



Google Web Toolkit GWT Java AJAX Programming

A practical guide to Google Web Toolkit for creating AJAX applications with Java

Prabhakar Chaganti



Chapter 2 "Creating a New GWT Application"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter 2 “Creating a New GWT Application”

A synopsis of the book’s content

Information on where to buy this book

About the Author

Prabhakar Chaganti

Prabhakar Chaganti is an enterprise software architect and open-source evangelist working for a cutting-edge software company in the suburbs of Atlanta. His interests include Linux, Ruby, Python, Java, and Virtualization. He recently won the community choice award for the most innovative virtual appliance in the 2006 VMW—this award is the Ultimate Global Virtual Appliance Challenge.

This book has gained immense support from various people. The staff at Packt Publishing were very helpful in providing key assistance to me and ably and enthusiastically led me through the various phases of this project. I would like to thank the technical reviewers for their comments, which have helped make this a much better book. The vibrant community around the GWT mailing list was very helpful whenever I needed clarifications on some of the more arcane corners of GWT.

I would like to thank my wife Nitika for her support and constant encouragement while I was writing this book. She effortlessly played the roles of both mom and dad while I was wrestling with GWT! Thanks and love to my two daughters Anika and Anya for putting up with daddy being stuck to his laptop for long periods of time instead of playing with them.

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

About the Reviewers

Luca Masini

Luca Masini was born in Florence in 1971. He is a senior software engineer and web architect. He has been heavily involved from the first days in the Java world as a consultant for the major Italian banks and firms, developing integration software, and as technical leader in many of the flagship projects. He worked for adoption of Sun's J2EE standard in an environment where COBOL was the leading language, and then he shifted his eyes toward open source, in particular IoC containers, ORM tools, and UI frameworks. As such he adopted early products like Spring, Hibernate, and Struts, giving customers a technological advantage. During last year he fell in love with GWT (of course !!) and he had to master a new project all done with Oracle's ADF Faces and JDeveloper as Visual IDE.

I want to thank my son Jacopo for being my lovely son and my wife for being the best wife a man can dream.

Travis S. Schmidt

Travis S. Schmidt (BS, MBA) is currently employed as an Applications Developer at the University of Iowa Hygienic Laboratory. He has several years of experience in designing and developing web-based clients and recently deployed a system utilizing the Google Web Toolkit.

I would like to thank my loving family: Rebecca, Jacqueline, and Alexander, for their unwavering support.

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

Google Web Toolkit GWT Java AJAX Programming

The client-server architecture has undergone a vast change over a short period of time. Earlier, each application had a different client software, with the software serving as the UI. This software had to be installed individually on every client, and needed to be updated every time we made changes to the application. We moved from that to the web era and deploying applications on the Internet, and then Internet enabled us to use the omnipresent web browser for accessing our applications from anywhere. This was a sea change, but we still had issues of performance and applications not having the same feel or responsiveness as desktop applications. Enter AJAX, and now we can build web pages that can rival a desktop application in responsiveness and nifty looks. AJAX underpins the current trend in developing applications for the Internet known as Web 2.0. In order to build Ajaxified applications you need to know HTML, XML, and JavaScript at the very least. The Google Web Toolkit (GWT) makes it even easier to design an AJAX application using just the Java programming language. It is an open-source Java development framework and its best feature is that we don't have to worry too much about incompatibilities between web browsers and platforms. In GWT, we write the code in Java and then GWT converts it into browser-compliant JavaScript and HTML. This helps a lot, because we can stop worrying about modular programming. It provides a programming framework that is similar to that used by developers building Java applications using one of the GUI toolkits such as Swing, AWT, or SWT. GWT provides all the common user-interface widgets, listeners to react to events happening in the widgets, and ways to combine them into more complex widgets to do things that the GWT team may never have envisioned! Moreover, it makes reusing chunks of program easy. This greatly reduces the number of different technologies that you will need to master. If you know Java, then you can use your favorite IDE (we use Eclipse in this book) to write and debug an AJAX GWT application in Java. Yes, that means you can actually put breakpoints in your code and debug seamlessly from the client side to the server side. You can deploy your applications in any servlet container, create and run unit tests, and essentially develop GWT applications like any Java application. So start reading this book, fire up Eclipse, and enter the wonderful world of AJAX and GWT programming!

In this book, we will start with downloading and installing GWT and walk through the creation, testing, debugging, and deployment of GWT applications. We will be creating a lot of highly interactive and fun user interfaces. We will also customize widgets and use JSNI to integrate GWT with other libraries such as Rico and Moo.fx. We will also learn to create our own custom widgets, and create a calendar and a weather widget. We will explore the I18N and XML support in GWT, create unit tests, and finally learn how to deploy GWT applications to a servlet container such as Tomcat. This book uses a typical task-based pattern, where we first show how to implement a task and then explain its working.

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

What This Book Covers

Chapter 1 introduces GWT, the download and installation of GWT, and running its sample application.

