel mapLayout = new MapLayoutPanel();
mapLayout.setPresenter(mapPresenter);
RootLayoutPanel.get().add(layout); // fills the complete browser view
```