GWT IN ACTION

Easy Ajax with the Google Web Toolkit

SAMPLE CHAPTER

Robert Hanson
Adam Tacy

MANNING

GWT in Action

by Robert Hanson
and Adam Tacy

Chapter 2

brief contents

2 Creating the default application

This chapter covers

- Starting a new GWT project
- Preparing for internationalization
- Creating JUnit tests
- Importing the application into your IDE

Chapter 1 got you up and running with your first GWT application; it was a quick and dirty approach, but it worked, and you quickly had a fully functioning Tic-Tac-Toe application. Now it's time to start shoring up your approach and take the first few steps toward building a full-scale application. Over the next two chapters, you'll create the first version of the Dashboard application, which is an example you'll keep expanding and adding to as you progress through this book.

You should read this chapter and the next together, because they're both needed to create the first version of the Dashboard. We split the discussion into two chapters so that in this chapter, we can explain the tasks that are necessary for every GWT application—these initial steps result in the production of an application that we call the *GWT default application*. Then, in chapter 3, we go through the steps needed to specialize the default application in order to produce your own application (in this case, the GWT Dashboard). If you're using a wizard provided by an IDE, then it may perform the steps necessary to generate the default application for you.

To help us explain how the contents of the two chapters go together, we first present a typical development lifecycle, which a real-world programmer might apply when developing GWT applications. With one possible specific high-level web development process in mind, chapter 2 looks again at the tools you used in chapter 1; but this time we'll consider them in detail. You'll actively use those tools shown first in chapter 1, which we'll refer to as the *creation tools*, to create the default directory structure and files used as the basis for the example Dashboard application. This step is performed for all GWT applications you build, and the output is the GWT default application that you first saw in figure 1.4 in section 1.3.1.

Let's take the first step toward building the Dashboard's directory and code structure by examining how you build the GWT default application.

2.1 The GWT application development lifecycle

You can create GWT applications three ways. The first way is to use the set of creation tools provided in the GWT download. These tools let you quickly create a directory and file structure suitable for your GWT application, which is also in line with the structure the GWT compiler expects. You saw these tools in use when you created the Tic-Tac-Toe application in chapter 1. If you use these creation tools, then the result is, by default, independent of any IDE that you may be using. By providing the extra `-eclipse` flag to the creation tools, you direct them to generate an additional set of files that enable the whole structure to be easily imported into the Eclipse IDE. (Eclipse isn't the only IDE you can use; GWT just makes it easy to integrate with Eclipse without using an IDE-specific wizard—it's also possible to import Eclipse projects into some of the other more common IDEs.)

As GWT matures, more third-party tools are appearing on the market that hide the need to use the GWT creation tools. In some cases, these third-party tools come in the form of wizards that IDEs support; in other cases, new IDEs are being built or tweaked specifically to support GWT application development. If you're using one of these tools, then generally the tool provides specific use instructions; but often, running an IDE-specific wizard results in a default application similar to that produced by the GWT creation tools.

If you don't want to use the GWT-provided tools or a tool provided by your IDE, then it's possible to create the directory structure and basic files yourself. You may want to do this, for example, if you're working in an environment where system restrictions prevent you from using the standard structure. Taking this approach means you will more than likely have to deal in more detail with the application's Module XML file (see chapter 9) in order to tell the compiler all the paths to the necessary files.

Figure 2.1 summarizes the three methods of creating GWT applications.

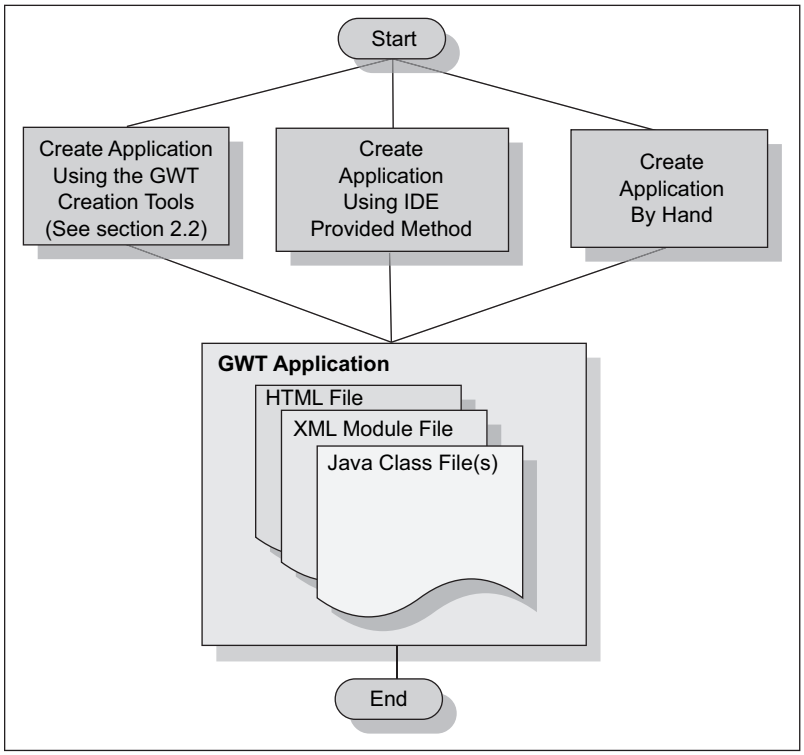

Figure 2.1 The three ways to create a GWT application: using the GWT tools, using a plug-in for an IDE, and creating the structure and files by hand. This chapter uses the GWT creation tools.

All three approaches lead to the production of the same set of basic files that represent the GWT default application—which is good, because all those files are necessary for your application to work. For simplicity at this stage in GWT's maturity, this book will follow the first approach—using the set of creation tools provided by the GWT distribution.