Chapter 2 deals with creation of a new GWT application from scratch, and using the Eclipse IDE with GWT projects, creating a new AJAX Random Quotes application, and running the new application.

Chapter 3 deals with an introduction to and overview of GWT asynchronous services, and creating a prime number service and a geocoder service.

Chapter 4 deals with using GWT to build simple interactive user interfaces. The samples included in this chapter are live search, auto fillable forms, sortable tables, dynamic lists, and a flickr-style editable label.

Chapter 5 introduces some of the more advanced features of GWT to build more complex user interfaces. The samples included in this chapter are pageable tables, editable tree nodes, a simple log spy, sticky notes, and a jigsaw puzzle.

Chapter 6 includes an introduction to JavaScript Native Interface (JSNI) and using it to wrap third-party Javascript libraries like Moo.fx and Rico. It also includes using the gwt-widgets project and its support for the Script.aculo.us effects.

Chapter 7 deals with creating custom GWT widgets. The samples included in this chapter are a calendar widget and a weather widget.

Chapter 8 concerns itself with creating and running unit tests for GWT services and applications.

Chapter 9 sees us using Internationalization (I18N) and client-side XML Support in GWT.

Chapter 10 includes the deployment of GWT applications using both Ant and Eclipse.

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

2

Creating a New GWT Application

In this chapter, we will use the GWT tools to generate a skeleton project structure and files, with and without Eclipse support. We will then create our first AJAX application (a random quote application) by modifying the generated application to add functionality and finally run the application in both hosted and web mode.

The tasks that we will address are:

- Generating a new application
- Generating a new application with Eclipse support
- Creating a random quote AJAX application
- Running the application in hosted mode
- Running the application in web mode

Generating a New Application

We will generate a new GWT application by using one of the GWT scripts. These helper scripts provided by GWT create the skeleton of a GWT project with the basic folder structure and initial project files, so that we can get started in creating our new application as quickly as possible.

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

Time for Action—Using the ApplicationCreator

The GWT distribution contains a command-line script named `applicationCreator` that can be used to create a skeleton GWT project with all the necessary scaffolding. To create a new application, follow the steps given below:

1. Create a new directory named `GWTBook`. We will refer to this directory location as `GWT_EXAMPLES_DIR`. This folder will contain all the projects that will be created while performing the various tasks in this book.
2. Now create a subdirectory and name it `HelloGWT`. This directory will contain the code and the files for the new project that we are going to create in this chapter.
3. Run the `GWT_HOME\applicationCreator` by providing the following parameters in the command prompt:

```
applicationCreator.cmd -out <directory location>\GWTBook\HelloGWT
                        com.packtpub.gwtbook>HelloGWT.client>HelloGWT
```

The `-out` parameter specifies that all the artifacts be generated in the directory named `HelloGWT`. The fully qualified class name provided as the last parameter is used as the name of the class that is generated by the `applicationCreator` script and marked as the `EntryPoint` class for this application (we will cover the `EntryPoint` class in the next section).

The above step will create the folder structure and generate several files in the `GWT_EXAMPLES_DIR\HelloGWT` directory as shown in the following screenshot:

```
C:\gwt-windows-1.3.1>applicationCreator.cmd -out C:\GWTBook\HelloGWT com.packtpub.gwtbook>HelloGWT.client>HelloGWT
Created directory C:\GWTBook\HelloGWT\src
Created directory C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT
Created directory C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\client
Created directory C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\public
Created file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\HelloGWT.gwt.xml
Created file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\public\HelloGWT.html
Created file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\client\HelloGWT.java
Created file C:\GWTBook\HelloGWT\HelloGWT-shell.cmd
Created file C:\GWTBook\HelloGWT\HelloGWT-compile.cmd
```

What Just Happened?

The `applicationCreator` script invokes the `ApplicationCreator` class in `gwt-dev-xxx.jar`, which in turn creates the folder structure and generates the application files. This makes it very easy to get started on a new project as the whole structure for the project is automatically created for you. All you need to do is start filling in the application with your code to provide the desired functionality. A uniform way of creating projects also ensures adherence to a standard directory structure, which makes it easier for you when you are working on different GWT projects.

