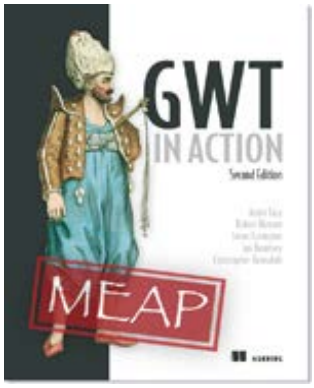
MANNING PUBLICATIONS



GWT in Action, Second Edition
By Adam Tacy, Robert Hanson, Jason Essington, Ian Bambury, and Christopher Ramsdale

This article from chapter 2 of GWT in Action, Second Edition explains how to use the Google Plugin for Eclipse to create a Hello World project, add a module, an EntryPoint and web page.

Creating the Hello World Application with the GPE

The easiest way to create a new GWT application is to use an automated tool to create the structure of a simple application, and then build up from that. Our recommendation is to use the Google Plugin for Eclipse. We'll now walk together through building our Hello World application using that particular tool.

Getting familiar with the toolbar

Figure 1 shows the Eclipse toolbar when you have the Google Plugin for Eclipse installed. (You may have additional buttons if you installed other items, such as GWT Designer.)

Figure 1 The Eclipse toolbar, showing the buttons relating to the Google Plugin for Eclipse (left), and Eclipse Run/Debug functions (right)

Just to avoid some potential confusion over what tool we are using where, we will use the:

- GPE to create a new project and add necessary aspects
- Eclipse Run/Debug buttons to launch development mode

The first thing to do is create the web application structure through the GPE wizards.

Creating a web application

To get started with creating our new application, you should click on the arrow next to the Google Plugin for Eclipse button shown in figure 1. That will reveal the drop down shown in figure 2.

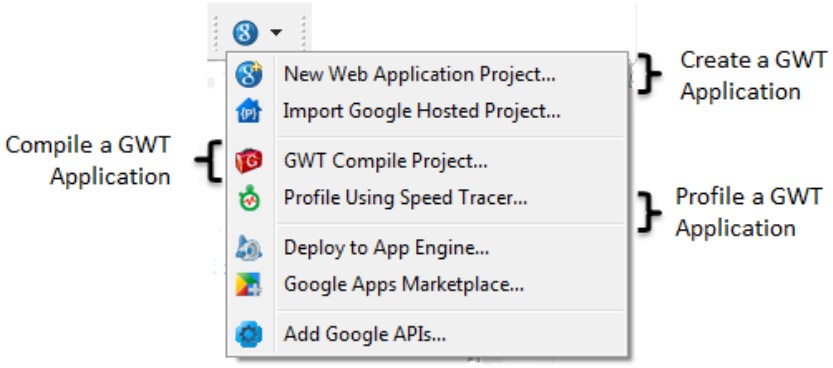


Figure 2 The Eclipse toolbar dropdown, showing the tools available within the plugin

The dropdown menu contains a number of tools, some directly related to GWT, some indirectly. In figure 2, we highlight three tools that are directly related to GWT—the New Web Application Project, GWT Compile Project, and Profile Using Speed Tracer wizards.

To create a new GWT application, just click on the **New Web Application** menu which will bring up the **New Web Application Project** wizard. We'll need to complete three pages of this wizard to create our application. The completed first page of the wizard can be seen in figure 3.