Looking at the other two approaches identified in figure 2.1, probably the most error-prone way to create a GWT application's structure and files is to do so by hand. We advise against this approach, although it can be useful if your environment forces you to use a different directory structure than the default. We won't discuss this "by hand" approach in this book; it isn't difficult, but it requires a lot of attention to detail to make sure all the necessary files are in the correct places, the hosted mode and compile tools have the correct classpaths set, and so on. We'd risk spending more of our time explaining that than getting going with development!

Even though we're using Eclipse in this book, GWT isn't tied to Eclipse; the files produced can be imported into any IDE, or your application can be created using the command-line tools. Eclipse, like any IDE, has pluses and minuses; but it's free and, probably more important, widely used. This book won't get into the pros and cons of any particular IDE. GWT provides great support at the creation-tool level for Eclipse, so we'll continue with that. Don't worry if you aren't using Eclipse as your IDE, in section 2.2.6, we'll look briefly at how you can import the files into other IDEs.

The GWT creation tools we discuss in this chapter—together with the GWT hosted-browser, GWT compiler, and your choice of web server and browsers—provide complete development and test environments. Figure 2.2 shows a simplified view of the lifecycle stages in which the creation tools and other tools just mentioned are typically used. (If you followed one of the other approaches indicated in figure 2.1, you would do so in figure 2.2's Stage 1.)

Table 2.1 details each stage of this typical web-application development lifecycle.

The remainder of this chapter and all of chapter 3 are given over to showing this theory in practice by stepping through each of the lifecycle stages in turn to produce the first basic version of the Dashboard application. You'll perform all the steps in Stage 1, using the GWT creation tools to create the directory and default code and load the application into the Eclipse IDE. (If you want to use your IDE's plug-in to create the structure for you, then you can try running it according to its manual now and pick us up again in chapter 3, where we should have similar structures.)

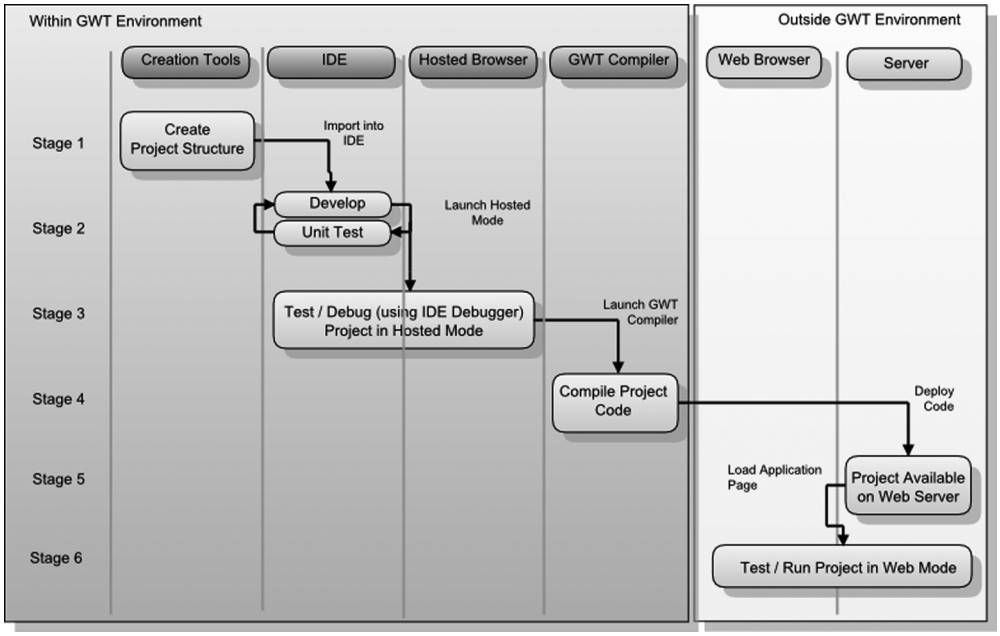

Figure 2.2 Generic lifecycle of GWT application development, showing along the top which tools are used and down the side the stage in which they're used. Some of the stages (for example, Stage 2) can be repeated many times before moving to the next stage. This is a waterfall approach; you may also follow a more rapid application development process by cycling between Stages 2, 3, 4, 5, and 6 as necessary.

Table 2.1 Stages involved in the typical development of a GWT application (the stages are shown in figure 2.2)

Stage	Description
1	The directory and code structure for the project is established. This can be performed using one of the three approaches suggested in figure 2.1—using the GWT creation tools, using an IDE-specific wizard, or by hand. If you use the GWT creation tools, they create a default application (you saw this in chapter 1). Often, if you used the GWT creation tools or the by-hand method, the next step you want to take is to import the basic outline of your project into an IDE. If you're using the GWT creation tools and the Eclipse IDE, then by adding the `-eclipse` flag to the command-line tools, you create all the additional files required to easily import your directory and code structure into Eclipse as an Eclipse project.
2	Once the directory and code structure is created, application development can proceed. Typically, you replace the default files created for you by the GWT creation tools or your IDE's wizard and add other files that you require for your application: additional Java classes, Cascading Style Sheets (CSS), images, and so on.

Table 2.1 Stages involved in the typical development of a GWT application (the stages are shown in figure 2.2) *(continued)*

Stage	Description
3	The development period typically contains several cycles that move between writing your application and testing in hosted mode, using the hosted-browser (Stage 3). You can launch hosted mode either directly from a shell window or from within your IDE. It acts as a managed environment, executing your code as pure Java code and deploying any server-side Java code you've developed into its own internal web server. Errors and exceptions that are raised in hosted mode, as well as any output from the GWT logging you may have included in your code, are safely captured by this managed environment. Compare this to web mode, where your application becomes JavaScript code and is executed directly by a web browser—it has no guaranteed safe management of exceptions and errors. Another benefit of hosted mode comes from the ability to link it to your IDE's debugger for both client and server-side Java code debugging.
4	When you're happy with the developed code in hosted mode, it's time to compile your Java code into JavaScript for use in web mode (Stage 4). You start compilation by invoking the GWT compiler: The resulting files are then ready to be viewed in a web browser. If you have only client-side code at this point, you can open the application directly from the filesystem; but if you have server-side code, you need to deploy the code to a web server. The compilation process produces a number of files that are required; chapter 17 discusses this process.
5	A compiled application is typically deployed to your standard test environment's web server so you can check the deployment process as well as ensure that the code executes correctly. Deploying is largely dependent on your web server; but if you have no server-side code, you can check web mode directly from your filesystem by double-clicking your application's HTML file from the compiled directory.
6	To finish development, check your functionality in a number of browsers in web mode before the final production release. It's nice to trust GWT and Google when they say that you can write once and run in many browsers, but maybe you're like us and don't take everything at face value!

