MapFish Workshop Documentation

Release 0.1

Éric Lemoine, Cédric Moullet, Claude Philipona

November 02, 2009

CONTENTS

Contents:

CHAPTER

ONE

MODULE 1 - GETTING STARTED

Start by creating a folder named `MapFish` in the `C:\` folder, this will be your working folder for this workshop.

1.1 Getting workshop material

Get the Workshop material by checking out http://www.mapfish.org/svn/mapfish/sandbox/camptocamp/mapfish_workshop with Turtoise SVN. For this open the explorer, go into `C:\MapFish`, right-click in the explorer window, choose `SVN Checkout...`, enter the above URL, check that the *Checkout directory* is `C:\MapFish\mapfish_workshop`, and click `OK`.

Export the `mapfish_workshop` folder to your Apache document root, for example by copying it in Apache's `htdocs` directory. You should be able to load http://localhost/mapfish_workshop/printing in your web browser.

1.2 Installing MapFish

To install MapFish, first make sure you've installed "Python for Windows extensions". If not, you can get it here: http://sourceforge.net/projects/pywin32/

Open a terminal command and follow these steps:

```
C:\>cd C:\MapFish
C:\MapFish>C:\Python25\python.exe mapfish_workshop\software\go-mapfish-framework-1.2.py env
```

This command creates a virtual Python environment named `env` and installs MapFish and its dependencies into it.

Now activate the virtual environment with:

```
C:\MapFish>env\Scripts\activate.bat
```

You command prompt should now look like this:

```
<env> C:\MapFish>
```

To check that MapFish is correctly installed, enter:

```
<env> C:\MapFish>paster create --list-templates
```

and check that the output is:

```
Available templates:
basic_package:   A basic setuptools-enabled package
mapfish:         MapFish application template
mapfish_client:  MapFish client plugin template
paste_deploy:    A web application deployed through paste.deploy
pylons:          Pylons application template
pylons_minimal:  Pylons minimal application template
```

1.3 Installing FireFox extensions

It is recommended that you use FireFox and install Firebug [1]. Firebug is an add-on for Firefox that allows you to debug JavaScript in any web page. Firebug requires Firefox and cannot be used with any other web browser. Installing the JSONView extension is also recommented, it will be used for viewing JSON responses in FireFox.

[1] http://getfirebug.com/

CHAPTER

TWO

MODULE 2 - CREATING APPLICATION

In this module you will learn how to create a MapFish application. You will study the structure of a MapFish application, and the code the MapFish framework generates for you when creating a MapFish application.

2.1 Generating the application

2.1.1 Generating the base

To create a MapFish application use:

```
<env> C:\MapFish> paster create -t mapfish MapFishApp
```

`MapFishApp` is the name of the MapFish application you're creating, you can pick any name of your choice. We'll assume that you choose `MapFishApp` in the rest of the document.

When asked what template engine to use answer `mako`, which is the default. When asked if SQLAlchemy 0.5 configuration is to be included, answer `True`, as your MapFish application will include web services relying on database tables.

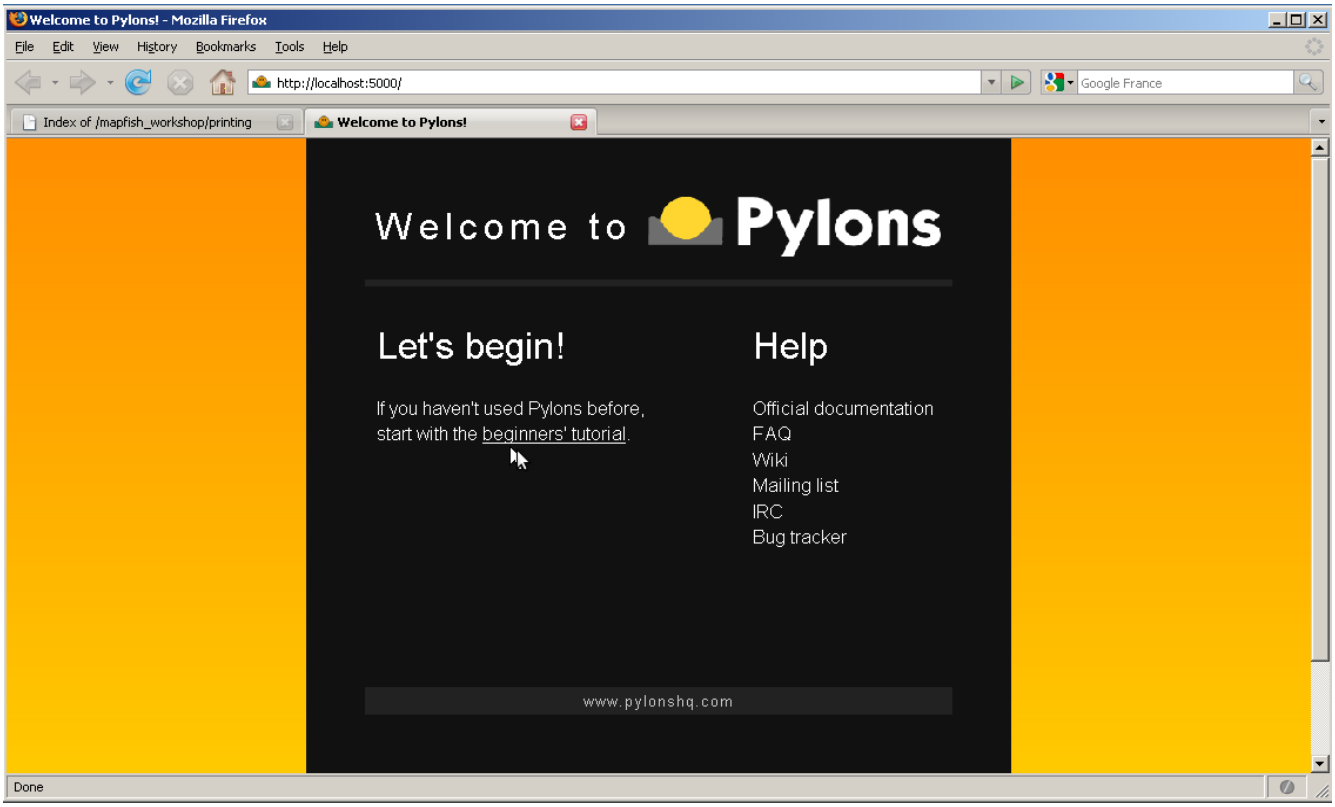
You should now have a folder named `MapFishApp`. This folder contains your application files, at this point mainly Python files.

