

Accelerated GWT: Building Enterprise Google Web Toolkit Applications

Copyright © 2008 by Vipul Gupta

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-975-4

ISBN-10 (pbk): 1-59059-975-6

ISBN-13 (electronic): 978-1-4302-0616-3

ISBN-10 (electronic): 1-4302-0616-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Clay Andres

Technical Reviewer: Eric Briley

Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell,

Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann,

Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Senior Project Manager: Tracy Brown Collins

Copy Editor: Kim Wimpsett

Associate Production Director: Kari Brooks-Copony

Production Editor: Ellie Fountain

Compositor: Molly Sharp

Proofreader: Liz Welch

Indexer: Beth Palmer

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Director: Tom Deboliski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit <http://www.apress.com>.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at <http://www.apress.com/info/bulksales>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com>.

PART 1



Getting Started with GWT

The Google Web Toolkit, better known as GWT, is a Java-based open source framework that helps you develop Ajax-based web applications without having to worry about the quirky cross-browser details. Released in 2006 and with millions of downloads so far, GWT is changing the face of Ajax-based web application development with faster turnaround times in development and testing; it also offers application debugging right in your favorite Java-based IDE.

Chapter 1 will help get you started with GWT. You will learn some basic GWT terminology and understand how to download and set up GWT on your computer. The chapter will also explain details about various libraries in the framework along with the different modes in which the application can be run. This chapter will also guide you through the development and operation of some basic GWT applications with the tools provided in the framework as well as without them.

Chapter 2 will go into details about how the GWT framework works, providing you with a complete picture of GWT's capabilities and its libraries. It will discuss the concept of deferred binding and give you details about autogenerating code for your applications. It will also explain the bootstrap/startup process followed by a GWT application and discuss the various files created by the GWT compiler.



GWT Basics and a First Application

GWT is an open source framework that allows you to develop Ajax-based applications in the Java language and provides tools to convert the Java code into JavaScript and HTML. This frees you, the developer, from the burden of rewriting your JavaScript to suit the peculiarities and lack of standards support in all the various browsers in use on people's computers.

Since its June 2006 release, the GWT framework has made a tremendous impact on the developer community engaged in developing Ajax-based web applications. This chapter will help you get started with GWT and help you understand how GWT fits into the next Ajax-based application you develop.

Note GWT currently supports Internet Explorer 6 and 7; Firefox 1.0, 1.5, and 2.0; Mozilla; Safari 2.0; and Opera 9.0. The GWT compiler converts the Java classes into separate script files in compliance with the JavaScript understood by the various JavaScript engines found in the underlying web browsers. What this means for you as a developer is that Java's original promise of "write once, run anywhere" now applies to the world of Ajax-based applications. This allows you to focus more on the internal domain and application logic, rather than spending precious time making the application work across multiple browsers.

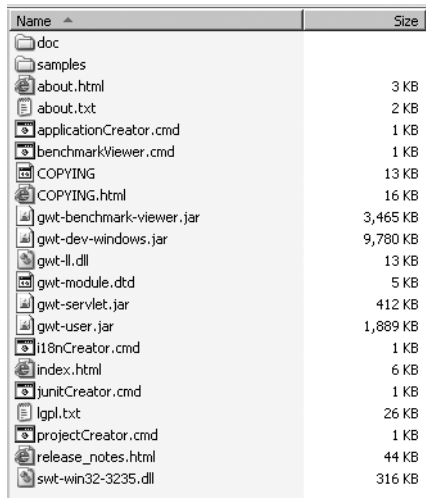
Debugging has long been a major problem for developers tasked with writing JavaScript. Although over the past few years a few very good JavaScript debugging tools have become available, most of them, such as Firebug, are designed to integrate closely with web browsers rather than with the modern IDEs used for Java development, such as Eclipse. GWT solves this problem by providing a mechanism to directly run and test the application from within the IDE as you'd do with typical Java code.

This chapter will start with the details of downloading and setting up GWT on your machine. Then it will go step by step through developing and running a sample application, using the tools and utilities provided by the GWT framework. The chapter will then dissect an entire sample application and explore its various parts to solidify your understanding of the different components of a GWT application. Finally, the chapter will close by showing how to write another application on your own without using the tools provided by GWT so that you have a clear understanding of the process involved in developing an application using GWT.

Setting Up Your GWT Environment

As of this writing, the current version of GWT is 1.4.61. You can download it from <http://code.google.com/webtoolkit/download.html>.

I downloaded the file named `gwt-windows-1.4.61.zip`. Then I extracted the `.zip` file to the root directory (`C:\`) of my system. In my case, I unzipped GWT at the location `C:\gwt-windows-1.4.61`. I renamed the folder to `C:\gwt` and will reference this name across all the code in this book and in the source code for the samples of this book. Figure 1-1 shows the contents of this directory.



Name	Size
doc	
samples	
about.html	3 KB
about.txt	2 KB
applicationCreator.cmd	1 KB
benchmarkViewer.cmd	1 KB
COPYING	13 KB
COPYING.html	16 KB
gwt-benchmark-viewer.jar	3,465 KB
gwt-dev-windows.jar	9,780 KB
gwt-ll.dll	13 KB
gwt-module.dtd	5 KB
gwt-servlet.jar	412 KB
gwt-user.jar	1,889 KB
i18nCreator.cmd	1 KB
index.html	6 KB
unitCreator.cmd	1 KB
lgpl.txt	26 KB
projectCreator.cmd	1 KB
release_notes.html	44 KB
swt-win32-3235.dll	316 KB

Figure 1-1. List of files after extracting the GWT package

Hosted Mode vs. Web Mode

Before discussing the files shown in Figure 1-1, it's important to discuss the various modes, web and hosted, in which an application can be run using GWT.