When producing a GWT application, you always start by creating the GWT default application in order to ensure that the file and directory structure is complete and correct. After producing the GWT default application, you need to take the resulting files and turn them into your own application. You'll do this work in chapter 3, where we first look at what is produced as the default application and then discuss the changes you need to make to produce the first version of the Dashboard. But as we mentioned, your first task is to go through all the Stage 1 steps to create the default directory and code structure.

2.2 Stage 1: Creating a GWT application

Looking into the GWT distribution you downloaded in chapter 1, you should see four command-line applications, all ending with the word *Creator*. Table 2.2 summarizes the functionality of these tools. They help you create the directory and file structure of a GWT application, and they also create a set of default files that go together to make the default application. In this stage, we'll look at each of these creation tools in turn, see how they can be invoked, and discuss the outputs they produce. The result is the GWT default application, which can support internationalization as well as some basic unit tests.

DEFINITION The GWT default application is the application created by the `applicationCreator` tool. It's useful because you can quickly check that the creation tools have executed correctly by running it. Creating your own application means changing the files provided in the default application—something we cover in chapter 3.

Table 2.2 GWT provides a number of creation tools that you can use to quickly develop the default GWT application. These tools are used at various stages in the typical development lifecycle.

Stage	Tool name	Overview
1A	`projectCreator`	GWT provides tight integration with the Eclipse IDE, and if you're going to use this IDE, you need to execute this tool first. If you aren't using Eclipse, you can safely ignore this stage. This tool establishes the necessary files required to enable the directory and file structure to be easily loaded into the Eclipse IDE as an Eclipse project. This essentially means at this stage creating the necessary .project and .classpath files.
1B	`applicationCreator`	This tool performs the following three functions: ▪ Creates the Java package structure in a directory that holds your GWT application. ▪ Creates default HTML and Java files together with a basic module XML file that is used to tie the GWT application together. These created files are the default application that you saw in chapter 1. For most applications, you overwrite all these files in Stage 2 of the lifecycle. ▪ Creates the command-line scripts that can be used to launch the GWT application in hosted mode and to compile it for web mode.

Table 2.2 GWT provides a number of creation tools that you can use to quickly develop the default GWT application. These tools are used at various stages in the typical development lifecycle. *(continued)*

Stage	Tool name	Overview
1C	`i18nCreator`	I18n is an abbreviation for internationalization (i + 18 missing letters + n). This tool performs the following two functions: ▪ Creates a simple properties file containing the key/value pairs that act as constants or messages in an application that uses the GWT i18n capabilities ▪ Creates a new command-line tool specifically for the GWT application being created, which you'll need to use in Stage 1D
1D	*Appl*`-i18n`	This command-line application is created by the `i18nCreator` tool in Stage 1C. Its purpose is to take the properties file containing constants or messages and produce a corresponding Java interface file. The resulting interface file is used in your GWT application code when you need to access i18n constants and/or messages from the properties files. In the compilation process, GWT binds together the interface file with the properties file so that the functionality works seamlessly. Don't worry; we explain this in detail in chapter 15.
1E	`junitCreator`	If you're going to perform unit testing of your GWT application using the JUnit tool, then the `junitCreator` tool creates a suitable directory and file structure. Your unit tests are written into the file created by this tool.
1F	Import to IDE	The final optional step in Stage 1 is importing the directory and code structure created by the creation tools into an IDE. If you performed Stage 1A and ensured that the `-eclipse` flag was used in the other tools, then your GWT application can easily be imported into the Eclipse IDE. If you aren't using Eclipse, then it's still possible to import the directory structure and files into other IDEs.

Although table 2.2 gives a good overview of each of the tools, it doesn't explain which tools are optional and in what order they should be used. When we, the authors, develop GWT applications, we follow the flow given in figure 2.3. This figure helps you understand when and where to use each of the tools in Stage 1 of the development lifecycle.

If you've created GWT applications before, you may wonder where the commands for compiling and executing a project in hosted mode are in figure 2.3. They aren't provided as standard command-line tools because they require some knowledge of your project name and the necessary classpaths. Don't worry, though: GWT doesn't require you to have all this knowledge about classpaths. The