Now is the time to check that your MapFish application works. For this go into the `MapFishApp` folder and start the application:

```
<env> C:\MapFish>cd MapFishApp
<env> C:\MapFish\MapFishApp>paster serve development.ini
```

This command starts your application in the Paster web server, which is a pure-Python web server, commonly used during development.

Open http://localhost:5000 in your web browser, you should get the default page:



2.1.2 Installing the MapFish JavaScript toolbox

You are now going to install the MapFish JavaScript toolbox in your application. This toolbox includes:

- the Ext, OpenLayers, GeoExt and MapFish Client JavaScript libraries,
- a sample JavaScript application based on those libraries,
- a build profile for minifying the JavaScript code of this sample application,
- a JavaScript testing framework, with a test example

Enter `Ctrl+C` to stop the Paster server and proceed with these commands:

```
<env> C:\MapFish\MapFishApp>cd ..
<env> C:\MapFish> paster create -t mapfish_client MapFishApp
```

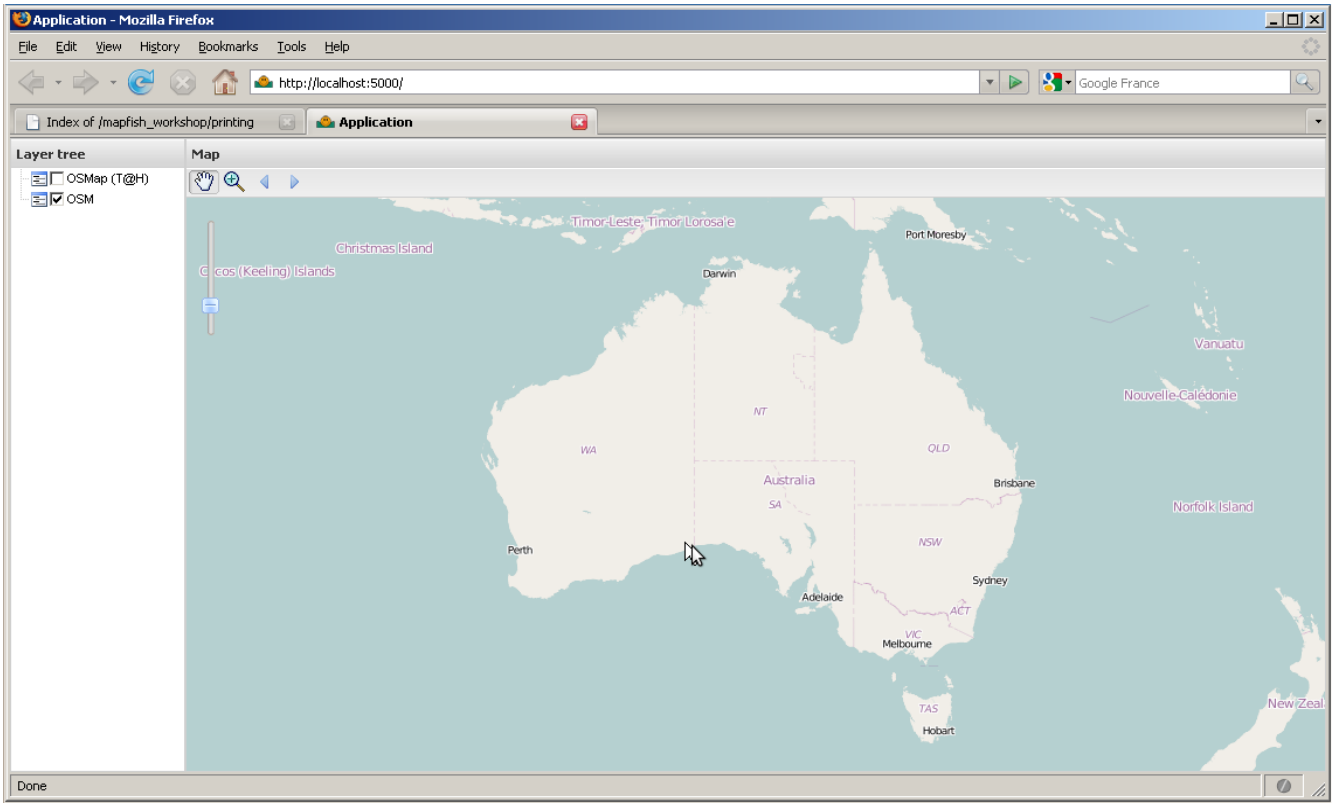
When asked whether to overwrite `index.html` answer `y`. This will overwrite the `index.html` page you saw in the last section by the one provided by the `mapfish_client` template.

Start the application again:

```
<env> C:\MapFish>cd MapFishApp
<env> C:\MapFish\MapFishApp>paster serve --reload development.ini
```

Note: Note the use of the `--reload` switch. This switch makes the Paste server monitor all Python modules used by the `MapFishApp` application and reload itself automatically if any of them is modified or if new modules are created. This is especially useful during development.

Open or reload http://localhost:5000 in your web browser, you should now get the default user interface:



This default user interface is composed of: a map, a toolbar above the map with tools acting on the map, and a layer tree for controlling the visibility of layers. The map itself is composed of two OpenStreetMap base layers (Mapnik and Tiles@Home).

The default user interface is provided to the application developer as an example. The application developer is free to build on it, or delete it to write his own if he wants.

As mentioned at the begining of this section, the JavaScript toolbox installed in the MapFish application comes with a JavaScript testing framework. We clearly see one of the goals of MapFish here: freeing the Application developer from tedious tasks and making him more productive in the development of high-quality, tested code. A test example is provided, to execute it load http://localhost:5000/tests in your browser.

2.2 Studying the application

The following sub-sections give you a quick tour through the folders and files of your MapFish application. Take some time to browse those folders and files, so you get a sense of how the application is structured.

2.2.1 Global structure

The application's main folder, `MapFishApp`, contains:

`development.ini` This is the application's configuration file. This file includes things like the IP address and TCP port the server should listen on, the database connection string, etc.

`layers.ini` This is where the application developer gives information about web services to be generated by the framework. We'll get back to this file in the *Creating Web Services* module later in the document.

jsbuild This folder contains the JavaScript build profile for the default user interface. We'll come back to that in the *Building JavaScript* module later in the document.

setup.cfg and setup.py These files control various aspects of how the MapFish application is packaged when you distribute it.