Here are all the files and folders that were automatically created under the `GWT_EXAMPLES_DIR\HelloGWT` directory when we ran the `applicationCreator` command:

- `src`
- `HelloGWT-compile.cmd`
- `HelloGWT-shell.cmd`

src: This folder contains all the generated source and configuration files for the applications, contained in the familiar Java package structure, with the root package being `com.packtpub.gwtbook.hellojwt`. This package name was deduced by `applicationCreator` from the fully qualified class name that we provided as a parameter to it. The generated files under this directory are:

- `com\packtpub\gwtbook\hellogwt\HelloGWT.gwt.xml`: This is the project module – an XML file that holds the entire configuration needed by a GWT project. The `inherits` tag specifies modules inherited by this module. In this simple case, we are inheriting only the functionality provided by the `User` module, which is built into the GWT. On more complex projects, module inheritance provides a nice way to reuse pieces of functionality. The `EntryPoint` refers to the class that will be instantiated by the GWT framework when the module is loaded. This is the class name provided to the `applicationCreator` command, when we created the project. The following code can be found in this file:

```
<module>
  <!-- Inherit the core Web Toolkit stuff.-->
  <inherits name="com.google.gwt.user.User"/>
  <!-- Specify the app entry point class. -->
  <entry-point class=
    "com.packtpub.gwtbook.hellojwt.client.HelloGWT"/>
</module>
```

- `com\packtpub\gwtbook\hellogwt\client\HelloGWT.java`: This is the entry point for our application. It extends the `EntryPoint` class, and when the `HelloGWT` module is loaded by the GWT framework, this class is instantiated and its `onModuleLoad()` method is automatically called. In this generated class, the `onModuleLoad()` method creates a button and a label, and then adds them to the page. It also adds a click listener for the button. We will be modifying the code in `HelloGWT.java` to create a new application later in this chapter. The current code in this file is as follows:

```
package com.packtpub.gwtbook.hellojwt.client;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
```

```
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;
/** Entry point classes define <code>onModuleLoad()</code>. */
public class HelloGWT implements EntryPoint
{
    /** This is the entry point method. */
    public void onModuleLoad()
    {
        final Button button = new Button("Click me");
        final Label label = new Label();
        button.addClickListener(new ClickListener()
        {
            public void onClick(Widget sender)
            {
                if (label.getText().equals(""))
                    label.setText("Hello World!");
                else
                    label.setText("");
            }
        })
        //Assume that the host HTML has elements defined whose
        //IDs are "slot1", "slot2". In a real app, you probably
        //would not want to hard-code IDs. Instead, you could,
        //for example, search for all elements with a
        //particular CSS class and replace them with widgets.
        RootPanel.get("slot1").add(button);
        RootPanel.get("slot2").add(label);
    }
}
```

- com\packtpub\gwtbook\hellogwt\public\HelloGWT.html: This is a generated HTML page that loads the HelloGWT application and is referred to as the **host page**, as this is the web page that hosts the HelloGWT application. Even though this HTML file is deceptively simple, there are some points that you need to be aware of:
 - Firstly, it contains a meta tag that points to the HelloGWT module directory. This tag is the connection between the HTML page and the HelloGWT application. The following code represents this connection:

```
<meta name='gwt:module'
      content='com.packtpub.gwtbook.hellogwt.HelloGWT'>
```

- Secondly, the `script` tag imports code from the `gwt.js` file. This file contains the code (shown below) required to bootstrap the GWT framework. It uses the configuration in the `HelloGWT.gwt.xml` file, and then dynamically loads the JavaScript created by compiling the `HelloGWT.java` file to present the application. The `gwt.js` file does not exist when we generate the skeleton project. It is generated by the GWT framework when we run the application in hosted mode or when we compile the application.

```
<script language="JavaScript" src="gwt.js"></script>
```

- `HelloGWT-compile.cmd`: This file contains a command script for compiling the application into HTML and JavaScript.
- `HelloGWT-shell.cmd`: This file contains a command script for running the application in the hosted mode.

There is a well-defined relationship between these generated files. The `HelloGWT.html` file is the host page that loads the `gwt.js` file.

There's More!

The `applicationCreator` provides options to control several parameters for a new application. You can see these options by executing it from the following command line:

```
applicationCreator.cmd -help
```

```
C:\gwt-windows-1.3.1>applicationCreator.cmd -help
Google Web Toolkit 1.3.1
ApplicationCreator [-eclipse projectName] [-out dir] [-overwrite] [-ignore] className
where
-eclipse    Creates a debug launch config for the named eclipse project
-out        The directory to write output files into (defaults to current)
-overwrite  Overwrite any existing files
-ignore     Ignore any existing files; do not overwrite
and
className  The fully-qualified name of the application class to create
```

`className` is the only required parameter for the `applicationCreator`. All the other parameters are optional. Here are some different ways to run `applicationCreator`:

- Create a new application without the Eclipse debug support:

```
applicationCreator.cmd -out C:\GWTBook\Test1
                        com.packtpub.gwtbook.Test1.client.Test1
```

- Create a new application with the Eclipse debug support:

```
applicationCreator.cmd -eclipse -out C:\GWTBook\Test1
                        com.packtpub.gwtbook.Test1.client.Test1
```
- Create a new application with the Eclipse debug support that overwrites any previously generated classes with the same name:

```
applicationCreator.cmd -eclipse -overwrite -out C:\GWTBook\Test1
                        com.packtpub.gwtbook.Test1.client.Test1
```

Google recommends the following package naming convention for the source code for a GWT application. This will separate your project code by its functionality.