Web Mode

Traditionally, developers of web-based applications had to go through the complete cycle of building, packaging, and deploying the application on a web server to test each new feature that was implemented. This slow and time-consuming process led to overly long development times and project delays.

In *web mode*, the application is run as pure JavaScript and HTML. These files are the result of compiling Java source code of your GWT-based modules by using the Java-to-JavaScript compiler. The JavaScript and HTML files obtained by the compilation step are used for the actual deployment to production environments.

Hosted Mode

In addition to traditional web mode, GWT provides another approach for testing and running applications known as *hosted mode*. In this mode, the actual Java byte code of the classes is run within the Java Virtual Machine (JVM). While web mode requires you to deploy your application to a web server, hosted mode allows a developer to run the application right off their favorite IDE in a web browser embedded in the framework. This greatly reduces the amount of time required to verify, test, and debug changes in an application.

A major advantage of Java byte code being run in hosted mode is the ability to debug the application, in the Java language, by using powerful IDEs available for the Java language. The debugging capabilities of a modern Java IDE are far more mature than the evolving JavaScript-related tools.

Figure 1-2 shows a GWT application being debugged in the Eclipse IDE.

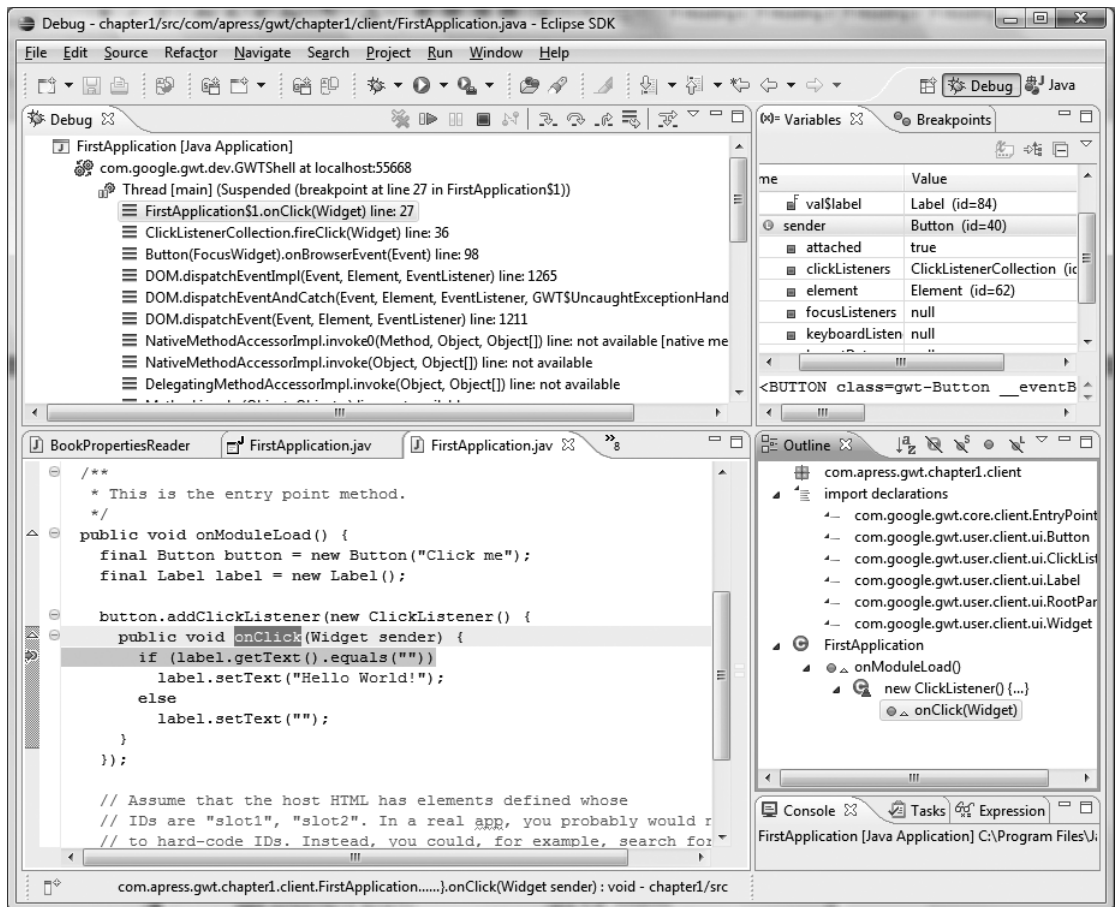


Figure 1-2. Debugging a GWT application directly in Eclipse while running the application in hosted mode

What Are All Those GWT Files For?

Here are details about some of the particularly important files shown in Figure 1-1:

`gwt-servlet.jar`: This provides all the GWT classes (including all the required RPC-related classes) that your application will need. This JAR file should be included in the application when running the application in web mode.

`gwt-user.jar`: This JAR file is needed when the application is run in hosted mode, that is, inside the web server bundled inside the GWT libraries. (This file should never be included as part of the application when deploying the application to an actual production environment.)

`gwt-dev-windows.jar`: This JAR file is needed if you want to run your application in hosted mode. Rather than a traditional web application development cycle of building, deploying, and testing the application in a web browser, GWT provides an internal hosted mode that you can use to test and debug the application by running the application in an embedded browser window. When the application is run in hosted mode, the startup class of your application should be `com.google.gwt.dev.GWTShell`.