`mapfishapp`

This is the main application folder, its name depends on the application name given as the argument to the `paster create` command. The main sub-folders of this folder are: `controllers`, `model`, `lib`, `config`, `tests`, `templates`, and `public`.

controllers The `controllers` folder contains the application controllers. The controllers are the components that handle HTTP requests and send HTTP responses. They often interact with the `model` and `templates` code.

model The `model` folder is where the database model is configured. This is basically where tables and relations are defined.

lib The `lib` folder includes Python code shared by different controllers, and third-party code.

config The `config` folder includes Python code generated by the framework and exposed to the application for customization.

tests The `tests` folder is where you can add Python automated tests for the application.

templates The `templates` folder is where view templates are stored. Note that we won't write templates as part of this workshop, as the HTML rendering will mostly be done client side.

`public`

The `public` folder includes the application's static files, i.e. HTML, CSS, JavaScript files, etc. Most of this folder was populated when you installed the JavaScript toolbox with `paster create -t mapfish_client`. The main files and folders inside this folder are: `index.html`, `mfbase`, `app`, and `tests`.

index.html The `index.html` file is the user interface's HTML page. This is where the JavaScript code is loaded.

mfbase The `mfbase` folder contains the MapFish JavaScript toolbox librairies, namely Ext, OpenLayers, GeoExt and MapFish Client.

app The `app` folder contains application-specific files. It `js` sub-folder includes the JavaScript code of the default user interface.

tests The `tests` folder is where the application developer can put its JavaScript tests. The folder includes the JavaScript testing framework, Test.AnotherWay [1], and a test example.

2.2.2 User interface

Let's now review the various files that make up the default user interface (i.e. the web page with the OSM layers).

Edit the `index.html` file and look up these lines:

```
<script type="text/javascript" src="mfbase/ext/adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="mfbase/ext/ext-all-debug.js"></script>
<script type="text/javascript" src="mfbase/openlayers/lib/OpenLayers.js"></script>
<script type="text/javascript" src="mfbase/geoext/lib/GeoExt.js"></script>
<script type="text/javascript" src="mfbase/mapfish/MapFish.js"></script>
```

[1] http://www.openjsan.org/doc/a/ar/artemkhodush/

These <script> tags make the ExtJS, OpenLayers, GeoExt and MapFish JavaScript libraries be loaded when the web page is openned in the browser. These are the versions of the libraries where the JavaScript code is not minified. Again, we'll talk about JavaScript minification in the *Building JavaScript* module.

The lines:

```
<script type="text/javascript" src="app/js/mapfishapp_layout.js"></script>
<script type="text/javascript" src="app/js/mapfishapp_init.js"></script>
```

in the `index.html` take care of loading the application-specific JavaScript code. As you have noticed the JavaScript code of the default user interface is composed of two files. The application developer is free to add more files if needed.

The `mapfishapp_init.js` file represents the entry point.

```
/*
 * @include mapfishapp_layout.js
 */

Ext.namespace("mapfishapp");

(function() {
    // run mapfishapp.layout.init() when the page
    // is ready
    Ext.onReady(function() {
        mapfishapp.layout.init()
    });
})();
```

This file just creates the application namespace, and registers a callback to be run when the entire HTML page and its components are loaded. The callback is registered using the `Ext.onReady` function; we infer from the namespace of the function that the function is provided by the Ext library. Using `Ext.onReady` is typical in Ext-based applications.

The `mapfishapp_layout.js` file is where the page layout is defined. This file contains private functions, i.e. functions that cannot be called from outside the `mapfishapp.layout` module, and the `init` public function, which is the function that was passed to the `Ext.onReady` function.

Let's review what the `mapfish.layout` module's functions do.

`createMap` This function creates the map, which is an instance of `OpenLayers.Map`. See the OpenLayers.Map doc [2].

`createLayers` This function creates the OSM layers and return them.

`createLayerStore` This function creates a `GeoExt.data.LayerStore` object with the `map` and `layers` passed as arguments. A `GeoExt.data.LayerStore` is necessary for creating a map panel. See the GeoExt.data.LayerStore doc [3].

`createTbarItems` This function create and return toolbar items. Here the toolbar items are `GeoExt.Action` objects. We'll go back to `GeoExt.Action` in the *Customize the Application* module.

`init`

This function starts by calling the `createMap`, `createLayers`, and `createLayerStore` functions to actually create the map, layers, and layer store.

[2] http://dev.openlayers.org/apidocs/files/OpenLayers/Map-js.html
[3] http://www.geoext.org/lib/GeoExt/data/LayerStore.html

It then creates an `Ext.Viewport`, which is a graphical component representing the entire browser viewport. The viewport contains other grahical components: its `items`. The viewport here contains two items, a map panel and a layer tree panel. See the Ext.Viewport doc [4].

The map panel, which is a GeoExt object, is configured with the map and the layer store objects. The map panel is also a container, with one item: the zoom slider. The map panel has a top toolbar, whose items are returned by the `createTbarItems` function. See the GeoExt.MapPanel doc [5].

The layer tree is a regular `Ext.tree.TreePanel` with a `GeoExt.tree.LayerContainer` as its root node. See the Ext.tree.TreePanel doc [6], and the GeoExt.tree.LayerContainer doc [7].

[4] http://www.extjs.com/deploy/ext-2.2.1/docs/?class=Ext.Viewport
[5] http://www.geoext.org/lib/GeoExt/widgets/MapPanel.html
[6] http://www.extjs.com/deploy/ext-2.2.1/docs/?class=Ext.tree.TreePanel
[7] http://www.geoext.org/lib/GeoExt/widgets/tree/LayerContainer.html

CHAPTER

THREE

MODULE 3 - CUSTOMIZING THE APPLICATION

In this section you're going to customize the default user interface. More specifically you're going to add layers and tools to the map. These tasks will involve adding JavaScript code in the `mapfish_layout.js` file.

3.1 Adding layers

You're going to add a WMS layer to the map.

Programming task

Edit the `mapfishapp_layout.js` file and add a `OpenLayers.Layer.WMS` object to the array of layers returned by the `createLayers` function. Here's the code for creating the `OpenLayers.Layer.WMS` object:

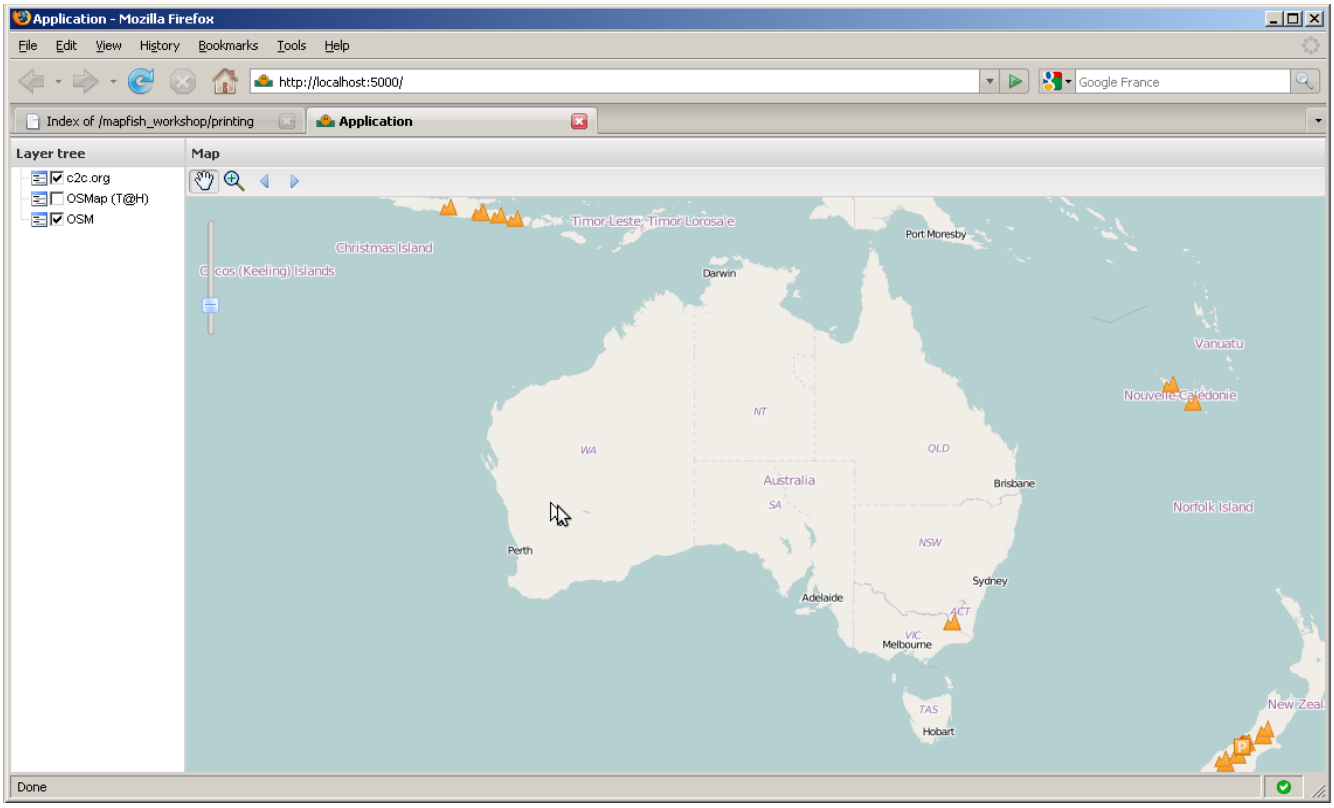
```
new OpenLayers.Layer.WMS(
    "c2c.org",
    "http://www.camptocamp.org/cgi-bin/mapserv_c2corg",
    {
        layers: 'summits,routes,huts,parkings,sites',
        format: 'png',
        transparent: true
    }, {
        singleTile: true
    }
)
```

The `OpenLayers.Layer.WMS` object is given a name, a URL to the WMS, WMS parameters, and options. The `singleTile` option indicates that the layer is to be displayed with a single image as opposed to a grid of image tiles. See the OpenLayers.Layer.WMS doc [1] to know more about the `OpenLayers.Layers.WMS` class.

Note: The WMS service used here is provided by the camptocamp.org [2] non-profit organization.

After reloading the application in the browser you should get this:

[1] http://dev.openlayers.org/apidocs/files/OpenLayers/Layer/WMS-js.html
[2] http://www.camptocamp.org



Please contribute to http://www.camptocamp.org if you want to see more summits, routes, huts, parkings and climbing sites added to your Australia map :-)

The code you added here makes use of the OpenLayers library only, you haven't written any GeoExt and MapFish code at this point.

[Correction here]

Bonus task

Add a Google Maps layer to the map. You can look at http://www.openlayers.org/dev/examples/spherical-mercator.html as an example.

3.2 Adding tools

Here you're going to add a *Zoom* - tool next to the *Zoom* + tool in the map's toolbar.

Programming task

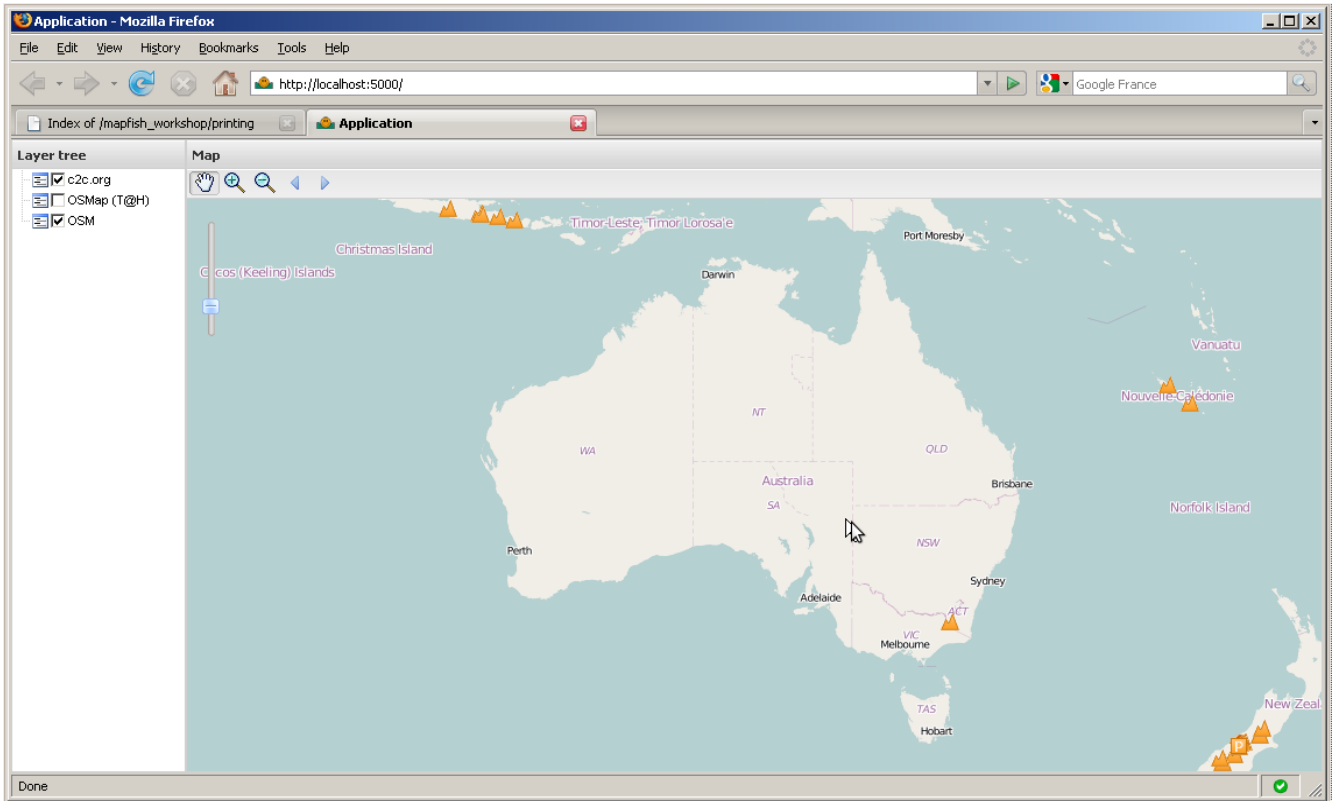
Edit the `mapfishapp_layout.js` file again, look up the `createTbarItems` function, and add a new `GeoExt.Action` object to the `actions` array returned by the function. Here's the code for creating the `GeoExt.Action` object:

```
actions.push(new GeoExt.Action({
    iconCls: "zoomout",
    map: map,
    toggleGroup: "tools",
    allowDepress: false,
    tooltip: "Zoom out",
    control: new OpenLayers.Control.ZoomBox({
```

```
        out: true
    })
}));
```