- `client`: This holds all the client-related application code. This code can only use the Java classes from the `java.util` and `java.lang` packages that are provided by the GWT's JRE Emulation library.
- `public`: This contains all the static web resources that are needed by the application, such as the HTML files, stylesheets, and image files. This directory includes the host page, which is the HTML file that contains the AJAX application (`HelloGWT.html` in the above case).
- `server`: This contains server-side code. These classes can use any Java class and any Java library to provide the functionality.

The modules for the application, such as `HelloGWT.gwt.xml` must be placed in the root package directory as a peer to the `client`, `public`, and `server` packages.

Generating a New Application with Eclipse Support

GWT comes out of the box with support for debugging GWT applications in the Eclipse IDE. This is a tremendously useful and time-saving feature. In this section, we are going to learn how to create new applications with the Eclipse IDE support.

Time for Action—Modifying HelloGWT

The `HelloGWT` application that we have created in the previous task works fine and we can make modifications to it, and run it easily. However, we are not taking advantage of one of GWT's biggest benefits—Eclipse IDE support that enhances the entire development experience. We will now recreate the same `HelloGWT` application, this time as an Eclipse project. It would have been nice if we could take the project that we created in the previous task and add Eclipse support for it. However, GWT does not support this at present. To do this, follow the steps given on the next page:

1. GWT provides a `projectCreator` script that creates Eclipse project files. Run the script with the parameters and you will see a screen as shown below:

```
projectCreator.cmd -out E:\GWTBook\HelloGWT -eclipse HelloGWT
```

```
C:\gwt-windows-1.3.1>projectCreator.cmd -out C:\GWTBook\HelloGWT -eclipse HelloGWT
Created directory C:\GWTBook\HelloGWT\test
Created file C:\GWTBook\HelloGWT\project
Created file C:\GWTBook\HelloGWT\classpath
```

2. Now run the `applicationCreator` again with the parameters given below to create the HelloGWT project as an Eclipse project:

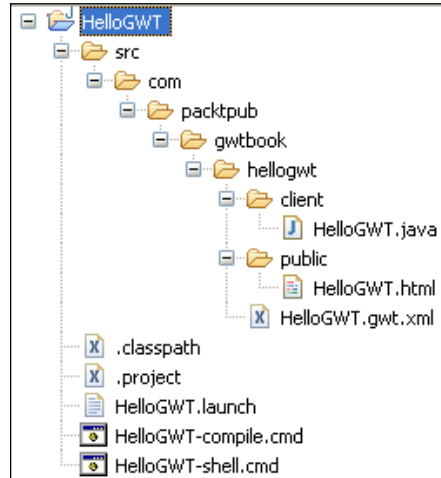
```
applicationCreator.cmd -out E:\GWTBook\HelloGWT -eclipse HelloGWT
-overwrite com.packtpub.gwtbook.hellogwt.client.HelloGWT
```

The `-overwrite` parameter will overwrite the files and folders in the HelloGWT directory. So, if you have made any changes that you would like to keep, please make sure you copy it to a different directory. You will see a screen as shown below:

```
C:\gwt-windows-1.3.1>applicationCreator.cmd -out C:\GWTBook\HelloGWT -eclipse HelloGWT -overwrite com.packtpub.gwtbo
elloGWT.client.HelloGWT
Overwriting existing file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\HelloGWT.gwt.xml
Overwriting existing file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\public\HelloGWT.html
Overwriting existing file C:\GWTBook\HelloGWT\src\com\packtpub\gwtbook\HelloGWT\client\HelloGWT.java
Overwriting existing file C:\GWTBook\HelloGWT\HelloGWT.launch
Overwriting existing file C:\GWTBook\HelloGWT\HelloGWT-shell.cmd
Overwriting existing file C:\GWTBook\HelloGWT\HelloGWT-compile.cmd
```

3. Import the newly created `HelloGWT` project into Eclipse. Navigate to the **Existing projects into Workspace** screen in Eclipse through the **File | Import** menu. Select the `HelloGWT` directory as the root folder, and click on the **Finish** button to import the project into your Eclipse workspace. Now you can edit, debug, and run your application, all from inside the Eclipse IDE!

4. Here are all the folders and files created after we have completed this task:



What Just Happened?

The `projectCreator` script invokes the `ProjectCreator` class in the `gwt-dev-xxx.jar`, which in turn creates the Eclipse project files. These files are then modified by `applicationCreator` to add the name of the project and classpath information for the project.

Here are the Eclipse-specific files created by running the `projectCreator` command:

- `.classpath`: Eclipse file for setting up the project classpath information
- `.project`: Eclipse project file with project name and builder information
- `HelloGWT.launch`: Eclipse configuration for launching the project from the **Run** and **Debug** Eclipse menus

There's More!