`gwt-benchmark-viewer.jar`: This JAR file contains the benchmarking classes that you can use to create benchmarking tests for your applications (GWT has a `Benchmark` class that can be subclassed and used for common benchmarking needs). You can use the `benchmarkViewer` tool (`benchmarkViewer.cmd`, located in the root of the GWT directory) to derive charts/reports from the XML file that is generated when running the benchmarking tests. You'll learn more about benchmark testing in Chapter 7.

Table 1-1 lists the various JAR files, which should be included in your application when running it in the different modes supported by GWT.

Table 1-1. *JAR Files to Be Included for Web/Hosted Mode*

Application Mode	JARs to Be Included
Web mode	<code>gwt-servlet.jar</code>
Hosted mode	<code>gwt-user.jar</code> and <code>gwt-dev-windows.jar</code> (or <code>gwt-dev-linux.jar</code>)

Creating Your First GWT Application

The following sections will go over the steps needed to create an application using GWT. You will also develop your first application as you go through these steps.

Tools for Creating a Project

The developers of the GWT framework, by providing a number of easy-to-use utilities, have done their part in making starting a project with GWT as easy as possible. The following sections will cover the utilities you'll use to build your first application using GWT in this chapter.

projectCreator

This utility creates the project structure and an empty Eclipse project or Ant build file (or both) for your application. The command takes the following flags/parameters:

```
projectCreator [-ant projectName] [-eclipse projectName] [-out dir]
               [-overwrite] [-ignore]
```

Specifically, the flags/parameters are as follows:

- -ant generates a project build file to compile the source of the application. (The build file is named <projectName>.ant.xml.)
- -eclipse generates an Eclipse project (.project and .classpath files).
- -out is the directory where the output files will be written (default is the current directory).
- -overwrite overwrites any existing files.
- -ignore ignores any existing files (does not overwrite them).

Note You have to specify either the -ant or -eclipse flag to use this command.

For example, if you want to create a new project named chapter1 in a directory named chapter1, then you should run `projectCreator -eclipse chapter1` while inside the chapter1 directory.

Listing 1-1 shows the result of executing this command.

Listing 1-1. Project Structure and Corresponding Files Created by Running projectCreator

```
C:\gwt\chapter1>projectCreator -eclipse chapter1
Created directory C:\gwt\chapter1\src
Created directory C:\gwt\chapter1\test
Created file C:\gwt\chapter1\.project
Created file C:\gwt\chapter1\.classpath
```

Executing the command shown previously creates the standard project structure in the chapter1 directory (namely, src and test directories for Java source and tests, respectively) and the .project and .classpath files for the Eclipse project for the application.

As another example, say instead that you want to create the project under a subdirectory mysub of your current directory. Then you should use the -out flag as mentioned earlier and pass the name of the subdirectory where the project should be created. Listing 1-2 shows the output of using this flag in the example.

Listing 1-2. *Project Structure and Ant Build File Created in a Specified Folder by Running projectCreator with the -ant and -out Flags*

```
C:\gwt>projectCreator.cmd -ant MySubProject -out mysub
Created directory mysub\src
Created directory mysub\test
Created file mysub\MySubProject.ant.xml
```

Note You should enter the gwt directory to the path of your system so the shell can recognize the utilities when they are called from outside the main gwt directory. In Windows, this would mean adding C:\gwt to your system's path variable.

applicationCreator

This utility helps create a basic starter application for a GWT-based Ajax application. By using the -eclipse flag with the utility, you can also create a launch configuration for debugging the application in Eclipse. The command takes the following flags/parameters:

```
applicationCreator [-eclipse projectName] [-out dir] [-overwrite]
                    [-ignore] className
```

Specifically, the flags/parameters are as follows:

- -eclipse creates a launch configuration for debugging the application in Eclipse.
- -out is the directory where the output files will be written (default is the current directory).
- -overwrite overwrites any existing files.
- -ignore ignores any existing files (does not overwrite them).
- className is the fully qualified name of the entry-point class in the application.

If you want to create the sample project in the chapter1 directory with the base GWT package location as com.apress.gwt.chapter1, then you should run applicationCreator -eclipse chapter1 com.apress.gwt.chapter1.client.FirstApplication from the shell while in the chapter1 directory. Listing 1-3 demonstrates this scenario.

Listing 1-3. *Sample Project Including Launch Configuration Files Created by Running the applicationCreator Command with the -eclipse Flag*

```
C:\gwt\chapter1>applicationCreator -eclipse chapter1 ➡
com.apress.gwt.chapter1.client.FirstApplication
Created directory C:\gwt\chapter1\src\com\apress\gwt\chapter1
Created directory C:\gwt\chapter1\src\com\apress\gwt\chapter1\client
```

```
Created directory C:\gwt\chapter1\src\com\apress\gwt\chapter1\public
Created file C:\gwt\chapter1\src\com\apress\gwt\chapter1\FirstApplication.gwt.xml
Created file ➡
    C:\gwt\chapter1\src\com\apress\gwt\chapter1\public\FirstApplication.html
Created file ➡
    C:\gwt\chapter1\src\com\apress\gwt\chapter1\client\FirstApplication.java
Created file C:\gwt\chapter1\FirstApplication.launch
Created file C:\gwt\chapter1\FirstApplication-shell.cmd
Created file C:\gwt\chapter1\FirstApplication-compile.cmd
```

junitCreator

This utility creates a JUnit test and scripts that you can use for testing the various components of the application in both hosted and web modes. You will learn more about this utility and the details of testing GWT-based applications in Chapter 7.

i18nCreator

This utility creates scripts and a property file for internationalizing your applications. (You'll learn more about internationalization and this utility in Chapter 8.)