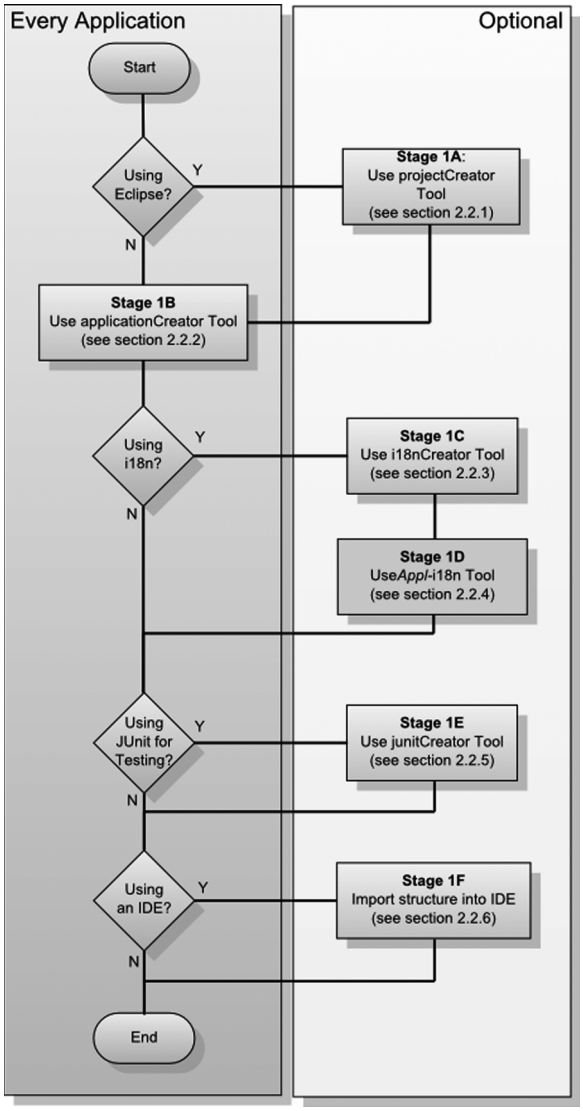

Figure 2.3
How the various GWT creation tools are used together in the development of the structure and files for the GWT default application (the application you have at the end of this chapter)

`applicationCreator` tool creates the necessary commands for compiling and hosted mode.

If you use the creation tools to create an application, then the simplest application that can be created by following the path through figure 2.3 uses only the `applicationCreator` tool. This results in just the plain box-standard GWT default application files being produced. Following all the steps in figure 2.3 results in the

generation of the GWT default application again, but this time it includes the files necessary to support internationalization and unit testing. If you're using Eclipse as the IDE, then you need to add the `-eclipse` flag when you execute these other creation commands to ensure that the required launch files for Eclipse are produced or appropriately updated. Don't forget that if you're using Eclipse, the first tool you need to execute is the `projectCreator` tool so that a number of Eclipse project-specific files are created.

Because the `projectCreator` tool is executed first when you use Eclipse, that's where you start the process of creating the GWT default application (which will, in chapter 3, turn into the Dashboard application). In the rest of this section, you'll follow all the steps in table 2.2 to create a GWT default application that supports internationalization and unit testing. We assume that you're using Eclipse as your IDE; however, if you aren't using Eclipse or an IDE that can import an Eclipse project, you can skip to Stage 1B (section 2.2.2) to create the GWT application.

2.2.1 Creating the project

Using the Eclipse editor as your IDE, you can quickly create the necessary Eclipse project structure and files by using the `projectCreator` command-line tool. Doing this and using the `-eclipse` flag in the other creation tools makes it easy to import your project directly into Eclipse, because it produces certain project files that Eclipse uses to describe classpaths and applications.

If you're using Eclipse, then you should execute the `projectCreator` command as the first step in the development process. Run the following command now to create the DashboardPrj Eclipse project in the DashboardDir directory:

```
projectCreator -eclipse DashboardPrj -out DashboardDir
```

The `projectCreator` command gives the following response if it executes successfully, having produced a source directory (src) and Eclipse-specific .project and .classpath files in the newly created DashboardDir directory:

```
Created directory DashboardDir\src
Created directory DashboardDir\test
Created file DashboardDir\.project
Created file DashboardDir\.classpath
```

The .project and .classpath files will be manipulated by some of the tools you apply in later steps of Stage 1, so if you don't create these files first, those tools will fail. If you don't get this output, then the tool will try to tell you what went wrong—most likely, issues with file permissions on directories at this stage.

NOTE If you forget to run `projectCreator` before `applicationCreator`, then it isn't the end of the world. You can run it afterward; just remember to use the `-ignore` parameter.

The full format for the `projectCreator` command is

```
projectCreator [-ant AntFile]
               [-eclipse ProjName]
               [-out DirName]
               [-overwrite]
               [-ignore]
```

The various flags used in the command are as follows:

- `-ant AntFile`—Requests the tool to produce an Ant build file to compile the source code of the project. The suffix .ant.xml is added to the value provided as the parameter. (Optional.)
- `-eclipse ProjName`—The Eclipse project name.
- `-out DirName`—The directory into which to write output files. (Defaults to current.)
- `-overwrite`—Overwrites any existing files in the output directory. (Optional.)
- `-ignore`—Ignores any existing files in the output directory; doesn't overwrite. (Optional.)

`-ant AntFile` refers to an argument that creates an Ant file. Including such a flag in the arguments to the `projectCreator` tool directs the tool to produce a valid Ant file with actions for compiling, packaging, and cleaning your GWT application (Ant is a popular tool used to ease the process of building and deploying applications). By default, the extension `ant.xml` is appended to the name you provide with the `ant` flag. You use the `ant` flag only if you intend to use the Ant tool to build/deploy your projects. If you're considering using Ant, then listing 2.1 shows the default contents of Dashboard.ant.xml that will be produced if you use the `-ant Dashboard` flag in the creation step.

Listing 2.1 Ant control file output produced when you use the `-ant` flag in the `projectCreator` tool

```
<?xml version="1.0" encoding="utf-8" ?>
<project name="Dashboard" default="compile" basedir=".">
  <description>
    Dashboard build file.  This is used to package up your project
    as a jar, if you want to distribute it. This isn't needed
    for normal operation.
  </description>

  <!-- set classpath -->
  <path id="project.class.path">
    <pathelement path="${java.class.path}/"/>
    <pathelement path=
  "C:/GWT/trunk/build/dist/Windows/gwt-windows-0.0.0/gwt-user.jar"/>
    <!-- Additional dependencies (such as junit) go here -->
  </path>
  <target name="compile" description="Compile src to bin">
    <mkdir dir="bin"/>
    <javac srcdir="src:test" destdir="bin" includes="**" debug="on"
         debuglevel="lines,vars,source" source="1.4">
      <classpath refid="project.class.path"/>
    </javac>
  </target>
  <target name="package" depends="compile" description=
      "Package up the project as a jar">
    <jar destfile="Dashboard.jar">
      <fileset dir="bin">
        <include name="**/*.class"/>
      </fileset>
      <!-- Get everything; source, modules, html files -->
      <fileset dir="src">
        <include name="**"/>
      </fileset>
      <fileset dir="test">
        <include name="**"/>
      </fileset>
    </jar>
  </target>
  <target name="clean">
    <!-- Delete the bin directory tree -->
    <delete file="Dashboard.jar"/>
    <delete>
      <fileset dir="bin" includes="**/*.class"/>
    </delete>
  </target>
  <target name="all" depends="package"/>
</project>
```