Here is a screenshot that displays the various options available for running the `projectCreator` when you run it from a command line with a `-help` option:

```
projectCreator.cmd -help
```

```
C:\gwt-windows-1.3.1>projectCreator.cmd -help
Google Web Toolkit 1.3.1
ProjectCreator [-ant projectName] [-eclipse projectName] [-out dir] [-overwrite] [-ignore]
where
-ant      Generate an Ant buildfile to compile source (.ant.xml will be appended)
-eclipse  Generate an eclipse project
-out      The directory to write output files into (defaults to current)
-overwrite Overwrite any existing files
-ignore   Ignore any existing files; do not overwrite
```

Creating a Random Quote AJAX Application

In this section, we will create our first AJAX application, which will display a random quote on the web page. This example application will familiarize us with the various pieces and modules in a GWT application, and lays the foundation for the rest of the book.

Time for Action—Modifying Auto-Generated Applications

We will create the above-mentioned application by modifying the auto-generated application from the previous task. The skeleton project structure that has been automatically created gives us a head start and demonstrates how quickly we can become productive using the GWT framework and tools.

The random quote is selected from a list of quotes stored on the server. Every second our application will retrieve the random quote provided by the server, and display it on the web page in true AJAX style—without refreshing the page.

1. Create a new Java file named `RandomQuoteService.java` in the `com.packtpub.gwtbook.hellojwt.client` package. Define a `RandomQuoteService` interface with one method to retrieve the quote:

```
public interface RandomQuoteService extends RemoteService
{
    public String getQuote();
}
```

2. Create a new Java file named `RandomQuoteServiceAsync.java` in the `com.packtpub.gwtbook.hellojwt.client` package. Define a `RandomQuoteServiceAsync` interface:

```
public interface RandomQuoteServiceAsync
{
    public void getQuote(AsyncCallback callback);
}
```

3. Create a new Java file named `RandomQuoteServiceImpl.java` in the `com.packtpub.gwtbook.hellojwt.server` package. Define a `RandomQuoteServiceImpl` class that extends `RemoteService` and implements the previously created `RandomQuoteService` interface. Add functionality to this class to return a random quote when the `getQuote()` method is called by a client.

```
public class RandomQuoteServiceImpl extends
    RemoteServiceServlet implements RandomQuoteService
{
    private Random randomizer = new Random();
    private static final long serialVersionUID=
        -1502084255979334403L;
    private static List quotes = new ArrayList();
    static
    {
        quotes.add("No great thing is created suddenly
                    - Epictetus");
        quotes.add("Well done is better than well said
                    - Ben Franklin");
        quotes.add("No wind favors he who has no destined port
                    -Montaigne");
        quotes.add("Sometimes even to live is an act of courage
                    - Seneca");
        quotes.add("Know thyself - Socrates");
    }
    public String getQuote()
        return (String) quotes.get(randomizer.nextInt(4));
    }
}
```

That's all we have to do for implementing functionality on the server. Now, we will modify the client to access the functionality we added to the server.

4. Modify `HelloGWT.java` to remove the existing label and button and add a label for displaying the retrieved quote. Add functionality in the `onModuleLoad()` to create a timer that goes off every second, and invokes the `RandomQuoteService` to retrieve a quote and display it in the label created in the previous step.

```
public void onModuleLoad()
{
    final Label quoteText = new Label();
    //create the service
    final RandomQuoteServiceAsync quoteService =
        (RandomQuoteServiceAsync)GWT.create
            (RandomQuoteService.class);
    //Specify the URL at which our service implementation is
    //running.
    ServiceDefTarget endpoint = (ServiceDefTarget)quoteService;
    endpoint.setServiceEntryPoint("/");
    Timer timer = new Timer()
    {
        public void run()
        {
```

```

//create an async callback to handle the result.
AsyncCallback callback = new AsyncCallback()
{
    public void onSuccess(Object result)
    {
        //display the retrieved quote in the label
        quoteText.setText((String) result);
    }
    public void onFailure(Throwable caught)
    {
        //display the error text if we cant get quote
        quoteText.setText("Failed to get a quote.");
    }
};
//Make the call.
quoteService.getQuote(callback);
}
};
//Schedule the timer to run once every second
timer.scheduleRepeating(1000);
RootPanel.get().add(quoteText);
}

```

We now have the client application accessing the server to retrieve the quote.

5. Modify the `HelloGWT.html` to add a paragraph describing our AJAX application.

```

<p>
This is an AJAX application that retrieves a random quote from
the Random Quote service every second. The data is retrieved
and the quote updated without refreshing the page !
</p>

```

6. Let's make the label look nicer by adding a CSS for the label. Create a new file named `HelloGWT.css` in the `com.packtpub.gwtbook.hello.gwt.public` package and add the following style class declaration to it:

```

quoteLabel
{
    color: white;
    display: block;
    width: 450px;
    padding: 2px 4px;
    text-decoration: none;
    text-align: center;
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
}

```

```
border: 1px solid;  
border-color: black;  
background-color: #704968;  
text-decoration: none;  
}
```