Running the Application Using Generated Scripts

You can execute the application in hosted mode by using `FirstApplication-shell.cmd`. With your current directory being your application's home directory, execute the command `<applicationName>-shell.cmd` on the command prompt to execute the application. You will see the hosted browser being loaded and the application running. Listing 1-4 shows the snippet with the script being used to run the application.

Listing 1-4. Command to Run Your First Application in Hosted Mode

```
C:\gwt\chapter1> FirstApplication-shell.cmd
```

Note You should ensure that all the Java files are compiled before running the application using generated scripts such as `FirstApplication-shell.cmd` mentioned previously. You can enable autocompile in the Eclipse IDE by select the Project ► Build Automatically option if it is not enabled already. This entire book will assume that this option is enabled and the project is fully built before running the application.

Figure 1-3 and Figure 1-4 show the result of running the application in hosted mode by running the script in Listing 1-4.

Specifically, Figure 1-3 shows the web server window. This window displays the application log and stack trace, if any, for the application.

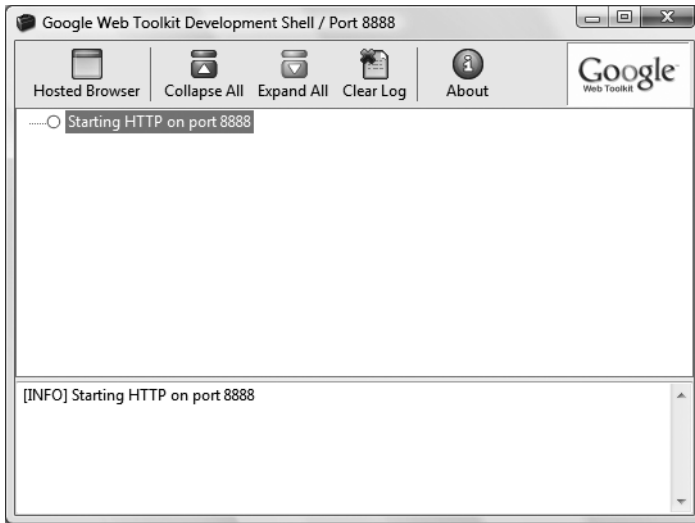


Figure 1-3. Running your sample application in hosted mode. The embedded web server window shows the application log and stack trace in case of any errors.

Figure 1-4 shows the actual web browser window with the application running in it.

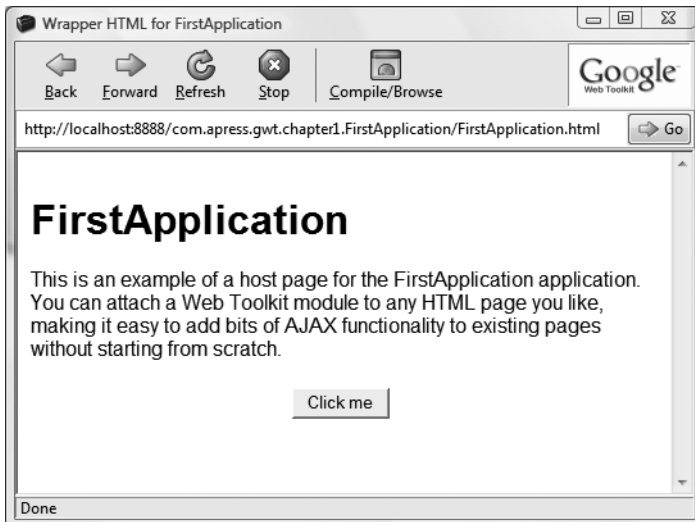


Figure 1-4. Running your sample application in hosted mode. The embedded web browser window shows the application running in it.

Understanding the Generated Scripts to Compile and Run the Application

The applicationCreator utility creates two script files for helping the development process. These files contain the Java commands to compile the application and to run the application in hosted mode (running the entry-point class). The file contents for the sample application are as follows:

FirstApplication-shell.cmd: This file loads the start page of the GWT module in the hosted browser and contains the following Java command:

```
@java -cp "%~dp0\src;%~dp0\bin;C:/gwt/gwt-user.jar;C:/gwt/gwt-dev-windows.jar"
    com.google.gwt.dev.GWTShell -out "%~dp0\www" %*
    com.apress.gwt.chapter1.FirstApplication/FirstApplication.html
```

FirstApplication-compile.cmd: This file is responsible for compiling the GWT module and outputs the resultant files in the folder named www. It contains the following Java command:

```
@java -cp "%~dp0\src;%~dp0\bin;C:/gwt/gwt-user.jar;C:/gwt/gwt-dev-windows.jar"
    com.google.gwt.dev.GWTCompiler -out "%~dp0\www" %*
    com.apress.gwt.chapter1.FirstApplication
```

Working with Modules in GWT

GWT applications are basically structured in terms of modules. Each module defines a well-defined functionality or component, and these modules are combined to make a full-fledged application.