Compile target

Package target

Clean target

"All" target

At this point, you've created the basic directory structure and files required for an Eclipse project; this makes it easy to import your project into the Eclipse IDE (or another IDE that can import Eclipse projects). In a little while, we'll show you how to load this project into the Eclipse editor. But first, we should note that you've created only the structure for an Eclipse project and not any GWT-specific files. Creating those GWT default application files is the next step as you rejoin the mainstream development path.

2.2.2 Creating an application

In this step, you'll create the directory structure and files that make up the GWT default application (which in the example will be transformed into the Dashboard application, so you'll set up the files using that name). The structure and files produced in this step are independent from any IDE you may be using. If you're using the Eclipse IDE, then you just created an Eclipse project, and this step will add the GWT default application's files to that project.

The `applicationCreator` tool creates a GWT application that conforms to the GWT expected directory structure (more on this in chapter 9) and also generates the hosted-mode and web-mode scripts. You use this tool to create a new GWT application (don't forget that if you're using the Eclipse IDE, you should run the `projectCreator` tool first, as discussed in section 2.2.1).

To create the GWT default application, you need to execute one of the two command lines shown in table 2.3. There is a non-Eclipse version to use if you aren't using Eclipse as your IDE and an Eclipse version to use if you are (the difference being the inclusion of the `-eclipse DashboardPrj` flag in the Eclipse version—don't worry, we'll explain the syntax and flags shortly).

Table 2.3 Two different versions of the `applicationCreator` tool in action, using the specific code to create the Dashboard application. If you're using the Eclipse version, you should have executed the `projectCreator` tool first.

Version	Command line
Non-Eclipse	`applicationCreator -out DashboardDir org.gwtbook.client.Dashboard`
Eclipse	`applicationCreator -eclipse DashboardPrj -out DashboardDir org.gwtbook.client.Dashboard`

Running either of the command-line versions listed in table 2.3 produces the following output:

```
Created directory DashboardDir\src\org\gwtbook
Created directory DashboardDir\src\org\gwtbook\client
Created directory DashboardDir\src\org\gwtbook\public
Created file DashboardDir\src\org\gwtbook\Dashboard.gwt.xml
Created file DashboardDir\src\org\gwtbook\public\Dashboard.html
Created file DashboardDir\src\org\gwtbook\client\Dashboard.java
Created file DashboardDir\Dashboard-shell.cmd
Created file DashboardDir\Dashboard-compile.cmd
```

The `applicationCreator` tool creates the expected Java package structure under the src directory; the module XML file Dashboard.gwt.xml, which we discuss in detail in chapter 9; the hosted-mode and web-mode command-line tools; and the default application's HTML and Java files.

If you use the Eclipse version, then the project name you used to create the project is added as an extra parameter with the `-eclipse` flag. This tells the command to create an Eclipse launch configuration specifically for this project, and results in the following additional output:

```
Created file DashboardDir\Dashboard.launch
```

If you examine the directory structure of the Dashboard application now, you'll see the structure shown in figure 2.4.

Figure 2.4 Examining the directory structure created by the GWT `applicationCreator` tool—in this case, for the Dashboard application. You can see the `org.gwtbook.client` package structure under the src directory, which is where your Java code will go. The application's basic HTML file is stored under the public directory.

NOTE If you can't use the default directory and file layout for your application (perhaps you have coding standards that are in conflict, or your system setup prevents using it), then you can create the directory structure by hand. But you need to ensure that the command tools' paths are set up correctly and that the GWT compiler can find the source code and public folders, by setting the `source` and `public` attributes in the module XML file (see chapter 9).

As you saw in figure 2.3 (which showed how the GWT creation tools are used together to develop the directory and file structure of a GWT application), the `applicationCreator` tool is all you need to create the structure of the basic GWT default application. To double-check that the `applicationCreator` tool has executed successfully, you can run the default application by executing the `Dashboard-shell` command. Doing so rewards you with the application shown in figure 2.5.

Figure 2.5
The default application created by the GWT creation tools. The files that produce this application are created by executing the `applicationCreator` tool and are usually replaced by your own files when you develop an application (we get to this in chapter 3). The default application is a useful tool to show that the creation tools have worked and that the basic dependencies are correct.

But where did this application come from? If you look in the src/org/gwtbook/client directory of DashboardDir, you'll find the Dashboard.java file, which created the application shown in figure 2.5. The contents of this file are repeated in listing 2.2. (Note that for brevity, we don't show the `import` statements and comment lines that are produced—typically, we won't show these in our code listings unless explicitly necessary.)

Listing 2.2 First view of the GWT code that adds a label and a button to the screen

```
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("");
      }
   });

   RootPanel.get("slot1").add(button);
   RootPanel.get("slot2").add(label);
}
```

Create GWT Button
Create GWT Label
Add ClickListener to Button
Add Button to web page
Add Label to web page

This code, together with the Dasboard.html file in the src/org/gwtbook/public directory, generates the output shown in figure 2.5. We won't go through what these files contain at the moment; we just wanted to let you see that this default application isn't created magically by GWT. To get the look and feel of the Dashboard application (or any other application), you need to replace these files with your functionality—but we leave that until Stage 2 of the development process, in chapter 3.