7. Modify the label to use this style in the `HelloGWT.java` file:

```
quoteText.setStyleName("quoteLabel");
```

8. Add a reference to this stylesheet in the `HelloGWT.html` so the page can find the styles defined in the stylesheet.

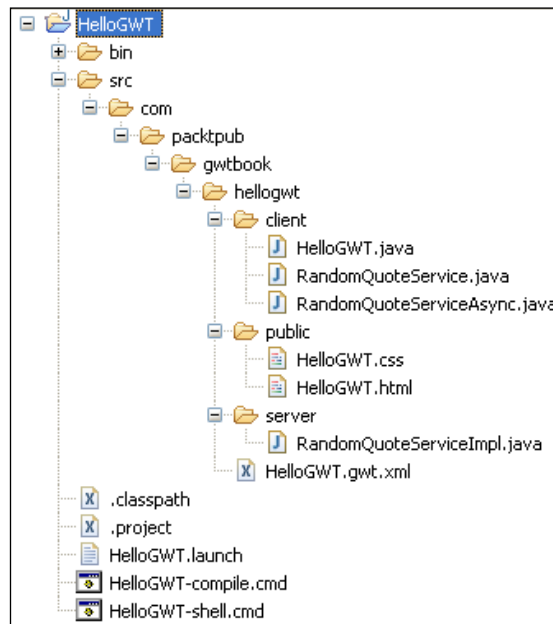
```
<link rel="stylesheet" href="HelloGWT.css">
```

9. The last thing we have to do is register our `RandomQuoteServiceImpl` servlet class in the `HelloGWT` module so that the client can find it. Add the following line to `HelloGWT.gwt.xml`:

```
<servlet path="/" class="com.packtpub.gwtbook.hellojwt.server.  
RandomQuoteServiceImpl"/>
```

This servlet reference will be registered by the GWT framework with the embedded Tomcat servlet container, so that when you run it in the hosted mode, the context path `/` is mapped so that all requests to it are served by the `RandomQuoteServiceImpl` servlet.

Here are the folders and files in the `HelloGWT` project after completing all the above modifications:



Our first AJAX application is now ready and we were able to create it entirely in Java without writing any HTML code!

What Just Happened?

The `RandomQuoteService` interface that we created is the client-side definition of our service. We also defined `RandomQuoteServiceAsync`, which is the client-side definition of the asynchronous version of our service. It provides a callback object that enables the asynchronous communication between the server and the client. The `RandomQuoteServiceImpl` is a servlet that implements this interface and provides the functionality for retrieving a random quote via RPC. We will look into creating services in detail in Chapter 3.

`HelloGWT.java` creates the user interface—just a label in this case—instantiates the `RandomQuote` service, and starts a timer that is scheduled to fire every second. Every time the timer fires, we communicate asynchronously with the `RandomQuoteService` to retrieve a quote, and update the label with the quote. The `RootPanel` is a GWT wrapper for the body of the HTML page. We attach our label to it so it can be displayed.

We modified the look and feel of the label by using a cascading stylesheet, and assigning the name of a style to the label in `HelloGWT.java`. We will learn more about using stylesheets and styles to beautify GWT in Chapter 6.

The user interface in this application is very simple. Hence we added the label straight to the `RootPanel`. However, in almost any non trivial user interface, we will need to position the widgets and lay them out more accurately. We can easily accomplish this by utilizing the various layout and panel classes in the GWT UI framework. We will learn how to use these classes in Chapters 4 and 5.

Running the Application in Hosted Mode

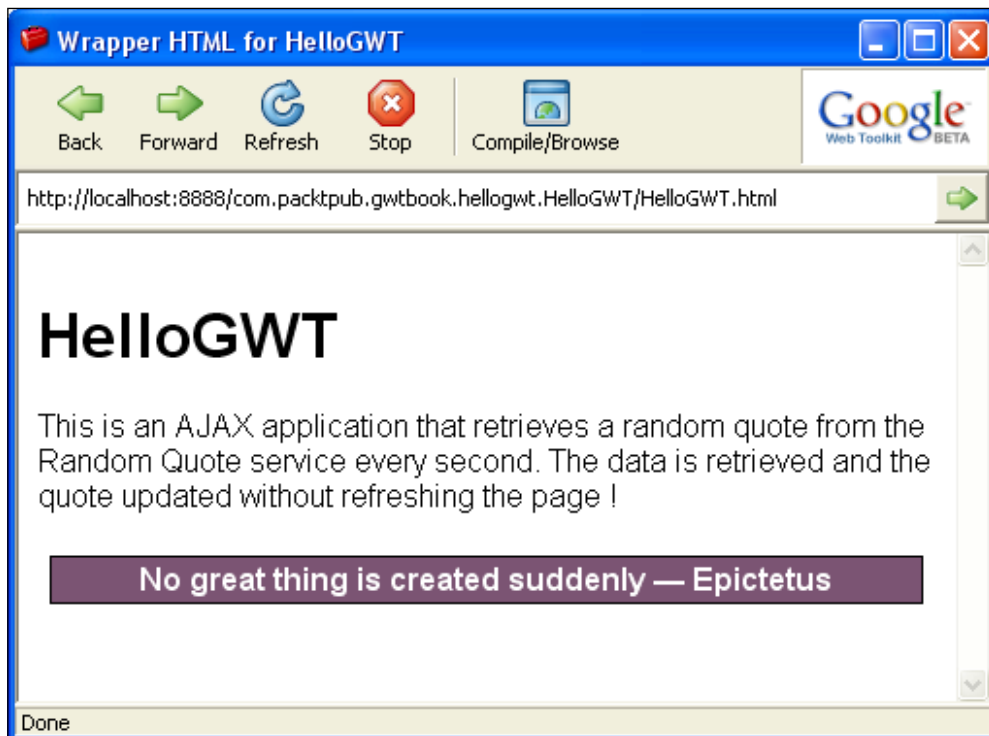
GWT provides a great way to test your application without deploying it but by running the application in a hosted mode. In this section, we will learn how to run the `HelloGWT` application in hosted mode.