Structure of a Module File

The recommended (default) structure of a GWT-based module (with the corresponding structure as an example from your first application) is as follows:

- **moduleName.gwt.xml:** This file includes the configuration of the module. For example:
com/apress/gwt/chapter1/ FirstApplication.gwt.xml
- **client package:** A client subpackage (and corresponding subpackages) with client-side code. For example:
com/apress/gwt/chapter1/client
com/apress/gwt/chapter1/client/data ... and so on
- **public package:** A public subpackage (and corresponding subpackages) with static web content. For example:
com/apress/gwt/chapter1/public
- **server package:** A server subpackage (and corresponding subpackages) with server-side code. For example:
com/apress/gwt/chapter1/server
com/apress/gwt/chapter1/server/data ... and so on

Note Even though using the listed module structure is just a recommendation, it's advisable that you stick to the previous structure because the entire community in general is sticking to this, and your modules will therefore be more easily understandable by others.

A *module* in GWT is simply an XML file and combines the entire configuration that your application would need in a single file. GWT modules define the entry point into the application. In the example discussed in this chapter, the module file is located at `chapter1/src/com/apress/gwt/chapter1` and is named `FirstApplication.gwt.xml`. Listing 1-5 shows the contents of the module file in the sample application.

Listing 1-5. *Contents of the Module File (FirstApplication.gwt.xml) for the Sample Application*

```
<module>
  <!-- Inherit the core Web Toolkit stuff. -->
  <inherits name='com.google.gwt.user.User' />

  <!--Specify the app entry-point class. -->
  <entry-point class='com.apress.gwt.chapter1.client.FirstApplication' />
</module>
```

Understanding Different Tags in a Module File

You can use the module file to define a number of properties related to the application:

- The entry-point class property
- The inherits property
- The source and public properties
- The servlet property
- The stylesheet property
- The script property
- The extend-property property

These are described in more detail in the following sections.

The entry-point class Property

When a module is loaded, the entry-point classes defined in the module's configuration file are instantiated, and the corresponding `onModuleLoad()` methods of these entry-point classes are called. This includes the entry-point classes of the inherited modules as well.

In the GWT library, `EntryPoint` is basically defined as an interface. This interface declares the `onModuleLoad()` method, which, as defined earlier, is automatically called when the module is loaded. (Adding an entry-point class is optional. However, you can add any number of entry-point classes in your application by including them in the configuration file of your application's module.)

Here's an example:

```
<entry-point class='com.apress.gwt.chapter1.client.FirstApplication' />
```

As you can see in the application's module file (`FirstApplication.gwt.xml`), the entry-point class for the application is defined as `com.apress.gwt.chapter1.client.FirstApplication`. Listing 1-6 shows the code for the entry-point class.

Listing 1-6. *Contents of the Entry-Point Class for the Sample Application*

```
package com.apress.gwt.chapter1.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 FirstApplication 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" and "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);
    }
}
```

The inherits Property

Inheriting a module is like including the entire configuration and code of that module in your application. This helps to reuse the existing functionality of other modules in your application. The configuration file has an `inherits` element that is used to inherit other modules in your application's module. You can inherit any number of modules by using this element.

Here's an example:

```
<inherits name='com.google.gwt.user.User' />
```

The source and public Properties

These two properties help declare the location of the source folder for the client code and for the public path. These two locations are relative to the path where the module XML file containing these attributes is found. If either of these attributes is missing, then default values for these properties are assumed.

For example, `<source path="myModuleClient"/>` defines that the `myModuleClient` directory and all its subpackages contain the client code, and all Java source files found in these locations adhere to the rules of GWT related to client-side code.

If the source element is missing, then `<source path="client">` is assumed as the default definition.

`<public path="myPublic"/>` defines that the `myPublic` directory acts as the root to the public path, and all files found in this folder and all its subfolders are directly accessible by external clients. The files found here are copied to the output directory during the translation steps followed by the GWT compiler.

If the public element is missing, then `<public path="public">` is assumed as the default definition.

If the module containing the previous property is located in the `com/apress/gwt/chapter1` folder and the module defines the public property as `<public path="myPublic"/>`, then the public folder for the application is located in the `com/apress/gwt/chapter1/myPublic` folder.

Note The `public` element allows filtering using patterns. This allows you to decide which resources to copy to the output directory while compiling the GWT module.

The servlet Property

This property allows you to test remote procedure calls (RPCs) by loading the servlets mapped to a specific path. (You are allowed to map all the servlets needed in your application by using this property.)

Here's an example:

```
<servlet path="url-path" class="classname"/>
```

The `url-path` value of the `path` attribute defines the location where the corresponding servlet for the application is mapped. The client should use the same mapping when registering

the call using the `ServiceDefTarget.setServiceEntryPoint(String)` method. (This will be explained in detail in Chapter 4.)

The stylesheet Property

This property allows you to add a CSS file to the module.

Here's an example:

```
<stylesheet src="css/Mycss.css"/>
```

The script Property

This property allows you to add external JavaScript code/libraries to the module.

Here's an example:

```
<script src="js/Mylibrary.js"/>
```

The extend-property Property

This property allows you to add values to a client property. Any number of such values may be introduced by adding them to the comma-separated list of values.

Here's an example:

```
<extend-property name="client-property-name" values="comma-separated-values"/>
```

This feature is most used in specifying locales for your application during internationalization. You will learn more about internationalization and using this property in Chapter 8.

Creating the Host HTML File

Listing 1-7 shows a sample startup HTML file that demonstrates how to add the GWT module to your application so that it is loaded when your application is started. It also shows how to add history support to the application. (Chapter 9 discusses how to handle history support in your applications.)