If you look at the command in full, you see that to create a GWT application, you call `applicationCreator` from the command line using the following template:

```
applicationCreator      [-out DirName]
                        [-eclipse ProjName]
                        [-overwrite]
                        [-ignore]
                        className
```

The options that are available for this command are as follows:

- `-out`—The directory into which to write output files. (Defaults to current.)
- `-eclipse`—Name of the Eclipse project previously used in the `projectCreator` tool execution. (Optional.)
- `-overwrite`—Overwrites any existing files. (Optional.)
- `-ignore`—Ignores any existing files; doesn't overwrite. (Optional.)

The command provides some flexibility over the output, allowing you to change the default directory (if you don't provide one, then the command writes all its output to the current directory). You can also tell the tool to overwrite any existing files or ignore them using the `-overwrite` and `-ignore` flags, respectively. If the application is being created with a view toward importing it into Eclipse, then you should add the `-eclipse` flag as well as the project name you used in section 2.2.1.

NOTE GWT requires the package name for your application's main code to have the subpackage `client` at the end. Not including it will cause the GWT compiler to raise an error.

The final input to the `applicationCreator` command is the class name. It needs to be a fully qualified class name constructed in the particular style shown in figure 2.6. It follows the Sun standard for fully qualified class names in Java but specifically requires that the last nested package name be called `client`. This restriction isn't required by Java, but it's enforced by GWT.

It's useful to follow this GWT-specific format, because in chapter 9, we'll talk about the Java package structure for an application. Later still, we'll introduce

server code under the `org.gwtbook.server` package and code for automatically generating new code under the `org.gwtbook.rebind` package.

Figure 2.6 Breakdown of a GWT fully qualified Java class name for your application. This is standard Java syntax, but GWT requires that your user interface code always be under a subpackage called `client`. (We cover other special subpackage names such as `server` and `rebind` at various points later in this book.)

Up to this point, you've created your project and basic application. If you wanted to, you could already run the `Dashboard-shell` command script to execute the application, as you saw in figure 2.5. For certain applications—those that have no internationalization and/or those for which you have no desire to perform unit testing with JUnit—this is all you need to do. However, in this example you'll add internationalization to the Dashboard; therefore, you'll go through the next two optional stages, 1C and 1D.

2.2.3 Setting up internationalization

Internationalization allows you to display different interfaces depending on the locale from which the application is being viewed (for example, the Dashboard will have different languages in the menus). We discuss internationalization (also known as i18n) in detail in chapter 15; but because this is the next step in Stage 1, you'll set up your application structure now so that i18n is supported. The `i18nCreator` tool lets you set up this structure. It's useful to note that you don't need to set up your i18n approach at this point; you can defer it until later in the development lifecycle if you wish. However, because you know you're going to use it, you'll set it up now.

The rationale behind GWT internationalization is to allow your application to replace specific constants and messages on the UI with a locale-specific version. For example, the Dashboard application will use both English and Swedish a little later. The generation of this functionality is a two-step process; the first step, which uses the `i18nCreator` tool, creates a sample properties file and a new script. The second step involves entering some constants or messages into the properties file and then executing the script generated by this tool. The output from the second stage is a Java interface file that is used in your application. Trust us: It's a lot easier in practice than it may sound!

NOTE Internationalization (i18n) is easy to set up and use in GWT. You just create some properties files and a simple Java interface; at compile time, GWT creates all the necessary Java plumbing code for you.

To add i18n to your existing structure, you need to select the appropriate command line to execute from table 2.4—either the Eclipse or non-Eclipse version—and execute it now.

Table 2.4 The different versions of the `i18nCreator` tool used to create the framework for the Dashboard internationalization

Version	Command line
Non-Eclipse	`i18nCreator -out DashboardDir org.gwtbook.client.DashboardConstants`
Eclipse	`i18nCreator -eclipse DashboardPrj -out DashboardDir org.gwtbook.client.DashboardConstants`

Successful output of executing either of the `i18nCreator` commands is as follows:

```
Created file
   DashboardDir\src\org\gwtbook\client\DashboardConstants.properties
Created file DashboardDir\DashboardConstants-i18n.cmd
```

The tool has created a sample properties file called DashboardConstants.properties that sits where the main code is, and a new command-line tool called `DashboardConstants-i18n` where the rest of the creator commands are. You'll use this new `DashboardConstants-i18n` command-line tool in Stage 1D.

If you use the Eclipse version, then the command also asks the tool to output the necessary Eclipse launch configurations for the second script:

```
Created file DashboardDir\DashboardConstants-i18n.launch
```

You've seen the internationalization tool in action, and you've created a simple structure into which to place the internationalization aspects of the Dashboard application (in this case, you created the structure necessary for constants—in chapter 15, we'll also look at internationalizing messages). Let's take a moment to look at the command line for this tool in detail and see what other arguments you can pass to the tool to alter its behavior. The tool is called using the following template:

```
i18nCreator [-eclipse ProjName]
            [-out DirName]
            [-createMessages]
            [-overwrite]
            [-ignore]
            interfaceName
```

The options that are available to the command are as follows:

- `-eclipse`—Name of the eclipse project. (Optional.)
- `-out`—The directory into which to write output files. (Defaults to current.)
- `-createMessages`—By default, the tool produces an interface that extends the GWT `Constants` interface; if you wish it to extend the GWT `Messages` interface instead, add this flag.
- `-overwrite`—Overwrites any existing files. (Optional.)
- `-ignore`—Ignores any existing files; doesn't overwrite. (Optional.)

By default, the tool produces output files that support the GWT i18n approach for constants, but adding the flag `-createMessages` alters the output to suit the GWT i18n approach for messages instead (we'll explain the differences between constants and messages in chapter 15). You can also direct the tool to overwrite or ignore any existing files in the directory indicated by the value passed to the `-out` flag.