Time for Action—Executing the HelloGWT-Shell Script

You can run the `HelloGWT` application in hosted mode by executing the `HelloGWT-shell` script. You can do this in three different ways:

- Executing the command script from the shell:
Open a command prompt and navigate to the `HelloGWT` directory. Run `HelloGWT-shell.cmd` to start the `HelloGWT` application in hosted mode.
- Executing the command script from inside Eclipse:
Double-click on the `HelloGWT-shell.cmd` file in the Eclipse **Package Explorer** or **navigator** view. This will execute the file and will start up the `HelloGWT` application in hosted mode.
- Running the `HelloGWT.launcher` from Eclipse:
In Eclipse, navigate to the **Run** screen by clicking on the **Run | Run** link. Expand the **Java Application** node. Select the `HelloGWT` directory. Click on the **Run** link to launch the `HelloGWT` application in hosted mode.

You will see the following screen if the application runs properly:



What Just Happened?

The command script executes the GWT development shell by providing it with the application class name as a parameter. The Eclipse launcher mimics the command script by creating a launch configuration that executes the GWT development shell from within the Eclipse environment. The launched GWT development shell loads the specified application in an embedded browser window, which displays the application. In hosted mode, the Java code in the project is not compiled into JavaScript. The application code is being run in the Java Virtual Machine as compiled bytecode.

Running the Application in Web Mode

In the previous section, we learned how to run GWT applications in hosted mode without deploying them. That is a great way to test and debug your application. However, when your application is running in a production environment, it will be deployed to a servlet container such as Tomcat. This task explains how to compile the `HelloGWT` application so that it can then be deployed to any servlet container. In GWT terms, this is referred to as running in the web mode.

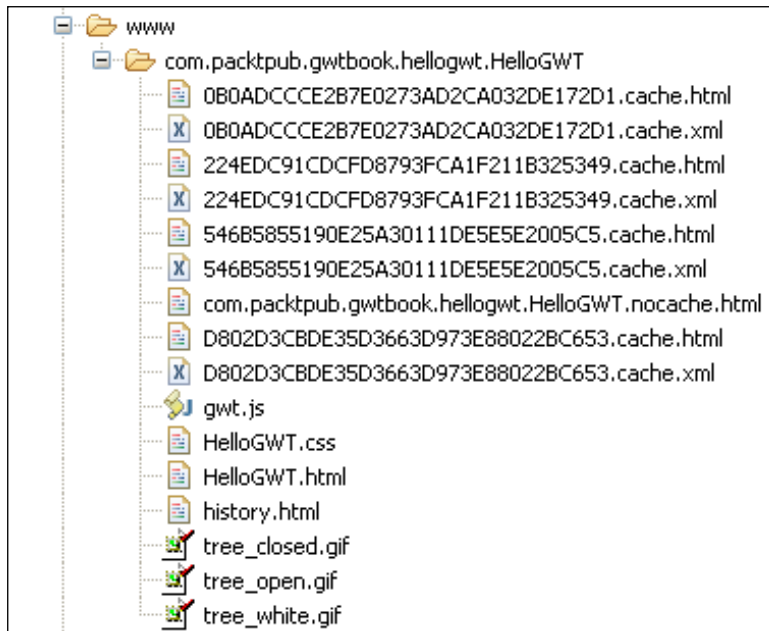
Time for Action—Compile the Application

In order to run the `HelloGWT` application in web mode we need to do the following:

1. Compile the `HelloGWT` application first, by running the `HelloGWT-compile` script.
`HelloGWT-compile.cmd`
2. The above step will create a `www` folder in the `HelloGWT` directory. Navigate to the `www/com.packtpub.gwt>HelloGWT>HelloGWT` directory.