This code creates a `GeoExt.Action` object configured with an `OpenLayers.Control.ZoomBox` instance. This code looks like that for the *Zoom* - tool, the main difference being the `out` control option set to `false` for *Zoom* + and to `true` for *Zoom* -. See the GeoExt.Action doc [3] and the OpenLayers.Control.ZoomBox doc [4] (no, actually don't look at the later, it's rather empty!).

After reloading the application in the browser you should get this:



In this section you have learned how to use `GeoExt.Action` objects together with `OpenLayers.Control` objects to add map tools in a tool bar.

[Correction there]

Bonus task

Add *zoom to max extent* and *draw polygons* tools to the toolbar. Adding a *draw polygons* tool would require adding an `OpenLayers.Layer.Vector` to the list of layers returned by the `createLayers` function. Taking a look at the GeoExt toolbar.html example [5] may help.

[3] http://www.geoext.org/lib/GeoExt/widgets/Action.html
[4] http://dev.openlayers.org/apidocs/files/OpenLayers/Control/ZoomBox-js.html
[5] http://dev.geoext.org/trunk/geoext/examples/toolbar.html

CHAPTER

FOUR

MODULE 4 - BUILDING JAVASCRIPT

In this module you're going to learn how to use the `jsbuild` tool to minify the JavaScript of your application. Minifying the JavaScript code is important if you care about performance, as it drastically reduces the time to load the JavaScript code.

4.1 Installation

The `jsbuild` tool is included in the `JSTools` Python package. The MapFish framework package depends on `JSTools`, so `JSTools` was installed in the virtual Python environment as part of the framework installation.

You can check that `jsbuild` is properly installed by running this command:

```
(env) C:\MapFish>C:\MapFish\env\Scripts\jsbuild.exe --help
```

It should produce this output:

```
Usage: jsbuild-script.py [options] filename1.cfg [filename2.cfg...]

Options:
  -h, --help            show this help message and exit
  -u, --uncompress      Don't compresses aggregated javascript
  -v, --verbose         print more info
  -o OUTPUT_DIR, --output=OUTPUT_DIR
                        Output directory
  -r RESOURCE_DIR, --resource=RESOURCE_DIR
                        resource base directory (for interpolation)
  -j SINGLE_FILE, --just=SINGLE_FILE
                        Only create file for this section
  -s CONCAT, --single-file-build=CONCAT
                        Create a single file of all of possible output
  -c COMPRESSOR, --compressor=COMPRESSOR
                        Compressor plugin to use in form
                        {specifier}:{'arguments_string'}
```

4.2 Creating Build Profile

To be able to minify your application JavaScript code you must first create a *build profile*. A build profile holds the build configuration: paths to JavaScript folders, etc.

A build profile for the default user interface is provided in `jsbuild/app.cfg`. It looks like this:

```
[MapFish.js]
root =
    ../mapfishapp/public/mfbase/openlayers/lib
    ../mapfishapp/public/mfbase/mapfish
    ../mapfishapp/public/mfbase/geoext/lib
    ../mapfishapp/public/app/js
first =
    OpenLayers/SingleFile.js
    OpenLayers.js
    OpenLayers/Util.js
    OpenLayers/Lang.js
    OpenLayers/Lang/en.js
    OpenLayers/Console.js
    OpenLayers/BaseTypes.js
    OpenLayers/BaseTypes/Class.js
    OpenLayers/BaseTypes/Pixel.js
    OpenLayers/BaseTypes/Bounds.js
    OpenLayers/BaseTypes/LonLat.js
    OpenLayers/BaseTypes/Element.js
    OpenLayers/BaseTypes/Size.js
    Rico/Corner.js
    SingleFile.js
    MapFish.js
    core/Util.js
include =
    mapfishapp_init.js
exclude =
    GeoExt.js
    GeoExt/SingleFile.js
```

[MapFish.js] `MapFish.js` will be the name of the resulting build file.

root The `root` property provides the paths to the folders containing JavaScript code. It is to be noted that, with the above build profile, the JavaScript code for OpenLayers, GeoExt, MapFish Client, and the application is minified and merged in a single build file, `MapFish.js`.

first The `first` property provides the list of JavaScript files that must be first in the resulting build file.

include The `include` property provides the list of files that should be included in the build file. You'll note that only one file is specified here: the entry point app file. The other JavaScript files to be included in the build file are specified using `@include` directives in the JavaScript application files themselves.

exclude The `exclude` property provides the list of files that should not be included in the build file.

4.3 Building

Before actually building the JavaScript code you need to create the folder where the build file will be placed. In the `public` folder create a folder named `build` with two sub-folders, `mapfish` and `openlayers`.

You can now launch the build command:

```
(env) C:\MapFish>cd MapFishApp/jsbuild
(env) C:\MapFish\MapFishApp\jsbuild>C:\MapFish\env\Scripts\jsbuild.exe -o ..\mapfishapp\public\build\
```

After a small while, the output should be:

```
Done:
..\mapfishapp\public\build\mapfish\MapFish.js
```

The last thing that you need to do is copy resource files. Copy the MapFish `img` folder in the `public/build/mapfish` folder, and copy the OpenLayers `img` and `theme` folders in the `public/build/openlayers` folder. Look at the `README.txt` file in the `jsbuild` folder to know where the MapFish `img`, the OpenLayers `img` and the OpenLayers `theme` folders are. Note that you need to do this copy operations only once.