This concludes the first part of setting up internationalization. You've laid the foundation for the i18n of your application; the next stage is to get your i18n approach ready for use in your code—that is, to create the specific locale properties files that contain the locale-specific constants or messages. Remember that for the Dashboard example, you'll be changing the text in the menu system based on the locale. In the next section, we'll focus on creating the properties files that contain the menu text.

2.2.4 Implementing internationalization

In the previous stage, the `i18nCreator` tool created a dummy properties file as well as a new `DashboardConstants-i18n` command-line application; but you can't yet use these files in your application, because they contain no data. In Stage 1D, you create the link between the properties file and your application code. To create that link, you must create key/value pairs for constants in the properties file and then execute the `DashboardConstants-i18n` command-line application. This tool takes that properties file and produces a Java interface class that contains a method for each key. It's these methods in the Java interface that you use in your application.

This step of Stage 1 may be performed more than once, possibly during the later stages of development. Each time new constants and/or messages are added to your properties file, this stage should be executed to ensure that the Java interface file is up to date. If you executed the `DashboardConstants-i18n` command line now, you'd get a simple interface file reflecting the default properties file.

When you update your i18n properties file in chapter 3, you'll run the `DashboardConstants-18n` command line again to make sure you take advantage of and gain access to the new constants.

That concludes the creation of internationalization. In the final step of using the creation tools (from figure 2.3), you'll set the foundations for unit testing. Although this step is optional, we feel that unit testing is an important part of development. It isn't necessary to perform this step at this stage; you can jump to section 2.2.6 if you wish to start getting your framework into an IDE or to chapter 3 if you want to start building the Dashboard functionality without an IDE.

2.2.5 Creating unit test cases

JUnit is a powerful approach to testing, and it's beneficial that GWT includes a simple way of integrating unit tests into your development approach. Undoubtedly, JUnit deserves—and has—a wealth of books written about it, and we won't attempt to cover using JUnit here. If you're interested in knowing more about JUnit, we recommend *JUnit in Action* by Vincent Massol and Ted Husted. (In chapter 16, we look at GWT testing in more detail.)

NOTE GWT makes writing and creating JUnit tests for your code a painless (maybe enjoyable is going too far) process.

Before you can begin creating a JUnit test-case structure, you need to have the JUnit JAR file somewhere on your development system that you can refer to (you can download it from http://www.junit.org). To add JUnit tests to the Dashboard example, execute one of the two versions listed in table 2.5 (for simplicity, we're assuming that the JUnit JAR is stored in the root of the C:\ directory; if you have it somewhere else, then you should replace the argument to the first parameter with your location).

Table 2.5 The different versions of the `junitCreator` tool used to create the framework for the Dashboard JUnit testing.

Version	Command Line
Non-Eclipse	`junitCreator -junit c:\junit.jar -module org.gwtbook.Dashboard -out DashboardDir org.gwtbook.client.test.DashboardTest`
Eclipse	`junitCreator -junit c:\junit.jar -eclipse DashboardPrj -module org.gwtbook.Dashboard -out DashboardDir org.gwtbook.client.test.DashboardTest`

Running either command creates a suitable directory and file structure ready to accept the unit-test code for any unit testing you may wish to perform on the Dashboard example (the Eclipse version provides more files for easy integration with Eclipse). You ask for the test class to be called `DashboardTest` and to be in the package `org.gwtbook.client.test`. The script creates a new directory called test under DashboardDir and stores your generated test-class files under that. Because you're creating the tests in the same package as the code, there is no need for the `junitCreator` command to alter the Dashboard module.

Successful output of both versions of the `junitCreator` command is as follows:

```
Created directory DashboaradDir\test\org\gwtbook\client\test
Created file DashboardDir\test\org\gwtbook\client\test\DashboardTest.java
Created file DashboardDir\DashboardTest-hosted.cmd
Created file DashboardDir\DashboardTest-web.cmd
```

If you're using the Eclipse version, then the following two lines are appended to the output, indicating the creation of the associated hosted-mode and web-mode launch scripts for Eclipse:

```
Created file DashboardDir\DashboardTest-hosted.launch
Created file DashboardDir\DashboardTest-web.launch
```

The `junitCreator` tool creates the necessary classes with stubs in them; places the necessary links in the GWT application module; and, if you're using Eclipse, creates the appropriate launch configurations. The full template for the command is

```
junitCreator      -junit PathToJUnitJarFile
                  [-eclipse ProjName]
                  [-module ModName]
                  [-out DirName]
                  [-ignore]
                  [-overwrite]
                  className
```

The options that are available to the command are as follows:

- `-junit`—Path to the JUnit libraries.
- `-eclipse`—Name of the Eclipse project. (Optional.)
- `-module`—The GWT module of the application you wish to test.
- `-out`—The directory into which to write output files. (Defaults to current.)
- `-overwrite`—Overwrites any existing files. (Optional.)
- `-ignore`—Ignores any existing files; doesn't overwrite. (Optional.)

The flags `-eclipse`, `-out`, `-overwrite`, and `-ignore` are the same as those discussed for the previous two tools. The `-junit` flag is required, and its value must point to the installation of the JUnit classes in your system. The `-module` flag must indicate the GWT module you want to test.

Running the `junitCreator` tool won't, unfortunately, create unit tests for you; it does, however, manipulate the necessary files and create a sensible directory structure, shown in figure 2.7. In addition, it creates a simple Java class file in which you can place your unit tests.

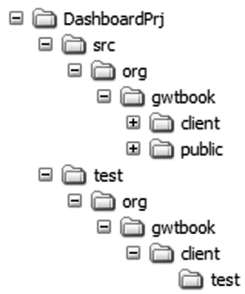

Figure 2.7 The directory structure that exists after you execute the GWT `junitCreator` tool. Notice that the structure for the application you saw in figure 2.4 is still intact. The new structure for testing sits under the test directory.

We'll look at using JUnit testing in more detail when we get to chapter 16.