3. Open the `HelloGWT.html` file in your web browser.

Everything needed to run the `HelloGWT` client application is contained in the `www` folder. You can deploy the contents of the folder to any servlet container and serve up the `HelloGWT` application. Here are the contents of the folder after completing the above steps:



What Just Happened?

The `HelloGWT-compile` script invokes the GWT compiler and compiles all the Java source code in the `com.packtpub.gwt.hellojwt.client` package into HTML and JavaScript and copies it to the `www\com.packtpub.gwt.hellojwt.HelloGWT` directory. This directory name is automatically created by GWT, by removing the `client` portion from the fully qualified class name provided to `applicationCreator` previously. This folder contains a ready-to-deploy version of the `HelloGWT` client application. It contains:

- `HelloGWT.html`: The host page that functions as the main HTML page for the `HelloGWT` application.
- `gwt.js`: A generated JavaScript file that contains bootstrap code for loading and initializing the GWT framework.
- `History.html`: An HTML file that provides history management support.

- `xxx-cache.html` and `xxx-cache.xml`: One HTML and XML file per supported browser are generated. These contain the JavaScript code generated by the compilation of the source Java files in the `com.packtpub.gwtbook.helloGWT.client` and `com.packtpub.gwtbook.helloGWT.server` packages. For instance, in this case, on Windows, the compilation produced these files:

```
0B0ADCCCE2B7E0273AD2CA032DE172D1.cache.html
0B0ADCCCE2B7E0273AD2CA032DE172D1.cache.xml
224EDC91CDCFD8793FCA1F211B325349.cache.html
224EDC91CDCFD8793FCA1F211B325349.cache.xml
546B5855190E25A30111DE5E5E2005C5.cache.html
546B5855190E25A30111DE5E5E2005C5.cache.xml
D802D3CBDE35D3663D973E88022BC653.cache.html
D802D3CBDE35D3663D973E88022BC653.cache.xml
```

Each set of HTML and XML files represents one supported browser:

```
0B0ADCCCE2B7E0273AD2CA032DE172D1 - Safari
224EDC91CDCFD8793FCA1F211B325349 - Mozilla or Firefox
546B5855190E25A30111DE5E5E2005C5 - Internet Explorer
D802D3CBDE35D3663D973E88022BC653 - Opera
```

The file names are created by generating **Globally Unique Identifiers (GUIDs)** and using the GUID as part of the name. These file names will be different on different computers, and will also be different every time you do a clean recompile of the application on your computer. There is also a master HTML file generated (`com.packtpub.gwtbook.helloGWT.nocache.html`) that selects the right HTML file from the above files and loads it, depending on the browser that is running the application.

The `www` folder does not contain the code from the `com.packtpub.gwtbook.helloGWT.server` package. This server code needs to be compiled and deployed in a servlet container so that the client application can communicate with the random quote service. We will learn about deploying to external servlet containers in Chapter 10. In normal development mode, we will use the hosted mode for testing, which runs the server code inside the embedded Tomcat servlet container in the GWT development shell. This makes it very easy to run and debug the server code from inside the same Eclipse environment as the client application code. This is another feature of GWT, which makes it an extremely productive environment for developing AJAX applications.

In the web mode, our client Java code has been compiled into JavaScript unlike in the hosted mode. Also, you will notice that the `HelloGWT.gwt.xml` is not in this directory. The configuration details from this module are included in the generated HTML and XML files above.

Thankfully, all this work is automatically done for us by the GWT framework when we run the `HelloGWT-compile` script. We can focus on the functionality provided by our AJAX applications and leave the browser-independent code generation and lower level XMLHttpRequest API to GWT.

We will learn how to deploy GWT applications to web servers and servlet containers in Chapter 10.

There's More!

You can also compile the `HelloGWT` application from the GWT development shell in hosted mode. Run the `HelloGWT-shell` command script to run the application in hosted mode. Click on the **Compile/Browse** button in the GWT development shell window. This will compile the application and launch the application in a separate web-browser window.

All this dynamic JavaScript magic means that when you try to view the source for the application from the web browser, you will always see the HTML from the host page. This can be disconcerting when you are trying to debug problems. But the fantastic Eclipse support in GWT means that you can debug issues from the comfort of a graphical debugger by setting breakpoints and stepping through the entire application one line at a time! We will learn more about debugging of GWT applications in Chapter 8.

Summary

In this chapter we generated a new GWT application using the provided helper scripts like `applicationCreator`. We then generated the Eclipse support files for the project. We also created a new random quote AJAX application. We saw how to run this new application in both the hosted and web modes.

In the next chapter, we are going to learn how to create GWT services that will enable us to provide asynchronous functionality that can be accessed via AJAX from the GWT application web pages.

Where to buy this book

You can buy Google Web Toolkit GWT Java AJAX Programming from the Packt Publishing website: <http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>

Free shipping to the US, UK, Europe, Australia, New Zealand and India.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information:

<http://www.packtpub.com/google-web-toolkit-GWT-Java-AJAX/book>