You can now edit `public/index.html` to use the built version of the JavaScript code. For that comment the *debug mode* section and uncomment the *non debug mode* section. Like this:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf8" />
    <meta name="content-language" content="en" />
    <title>Application</title>

    <link rel="stylesheet" type="text/css" href="mfbase/ext/resources/css/ext-all.css"></link>
    <link rel="stylesheet" type="text/css" href="mfbase/ext/resources/css/xtheme-gray.css"></link>
    <link rel="stylesheet" type="text/css" href="mfbase/mapfish/mapfish.css"></link>

    <script type="text/javascript" src="mfbase/ext/adapter/ext/ext-base.js"></script>

    <!-- debug mode (begin) -->
    <!--
    <script type="text/javascript">
        // Because of a bug in Firefox 2 we need to specify the MapFish base path.
        // See https://bugzilla.mozilla.org/show_bug.cgi?id=351282
        var gMfLocation = "mfbase/mapfish/";
    </script>
    <script type="text/javascript" src="mfbase/ext/ext-all-debug.js"></script>
    <script type="text/javascript" src="mfbase/openlayers/lib/OpenLayers.js"></script>
    <script type="text/javascript" src="mfbase/geoext/lib/GeoExt.js"></script>
    <script type="text/javascript" src="mfbase/mapfish/MapFish.js"></script>
    <script type="text/javascript" src="app/js/mapfishapp_layout.js"></script>
    <script type="text/javascript" src="app/js/mapfishapp_init.js"></script>
    -->
    <!-- debug mode (end) -->

    <!-- non debug mode (begin) -->
    <script type="text/javascript" src="mfbase/ext/ext-all.js"></script>
    <script type="text/javascript" src="build/mapfish/MapFish.js"></script>
    <!-- non debug mode (end) -->

</head>
<body>
</body>
<html>
```

You can now reload http://localhost:5000 in your browser.

You should get JavaScript errors in the FireBug console, this is because you added functionality in the previous modules of this workshop without adding proper `@include` directives in the `mapfishapp_layout.js`. You should be able to fix that by adding the missing `@include` directives, for instance:

```
@include OpenLayers/Layer/Google.js
@include OpenLayers/Layer/WMS.js
@include OpenLayers/Request/XMLHttpRequest.js
@include core/Searcher/Map.js
@include core/Protocol/MapFish.js
@include core/Protocol/TriggerEventDecorator.js
@include core/Protocol/MergeFilterDecorator.js
```

CHAPTER

FIVE

MODULE 5 - CREATING WEB SERVICES

In this module you are going to learn how to use the framework to create *MapFish web services* in your application.

MapFish web services are web services for creating, reading, updating and deleting geographic objects (features) through the *MapFish Protocol*.

The MapFish Protocol is a collection of HTTP APIs. It is highly recommended to take some time to go through the description of these APIs [1] before moving on with the rest of this module.

5.1 Installing data

A MapFish web service relies on a spatial data source.

Note: Only PostgreSQL/PostGIS tables are currently supported by MapFish. There's currently work being done to support SQLite/Spatialite and MySQL. Stay tuned.

Before creating the web service we need to create a PostGIS table with some data into it. You're going to create a PostGIS table from a Shapefile of countries.

First, create a PostGIS-enabled database and name it `mapfish_workshop`. For that you can launch an `SQL Shell` and enter:

```
CREATE DATABASE mapfish_workshop TEMPLATE=template_postgis;
```

Then, open the explorer, go into the `C:\MapFish\mapfish_workshop\data` folder and extract the `countries.zip` file. And enter the following commands to import the `countries` Shapefile as a table named `countries` in the `mapfish_workshop` database:

```
C:\MapFish>cd mapfish_workshop\data
C:\MapFish\mapfish_workshop\data>"C:\Program Files\PostgreSQL\8.3\bin\shp2pgsql.exe" -s 4326 -I count
```

You can start pgAdmin and connect to the `mapfish_workshop` database to check that the `countries` table is present and non-empty.

[1] http://www.mapfish.org/doc/1.2/protocol.html

5.2 Connecting to the database

You now need to setup the connection to the `mapfish_workshop` database from `MapFishApp`. This is done in the `development.ini` file.

Edit `development.ini` and replace the line

```
sqlalchemy.url = sqlite:///%(here)s/development.db
```

by this one:

```
sqlalchemy.url = postgres://postgres:postgres@localhost:5432/mapfish_workshop
```

The connection string specifies that the `postgres` driver must be used, the database system listens on `localhost` and on `port` 5432, and the name of the database is `mapfish_workshop`.

5.3 Creating web service

Now that the table is created and the connection to the database is set up, you're ready to create the web service.

Creating a web service is done in three steps:

1. create a layer configuration in the `layers.ini` file, in our case:

   ```
   [countries]
   singular=country
   plural=countries
   table=countries
   epsg=4326
   geomcolumn=the_geom
   ```

   `singular` provides a `singular` name for the layer. `plural` provides a plural name for the layer. Both are used by the code generator when substituting variables. `table` provides the name of the database. `epsg` provides the coordinate system of the table data. `geomcolumn` provides the name of the geometry column.

2. generate the web service code with the `mf-layer` command:

   ```
   <env> C:\MapFish\MapFishApp>paster mf-layer countries
   ```

3. configure a route to the `countries` controller, this is done by adding the following statement after the "CUSTOM ROUTES HERE" comment in the `mapfishapp/config/routing.py` file:

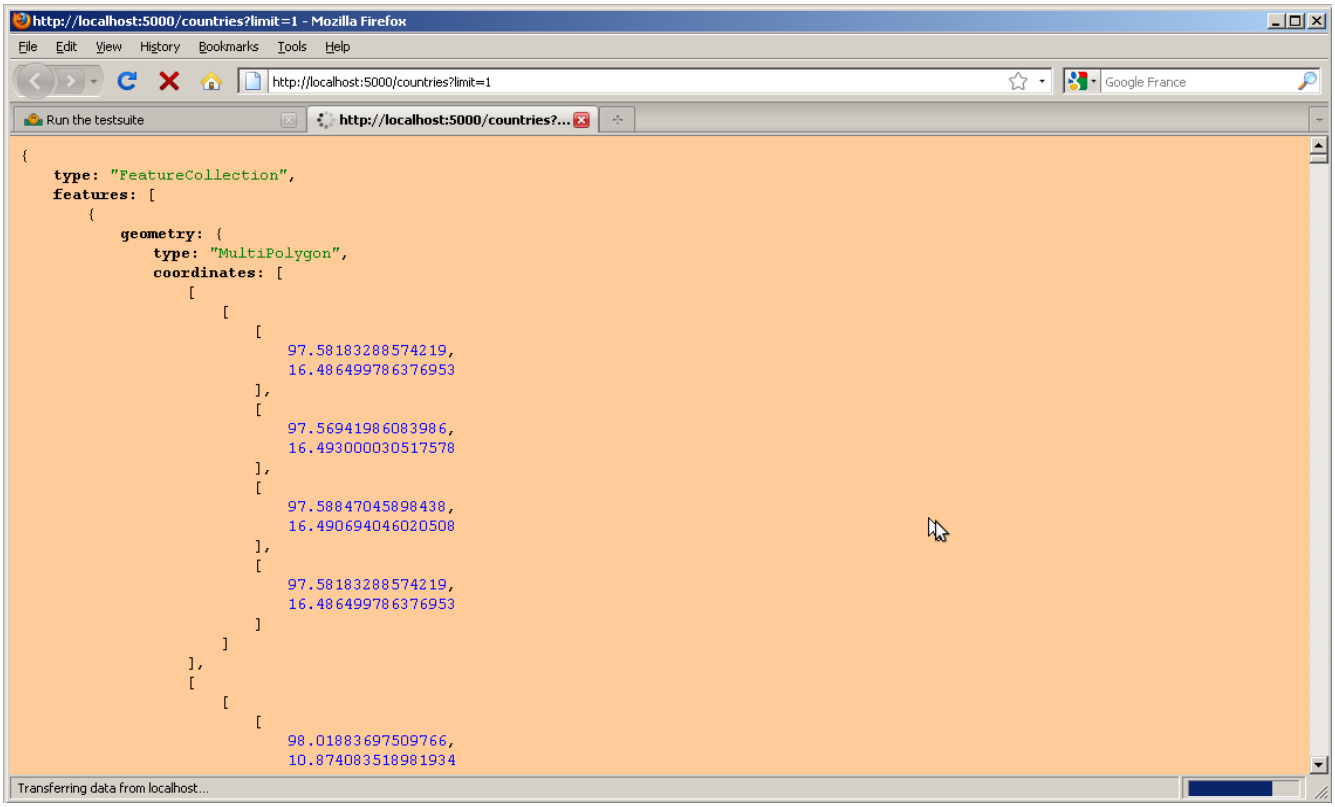
   ```
   map.resource("country", "countries")
   ```

Watch the indentation! 4 spaces are needed here.

If you killed `paster serve` or if you did not add the `--reload` switch, restart `MapFishApp` with:

```
<env> C:\MapFish\MapFishApp>paster serve --reload development.ini
```

You can now open http://localhost:5000/countries?limit=1 in your browser, you should see a GeoJSON representation of the first object in the `countries` table:



Bonus task

Open the MapFish Protocol description again and write the URLs for the following queries:

- get the country whose identifier is 1
- get the country which contains the point (5, 45)
- get the country which contains the point (5, 45) but don't receive its geometry
- get the country which contains the point (5, 45) and receive only the attributes `pays` and `population`

5.4 Studying the web service code

The `paster mf-layer countries` command created three Python files:

`mapfishapp/controllers/countries.py` This file includes the controller code of the `countries` web service. This is the core of the web service.

```
class CountriesController(BaseController):
    readonly = False # if set to True, only GET is supported

    def __init__(self):
        self.protocol = Protocol(Session, Country, self.readonly)

    def index(self, format='json'):
        """GET /: return all features."""
        return self.protocol.index(request, response, format=format)

    def show(self, id, format='json'):
```

```
        """GET /id: Show a specific feature."""
        return self.protocol.show(request, response, id, format=format)

    def create(self):
        """POST /: Create a new feature."""
        return self.protocol.create(request, response)

    def update(self, id):
        """PUT /id: Update an existing feature."""
        return self.protocol.update(request, response, id)

    def delete(self, id):
        """DELETE /id: Delete an existing feature."""
        return self.protocol.delete(request, response, id)
```

The controller has methods for each protocol operation: *get features* (`index`), *get a feature* (`show`), *create features* (`create`), *update a feature* (`update`), and *delete a feature* (`delete`). These methods all rely on `Protocol` object, this protocol object includes all the logic of the MapFish Protocol as defined in the description.

`mapfishapp/model/countries.py` This file includes the model code of the `countries` web service. The model defines the `countries` table object, the `Country` class representing a table record, and the mapping between the two.

```
countries_table = Table(
    'countries', metadata,
    Column('the_geom', Geometry(4326)),
    autoload=True, autoload_with=engine)

class Country(GeometryTableMixIn):
    # for GeometryTableMixIn to do its job the __table__ property
    # must be set here
    __table__ = countries_table