Listing 1-7. *Sample Host HTML File to Load the Module and Add History Support in the Application*

```
<html>
  <head>
    ...
    <!-- This script loads your compiled module. If you add any GWT meta tags,
         they must be added before this line -->
    <script language='javascript'
      src='com.apress.gwt.chapter1.FirstApplication.nocache.js'>
    </script>
  </head>
```



```

<body>
  <!-- OPTIONAL: include this if you want history support -->
  <iframe src="javascript:''" id="__gwt_historyFrame"
    style="width:0;height:0;border:0"></iframe>

    <!--Rest of HTML body goes here, for example, we have a table with two
    columns -- >
    <table align=center>
      <tr>
        <td id="slot1"/>
        <td id="slot2"/>
      </tr>
    </table>
</body>
</html>

```

Here I'll briefly explain some of the important parts of the previous HTML page:

- `<script ... src='com.apress.gwt.chapter1.FirstApplication.nocache.js'>`: After compilation, the application module is stored as a JavaScript file (with a `nocache.js` extension). The path for this file is added to the HTML page so that it can be downloaded and loaded when the page is referenced.
- `<iframe src="javascript:''" id="__gwt_historyFrame" ...>`: This adds history support to the application. This is an interesting functionality provided by the GWT framework that makes adding history management support to your application a simple task. (Chapter 9 discusses how to add history support to your applications.)

Steps to Create a GWT Application

The process of creating a GWT application can be broken down into the following four steps:

1. Create the basic project structure with the corresponding package. Add client, server, and public subpackages.
2. Add the Application module file, named `<appName>.gwt.xml`, to the basic package. Add the corresponding configuration details to the file.
3. Create an entry-point class (by implementing the `EntryPoint` interface) named `<appName>.java` in the client subpackage, and override the `onModuleLoad()` method with the required functionality.
4. Create an HTML file named `<appName>.html` in your application's public folder, with a script `src` reference to your application's package and with `<appName>.nocache.js` appended.

Creating Another Application Step-by-Step

With all these basic details behind you, let's create another simple application without using the GWT tools in order to solidify your understanding of using and configuring a simple GWT application. There will be numerous situations where you'll want to add GWT support to your existing applications, and this example will serve as good starting point to understanding the integration step required at that point.

In this application, I'll show how to build a simplified version of a news-serving client that will iterate through a set of news stories in the web browser. (In future chapters, you will extend the same application to support displaying real-time news events in a proper news display widget, but for simplicity here, this example will just flash news entries in the browser's window.)

You will follow the steps mentioned earlier to create your application. So, here goes . . .

Creating the Basic Project Structure

As mentioned earlier, the first step is to create the basic project structure with the corresponding base package, as well as the client, server, and public subpackages.

So, let's start by creating a new Java project in Eclipse. I created the project in a folder called `NewsClient` in the `gwt` directory (`C:\gwt`) of my system. Then I added a source folder named `src` in the project and added necessary packages to it so that the structure looks like the one shown in Figure 1-5. (I also added the JRE system library and `gwt-user.jar` to the build path of the project in Eclipse.) I did not add the server package at this time because the application is not making any RPC calls to server-side logic as of yet. You will learn about RPC (server-side code) starting in Chapter 4 in the book.

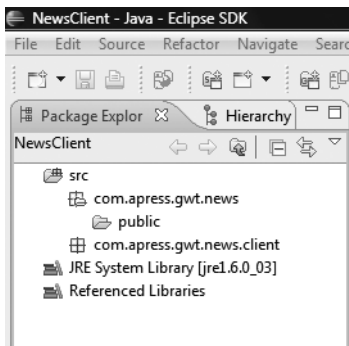


Figure 1-5. *Structure of the NewsClient application*

Adding the Module File

As mentioned earlier, the second step is to add the application's module file, named `NewsClient.gwt.xml`, to the base package, as shown in Figure 1-6.



Figure 1-6. Adding the module file named `NewsClient.gwt.xml`

In addition, let's quickly add the configuration details to the module file (`NewsClient.gwt.xml`). Listing 1-8 shows the `NewsClient` application's module configuration file.

Listing 1-8. Contents of the Module File for the `NewsClient` Application

```
<module>
  <!-- Inherit the core Web Toolkit.          -->
  <inherits name='com.google.gwt.user.User' />

  <!-- Specify the app entry-point class.      -->
  <entry-point class='com.apress.gwt.news.client.NewsClient' />
</module>
```

As you can see, the module file references the entry-point class as `com.google.gwt.news.client.NewsClient`, so that takes you to step 3.

Caution The client entry-point class and the module should have the same prefix as part of their names. They're `NewsClient.gwt.xml` and `NewsClient.java` in this example.

Creating the Entry-Point Class

As mentioned earlier, the third step is to create an entry-point class, named `NewsClient.java`, in the client subpackage, as shown in Figure 1-7, and override the `onModuleLoad()` method of the `EntryPoint` interface with the required functionality.



Figure 1-7. Adding the `NewsClient` entry-point class to the application

Listing 1-9 shows the code for the NewsClient entry-point class.