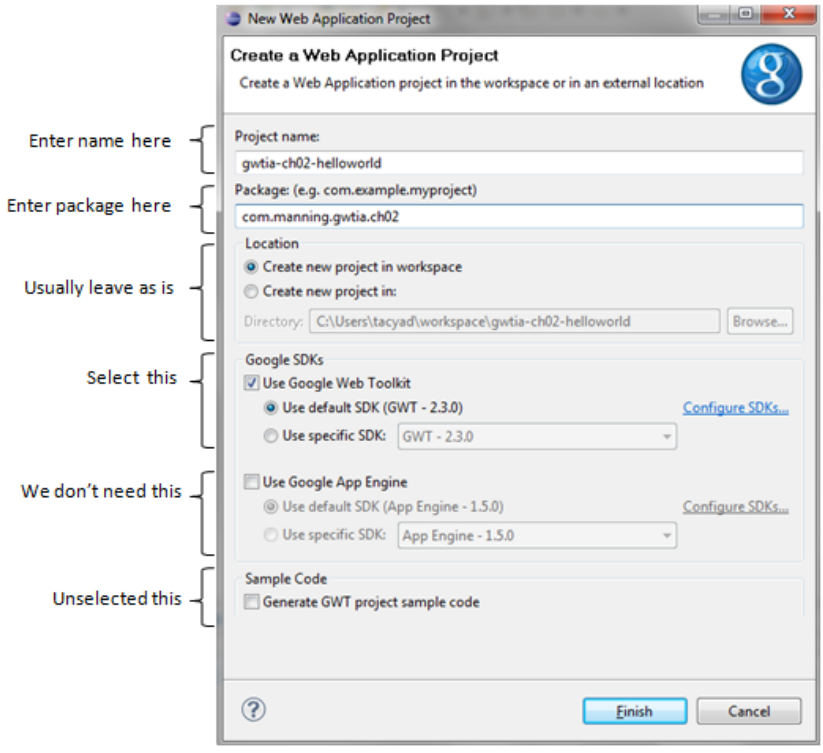


Figure 3 The GPE New Web Application Wizard: fill in the project and package name and unselect the Use of App Engine and Generate sample code.

All you really need to do is simply enter the project name, `gwtia-ch02-helloworld`, and a package name, `com.manning.gwtia.ch02`.

If you wanted, you could select a specific GWT SDK if you have several installed. We'll just leave as is. We don't want to use Google App Engine (for now), so unselect that.

You should also unselect the **Generate GWT project sample code** because we will create the application ourselves. Leaving this selected means the wizard will produce a sample application with client and server side code that we'll end up tying ourselves in knots trying to delete bits to get a basis to move forwards. (The example is good if you want to see a whole prebuilt application, so you may want to do that at some point, but, for now, leave the box unchecked.)

Once you click the **Finish** button, the basic project structure is created, and you should see something like figure 4.

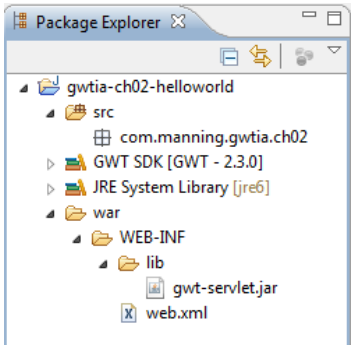


Figure 4 Project structure and contents after creation in Eclipse using the GPE create new project wizard.

If you have more classes and packages in the src folder, then you probably forgot to stop the wizard from creating its sample application. You should delete the project and start again in this case.

The project contains two folders—`src` and `war`. The `src` folder is going to be where our GWT code sits, and the `war` file is where the deployable application sits. Just now, the client side is an empty Java package and the deployable side contains only a servlet, `gwt-servlet.jar`, that will handle any GWT-RPC communication we might implement. (We will have no server side in our example, so you could delete this servlet.)

To take what we have now and make it useful, we need to create a GWT module, a Java `EntryPoint` class, and, in the deployable side, a HTML file that will request the bootstrap code for the application. We can do all that through the GPE plugin, which has helpfully added some items to the area that you create new items in Eclipse.

If you go to the **File** menu and select **New** and the **Other...**, you get a list of wizards. If you expand the Google Web Toolkit folder, you will see figure 5.

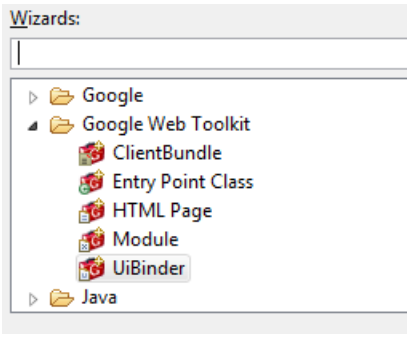


Figure 5 List of other GPE wizards that are available to help build GWT applications in Eclipse

What we need to do is create an entry point class, a HTML page, and a module. There's no strict order that this has to be done in, but we find it easier to start with the module as it is the unit of a GWT application.

Defining a GWT Module

The first thing we need to do is create a Module. Highlight the `com.manning.gwtia.ch02` package and start the Module wizard. You will get a dialog box similar to that shown in figure 6.