mapper(Country, countries_table)
```

`tests/functional/test_countries.py` This file is where the application developer can put functional tests for the `countries` web service. This is an empty shell.

The code generated by the `paster mf-layer` command belongs to the application developer. The developer is free to modify it, based on his needs.

CHAPTER

SIX

MODULE 6 - ADDING SEARCH FUNCTIONALITY

In this module you're going to add a search functionality to the user interface. This search functionality will rely on the `countries` web service that you created in the previous module.

With this search functionality users will be able to click on the map and get a popup giving information about the clicked country.

The MapFish Client library provides the `mapfish.Searcher.Map` class for enabling that. See the mapfish.Searcher.Map doc [1]. A `mapfish.Searcher.Map` object is an OpenLayers control so it can be wrapped in a `GeoExt.Action` just like the *Zoom +* and *Zoom -* controls that we saw earlier in the workshop.

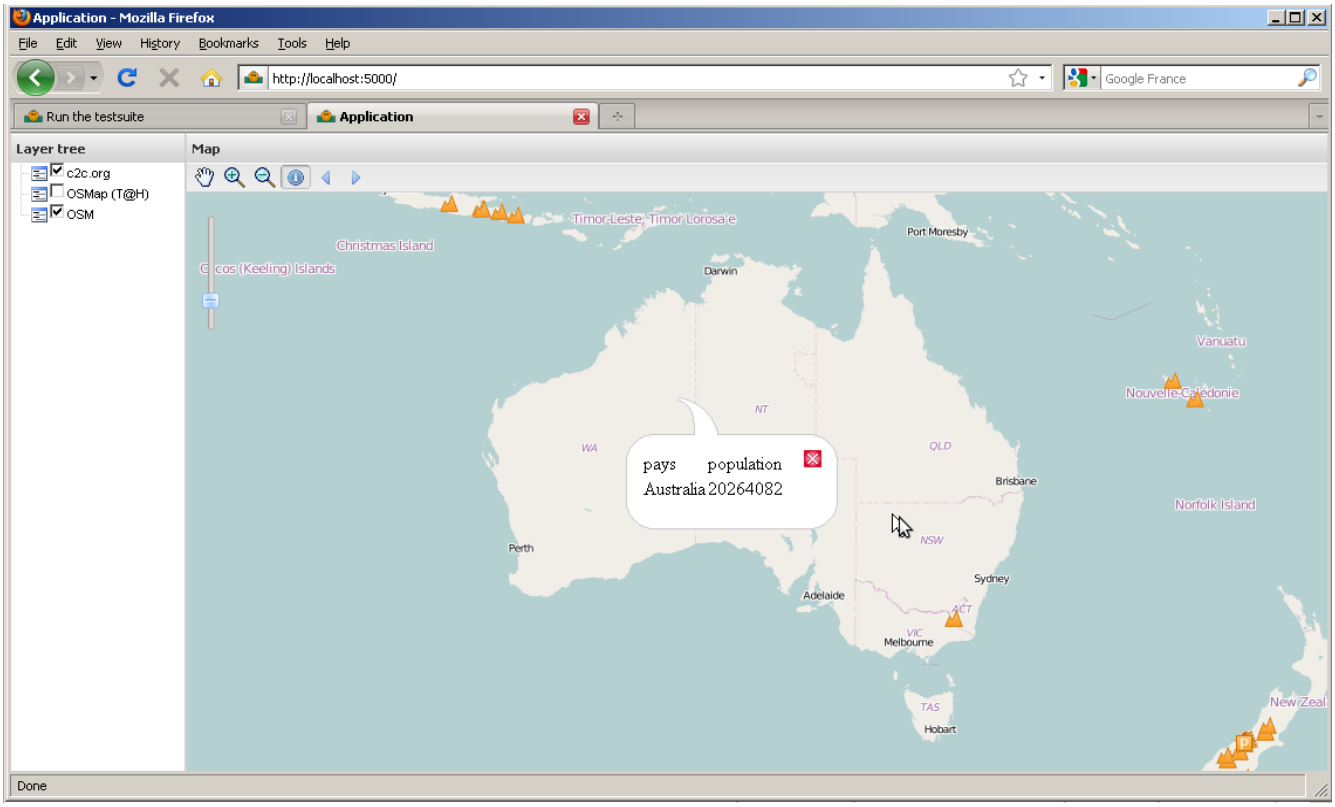
Progamming task

Edit the `mapfishapp_layout.js` file and add a new `GeoExt.Action` to the `actions` array returned by the `addTbarItems` function. This `GeoExt.Action` will be configured with a `mapfish.Searcher.Map` as its control. The `mapfish.Searcher.Map` object must be configured so it

- relies on the `countries` MapFish web service (`url` option)
- triggers search requests on clicks (`mode` option)
- displays a popup with the country name and the population (`displayDefaultPopup` option)

When clicking on a country you should get a popup looking like this:

[1] http://www.mapfish.org/apidoc/trunk/files/mapfish/core/Searcher/Map-js.html



[Correction here]

Note that the AJAX responses may show that the database complains about EPSG:900913 code not being known. In that case, you may run the following INSERT statement in your database:

```
INSERT INTO spatial_ref_sys (srid,auth_name,auth_srid,srtext,proj4text) VALUES
(900913,'EPSG',900913,'PROJCS["Google Mercator", GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
```

Bonus tasks

1. Change the `mapfish.Searcher.Map` configuration so it sends requests as the user pauses on the map.

2. Make the search results be displayed in a `GeoExt.Popup`. For that you'll need to create a `searchcomplete` listener on the searcher and have this listener create the `GeoExt.Popup`. Look at the GeoExt.Popup documentation [2] to know how to use it.

[2] http://geoext.org/lib/GeoExt/widgets/Popup.html

CHAPTER

SEVEN

MODULE 7 - CUSTOMIZING THE WEB SERVICE

This module will demonstrate two examples of web service customization. The first example is based on MapFish-provides APIs. The second example involves knowledge in the powerful SQLAlchemy database toolkit.

7.1 Simple customization

You're going to customize the `countries` web service so it includes only countries of the Oceana continent in its GeoJSON responses.

Programming task

This is done by adding code to the `index` action (function) of the `CountriesController`. The code to be added involves creating a `Comparison` filter and combining it with the default MapFish filter:

```
def index(self, format='json'):
    """GET /: return all features."""
    default_filter = create_default_filter(request, Country)
    compare_filter = comparison.Comparison(
        comparison.Comparison.EQUAL_TO,
        Country.continent,
        value="Oceana"
    )
    filter = logical.Logical(logical.Logical.AND, [default_filter, compare_filter])
    return self.protocol.index(request, response, format=format, filter=filter)
```

MapFish provides several filter classes that the application developer can use to customize his web services. See the filters reference API [1].

[Correction here (controller/countries.py)]

7.2 Advanced customization

You're now going to customize the `countries` web sercice so it sends simplified geometries in its GeoJSON responses.

Programming task

[1] http://www.mapfish.org/doc/1.2/reference/filters.html

This is done by modifying the `countries` model so that SQL queries of type `SELECT simplify(the_geom, 2) ...` are executed by the database. This modification requires knowledge in the SQLAlchemy ORM. Here it is:

```
countries_table = Table(
    'countries', metadata,
    Column('the_geom', Geometry(4326)),
    autoload=True, autoload_with=engine)

class Country(GeometryTableMixIn):
    # for GeometryTableMixIn to do its job the __table__ property
    # must be set here
    __table__ = countries_table

    def toFeature(self):
        # overload toFeature to replace the geometry value with the
        # simplified geometry value
        self.the_geom = wkb.loads(self.the_geom_simple.decode("hex"))
        return GeometryTableMixIn.toFeature(self)

mapper(Country, countries_table, properties={
    "the_geom_simple": column_property(
        func.simplify(countries_table.c.the_geom, 2).label("the_geom")
    )
})
```

[Correction here (model/countries.py)]

Bonus task

Modify the configuration of the `mapfish.Searcher.Map` object so that, in addition to displaying the popup, it also draws the geometry in a vector layer of the map. This is done by listening to `searchcomplete` events and adding the received feature to a vector layer.

CHAPTER

EIGHT

WARRANTY DISCLAIMER AND LICENSE

Camptocamp and authors provide these documents “AS IS,” without warranty of any kind either expressed or implied.

Documents under Creative Common License Attribution-Share Alike 2.5 Generic.

Authors: Éric Lemoine, François Van Der Biest