Listing 1-9. *Contents of the Entry-Point Class (Named NewsClient.java) for the Application*

```
package com.apress.gwt.news.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Random;
import com.google.gwt.user.client.Timer;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;

/**
 * Entry-point class for the news client application displays a news entry
 * every two seconds on the browser window.
 *
 * @author Vipul Gupta (vipulgupta.vg@gmail.com)
 */
public class NewsClient implements EntryPoint{

    private final String[] newsEntries = new String[] {
        "News Entry 1", "Another News Entry", "Yet another news entry",
        "One Final news Entry" };

    public void onModuleLoad() {

        final Label label = new Label();

        // Create a new timer that keeps changing the news text
        Timer t = new Timer() {
            public void run() {
                label.setText(getNewsEntry());
            }
        };

        // Schedule the timer to run every two seconds.
        t.scheduleRepeating (2000);

        RootPanel.get("newsEntryDivId").add(label);
    }

    private String getNewsEntry() {
        return newsEntries[Random.nextInt(newsEntries.length)];
    }
}
```

Let's inspect what's happening in this class in detail:

1. You start by creating a dummy-entry array containing some strings that act as dummy news entries for the class:

```
private final String[] newsEntries = new String[] { ...
```

Note In an actual production environment, the client will make a call to the server, which will return actual real-time news events for display. We will simulate the same type of application in Chapter 4.

2. You then define the `onModuleLoad()` method of the `EntryPoint` interface and give a concrete implementation to it. Eventually this method will be compiled to JavaScript by using the GWT compiler and will be called when the page containing the link to this module is loaded.

In this method definition, I have created a label in which the news-entry text will be shown. I have also created a `Timer` object and scheduled it to run every two seconds. The overridden `run()` method of the `Timer` class is called at every scheduled interval of the `Timer` object (two seconds in this example):

```
public void run() {  
    label.setText(getNewsEntry());  
}
```

The `Timer` object is configured to execute the `run()` method every two seconds by the following code:

```
t.scheduleRepeating (2000);
```

Note The `Timer` (`com.google.gwt.user.client.Timer`) class in the GWT library serves the same purpose as the `Timer` class in the Java library and has been implemented to be safe across the various browsers. The methods provided by the `Timer` class are as follows:

- `cancel()`: Cancels the timer
 - `run()`: Called when the timer is executed
 - `schedule(int)`: Schedules the timer to execute once after `int` milliseconds
 - `scheduleRepeating(int)`: Schedules the timer to execute repeatedly after `int` milliseconds
-

The `run()` method in the sample application just updates the text of the label with one of the randomly chosen strings from the `newsEntries` array that it gets by calling the `getNewsEntry()` method:

```
private String getNewsEntry() {  
    return newsEntries[Random.nextInt(newsEntries.length)];  
}
```

The `getNewsEntry()` method as shown here just uses the `Random` class to get an index into the `newsEntries` array, and then the `getNewsEntry()` method returns the string at that array location.

Note The `Random` (`com.google.gwt.user.client.Random`) class in the GWT library serves the same purpose as the `Random` class in the Java library. The major difference from the Java version is that the class in the GWT library cannot be seeded to start with. As a result, you can never ensure the same sequence of generated values. The methods provided by the `Random` class in the GWT library are as follows:

- `nextBoolean()`: Returns true or false with roughly equal probability
 - `nextDouble()`: Returns a random double between 0 (inclusive) and 1 (exclusive)
 - `nextInt()`: Returns a random int between -2147483648 and 2147483647 (inclusive) with roughly equal probability of returning any particular int in this range
 - `nextInt(int)`: Returns a random int between 0 (inclusive) and `upperBound` (exclusive) with roughly equal probability of returning any particular int in this range
-

The `onModuleLoad()` method in the class shown earlier has the following statement at the end of the method:

```
RootPanel.get("newsEntryDivId").add(label);
```

The `RootPanel` gets a reference to the panel to which all your application widgets must ultimately be added. (`RootPanel` returns the reference to the document's body in the browser.) You would generally inject the starting point of your application into an HTML page. You can do this by using the `RootPanel` class's `get()` methods. In this example, you added the label to the element with an ID of `newsEntryDivId` in the HTML page hosting the application. The next step is to create this hosting HTML page.

Creating the Host HTML File

As mentioned earlier, the fourth step is to create an HTML file, named `NewsClient.html`, in your application's public folder, as shown in Figure 1-8, with a script `src` reference to the application's package and with `NewsClient.nocache.js` appended to the package.

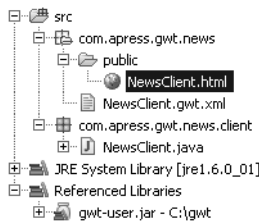


Figure 1-8. Adding the main HTML file named `NewsClient.html`

Listing 1-10 shows the code for the host HTML file (`NewsClient.html`) for the application.

Listing 1-10. *Contents of the Host HTML File (Named `NewsClient.html`) for the Application*

```
<html>
  <head>
    <!-- This script loads your compiled GWT module.  -->
    <script language='javascript'
      src='com.apress.gwt.news.NewsClient.nocache.js'></script>
  </head>

  <body>
    <div id="newsEntryDivId"/>
  </body>
</html>
```

Running the Application in Hosted Mode

With all this behind you, you should now run the `NewsClient` application in hosted mode to see the application in action. The class named `GWTShell` is used to run the application in hosted mode and is explained next.

Understanding and Using `GWTShell`

As mentioned earlier in this chapter, the startup class for running the application in hosted mode is `GWTShell` (`com.google.gwt.dev.GWTShell`), and you can use it to run your applications.

The `GWTShell` class supports the following command-line flags/parameters:

```
GWTShell [-port port-number | "auto"] [-noserver] [-whitelist whitelist-string]
         [-blacklist blacklist-string] [-logLevel level] [-gen dir] [-out dir]
         [-style style] [url]
```

You can find this class in `gwt-dev-windows.jar` (or `gwt-dev-linux.jar`). Table 1-2 lists `GWTShell`'s command-line options.