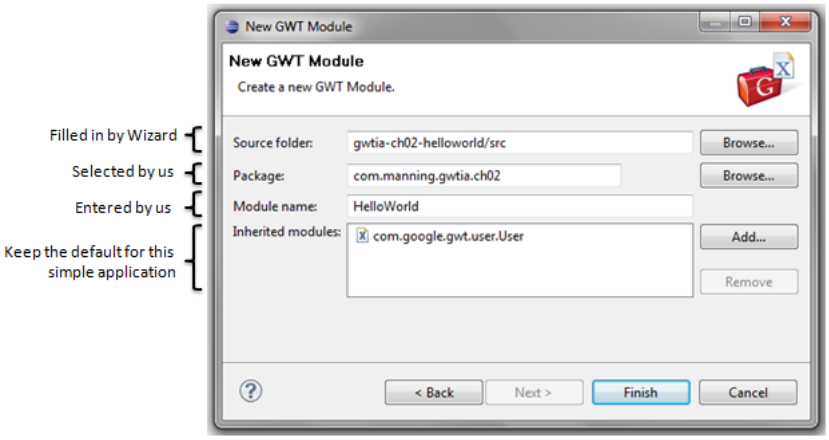


Figure 6 The GPE Module wizard: you should type in the module name and make sure the source folder and package are what you want.

The wizard automatically fills in the Source Folder and package name from the context. If you had highlighted the wrong package name before launching the widget or didn't select one, you can change it now through the browse button. The wizard also indicates that this module will inherit the standard `com.google.gwt.user.User` module.

All you need to do is enter the Module name (`HelloWorld`) and click **Finish**. The wizard creates the Module file in the package we indicated. That module file, for now, just tells GWT to inherit the User module and that code for compilation will be found in the client package.

```
<module>
    <inherits name="com.google.gwt.user.User" />
    <source path="client"/>
</module>
```

Ours is a very simple application with only one module. We will do one little thing to make ours and our user's lives a little easier. Go into the module file and update the module tag to read as follows:

```
<module rename-to="gwtia_ch02_helloworld">
```

This `rename-to` attribute means we have much prettier/manageable URLs to use in our HTML file. Without using the attribute, the bootstrap code is found at:

```
com.manning.gwtia.ch02.HelloWorld/com.manning.gwtia.ch02.HelloWorld.nocache.js
```

When using it, the bootstrap code is found at the more manageable:

```
gwtia_ch02_helloworld/gwtia_ch02_helloworld.nocache.js
```

However, our application is not so useful yet, as it has no entry point.

Adding an EntryPoint

We can create an entry point using the Entry Point Class wizard. It is contextually aware so, if you run it while the module text just created is active in the IDE, then you will see some data filled in; if you run it when the project is selected, other data is filled in. The end point you want to get to is shown in figure 7.

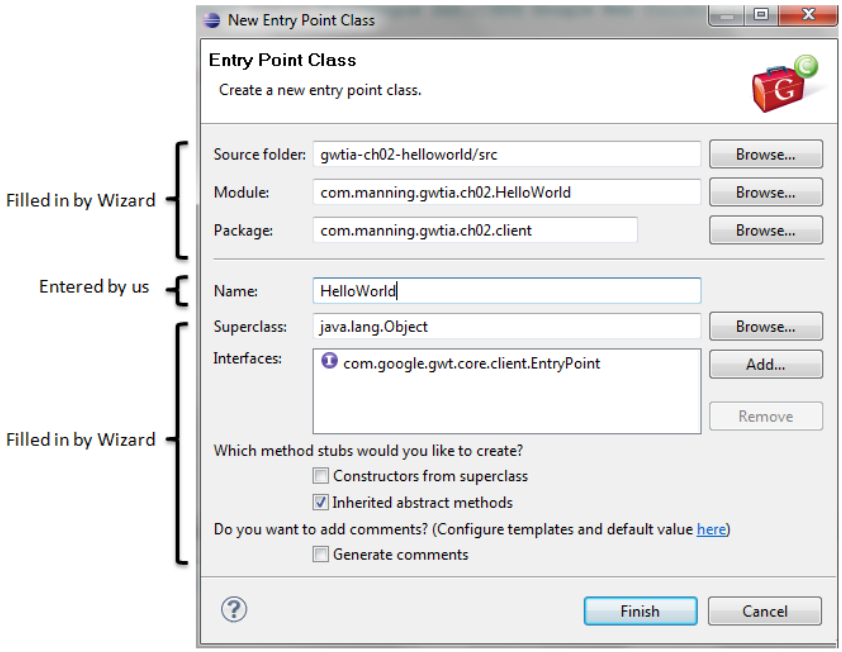


Figure 7 The Entry Point GPE wizard: you should enter the class name that will be the entry point and just check that the rest of the data is what you wanted.

The wizard creates the class `HelloWorld` in the client package and adds an entry point item to the `HelloWorld` module:

```
<entry-point class="com.manning.gwtia.ch02.client.HelloWorld"/>
```

Open the `HelloWorld.java` file up in Eclipse and replace the `onModuleLoad` method with the following:

```
public void onModuleLoad(){
   RootPanel.get().add(new Label("Hello World!"));
}
```

You'll need to import the `RootPanel` GWT panel and `Label` GWT widget in your class to remove the IDE errors. Now our entry point will actually do something—it will display Hello World on the web page when executed.