Table 1-2. *GWTShell's Command-Line Options*

Flag Name	Description
-port	Runs an embedded Tomcat instance on the specified port (defaults to 8888)
-noserver	Prevents the embedded Tomcat server from running, even if a port is specified
-whitelist	Allows the user to browse URLs that match the specified regular expressions (comma separated or space separated)
-blacklist	Prevents the user from browsing URLs that match the specified regexes (comma separated or space separated)
-logLevel	Specifies the level of logging detail: ERROR, WARN, INFO, TRACE, DEBUG, SPAM, or ALL

Flag Name	Description
-gen	Specifies the directory into which generated files will be written for review
-out	Specifies the directory to write output files into (defaults to current)
-style	Specifies the script output style; GWT supports OBF[USCATED], PRETTY, and DETAILED (defaults to OBF)
url	Automatically launches the specified URL

To run your project from the command line, you need to set up the proper classpath for the Java interpreter. The classpath should include `gwt-user.jar` (needed by the application), `gwt-dev-windows.jar` (or `gwt-dev-linux.jar` in case of Linux; the startup class `GWTShell` is in this JAR), and the `src` folder of the application (contains the package/classes/module of the application).

So, at your command prompt (from the application's home directory), type the command listed in Listing 1-11 to get the application running.

Listing 1-11. *Command to Run the NewsClient Application*

```
C:\gwt\NewsClient> java -cp c:\gwt\gwt-dev-windows.jar;c:\gwt\gwt-user.jar;src
                    com.google.gwt.dev.GWTShell
                    com.apress.gwt.news.NewsClient/NewsClient.html
```

Note Listing 1-11 is a single command, but individual elements of it have been broken into multiple lines to fit the width of this page. You can just type the same command on a single line and see the hosted browser load with your application.

I'll now break down the command in Listing 1-11 so you can understand it:

1. You start by setting the classpath for the Java command, as explained earlier:

```
-cp c:\gwt\gwt-dev-windows.jar;c:\gwt\gwt-user.jar;src
```

2. Next you pass the main bootstrap class to the Java command to run:

```
com.google.gwt.dev.GWTShell
```

3. You then pass the URL to the `GWTShell` command to launch:

```
com.apress.gwt.news.NewsClient/NewsClient.html
```

The URL passed to the `GWTShell` class shown earlier deserves more explanation. The URL of the application is automatically decided in case of hosted mode and is based on the name of the module (and the corresponding package).

For example, if your module is named `NewsClient` and it resides in a package named `com.apress.gwt.news`, then the URL in hosted mode would be `com.apress.gwt.news.NewsClient/NewsClient.html`, and that's what you used in the earlier example.

So, the format of the URL is `<package>.<moduleName>/<moduleName>.html`.

Figure 1-9, Figure 1-10, and Figure 1-11 show the NewsClient application in hosted mode at different points of time.

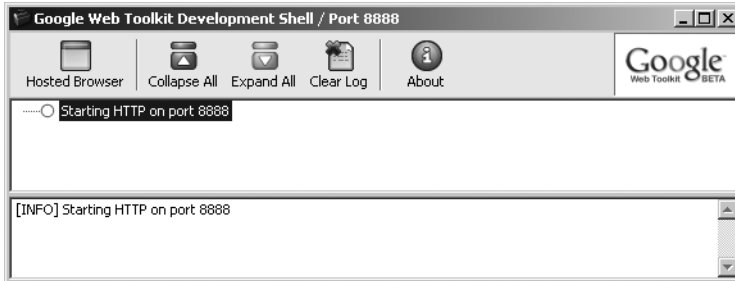


Figure 1-9. *Embedded web server window that shows the application log and error stack trace, if any*

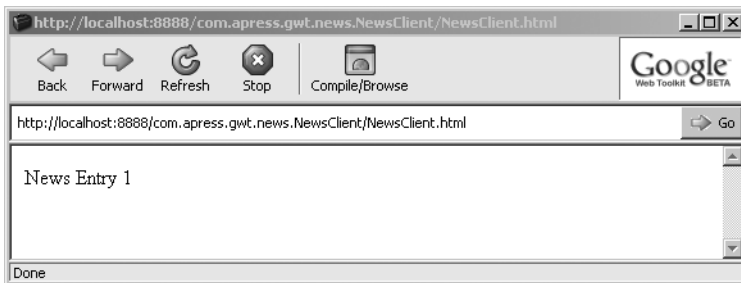


Figure 1-10. *NewsClient application at time T1 showing a news entry*



Figure 1-11. *NewsClient application at time T2 showing a different news entry*

Summary

This chapter covered various aspects of the GWT framework, including setting up the GWT environment and basic application development with and without the tools provided by the framework. It also discussed the structure of a GWT application and the configuration required for a GWT module.

In addition, the chapter discussed the various JAR files available in the framework and the various modes in which the application can be run. None of the applications demonstrated in this chapter interfaced with server-side logic and made any RPC calls, even though that would have made much more sense, especially in the NewsClient application. However, this avoided complicating things too much in this first chapter of the book.

Lots of the topics mentioned in this chapter are probably new to you, so it's highly recommended that you try all the examples in this chapter (and the rest of the book too) by hand to solidify your understanding of GWT. The book will delve into a lot more details about all the topics mentioned here and the GWT framework itself in the remaining chapters.