But that web page is not yet provided.

Providing the web page

The final wizard we use for our application is the HTML Page wizard, as shown in figure 8.

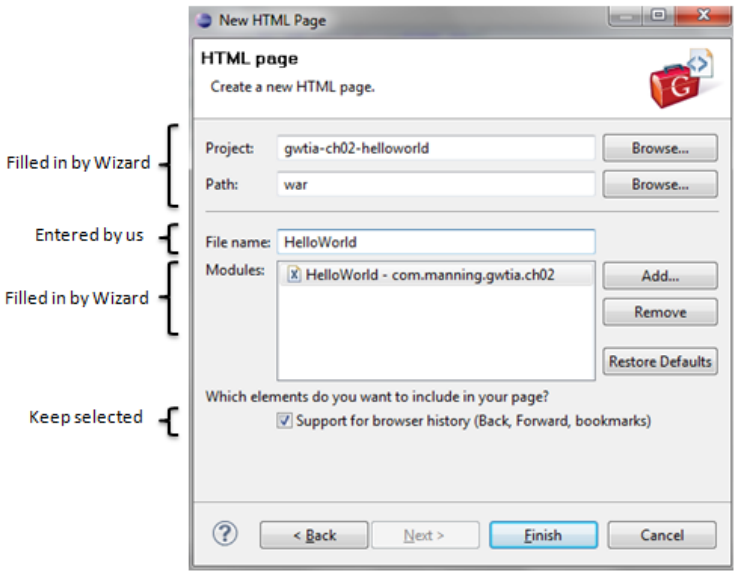


Figure 8 The HTML page GPE wizard: you should enter the file name and usually keep the history support selected.

You need to fill in the filename of the web page. We call it `HelloWorld.html`. You should leave support for browser history checked (which inserts an iframe into the HTML page) as most applications will need this.

Clicking **Finish** means the wizard created the `HelloWorld.html` file in the war directory, which is shown in listing 1.

Listing 1 Simple HTML page for our GWT application

```
<!doctype html>                                                    #1
<html>
   <head>
      <meta http-equiv="content-type" content="text/html; charset=UTF-8">
      <script language="javascript"
         src=" gwtia_ch02_helloworld/gwtia_ch02_helloworld.nocache.js">
      </script>                                                    #2
   </head>
  <body>
    <iframe src="javascript:''" id="__gwt_historyFrame"            #3
            tabIndex='-1'
            style="position:absolute;width:0;height:0;border:0">
   </iframe>
  </body>
</html>
```

#1 Indicates standards mode
#2 Bootstrap application
#3 Support History in IE

The web page doctype indicates the application will run in standards mode—the browser will adhere to W3C standards (#1). It also contains the bootstrap code for the application (#2) and has an iframe so that GWT can support History (back/forward buttons) in IE.

So, now we have created all the structure and code for our first GWT application, and let's see our application in action by running it in Development Mode—the result will be as shown in figure 9.

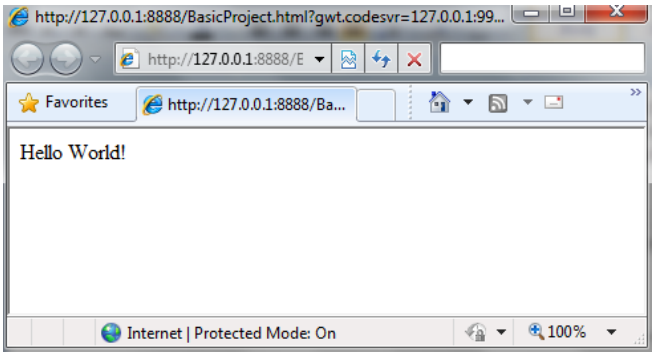


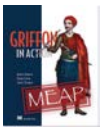
Figure 9 The result of running the Hello World application created from the GWT Designer wizard

Summary

You saw how to create a simple GWT project that displays Hello World on the screen. It's not an earth-shattering application, but it has been useful to get to because we have introduced some of the terminology used in GWT and seen how to use the GPE plugin wizards to create a HelloWorld project.

Here are some other Manning titles you might be interested in:

Grails in Action
Glen Smith and Peter Ledbrook



Griffon in Action
Andres Almiray, Danno Ferrin, and James Shingler

Spring in Action, Third Edition
Craig Walls

Last updated: February 14, 2012