Let's take a second to recap where you are with respect to creating the GWT default application. You've created the directory and code structure for the GWT default application, and you've added support for internationalization. If you took the last step, you've also incorporated unit testing. These steps are common to all GWT applications you'll build; only by altering and adding to the default application code do you begin to create your own application.

Before you start building the Dashboard application in the next chapter, let's look at the final Stage 1 application development task: importing your project into an IDE. We recommend this optional step because it makes code development/debugging much easier and quicker.

2.2.6 Importing into your IDE

We're convinced that a great benefit of GWT is the ability to use a familiar development environment in the development of your web applications (the GWT team members often say on the forums that they chose Java not because they're Java addicts but because at this moment in time it has great tool support).

Right now, GWT integrates easily into Eclipse, and other IDEs are rapidly delivering easier integration approaches. There are even approaches to build GUI development tools for GWT in a similar manner to Matisse (the NetBeans IDE GUI builder).

In this section, we'll demonstrate the use of Eclipse as an IDE and also the generic steps you can take to use another IDE if you created your application following the instructions given in the early part of this chapter.

Importing into Eclipse

Importing a project into Eclipse after you create it using the project and application creator tools is simple and quick. In the Eclipse Package Explorer window, right-click, and select the Import option. Figure 2.8 shows this step, together with the next step of choosing the Existing Projects into Workspace option under the General folder.

Click the Next button, and then click the Browse button next to the Select Root Directory option. Navigate the filesystem to find DashboardDir, and select it; see figure 2.9. Click OK.

At this point, you can decide where you want Eclipse to store the project file—in the Import Projects dialog box, if you select the Copy Projects into Workspace

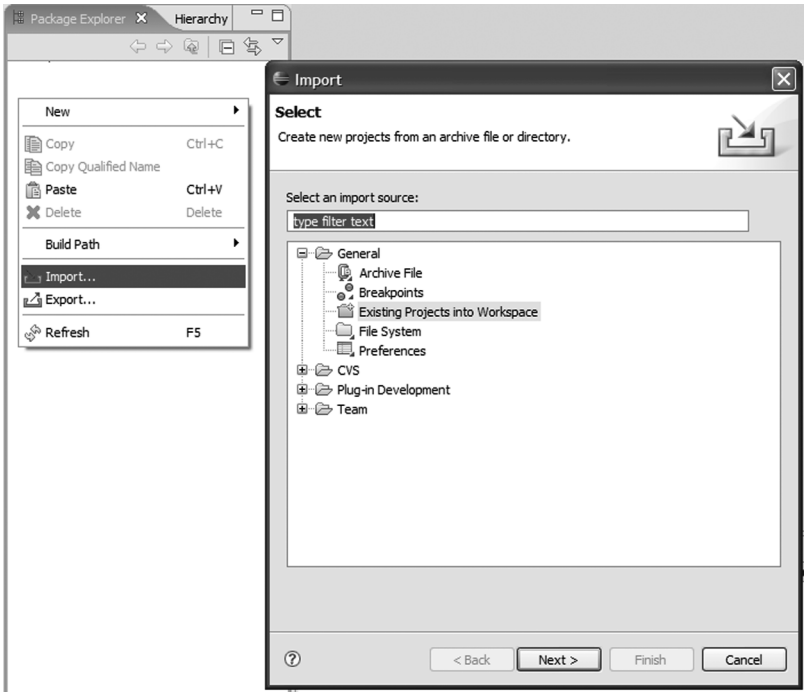

Figure 2.8 To import the newly created Dashboard project into the Eclipse IDE for development of the application, right-click the Eclipse Package Explorer window and select Import.

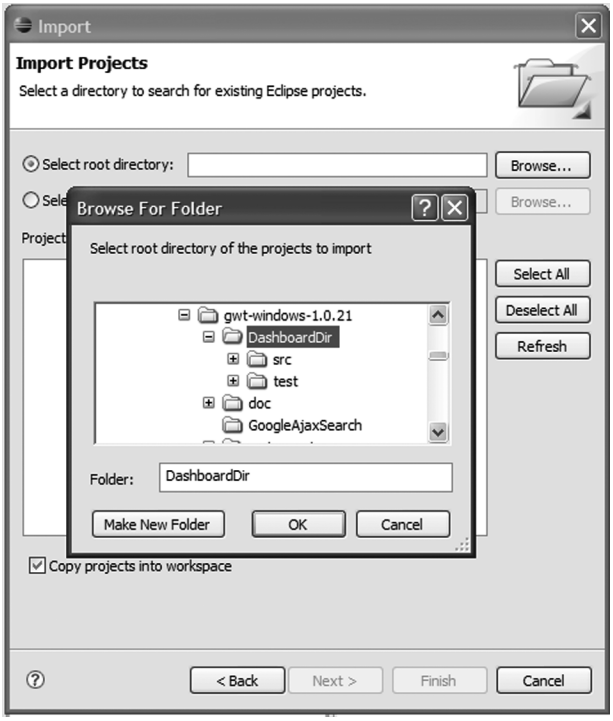

Figure 2.9
The second stage of importing the Dashboard project into Eclipse: finding the project that you're importing in your directory structure.

check box, Eclipse copies all the project files into its workspace directory. If you don't select this option, Eclipse leaves the files where they are. It's important to remember which way you choose, because that dictates where you'll need to pick up the GWT-generated files from later for web deployment (GWT will place them relative to the source files).

Regardless of which choice you make about where to store the project files, now you should click the Finish button to import the project. A short while later, you'll see a view similar to that shown in figure 2.10. The project is now loaded into Eclipse.

Eclipse isn't the only IDE you can use to develop GWT projects, although it's the easiest to integrate straight out of the box using the tools provided in GWT. In case you choose not to use the `-eclipse` flag with the creation tools, or if you wish to import your files into a different IDE, we'll look at the steps to do that next.

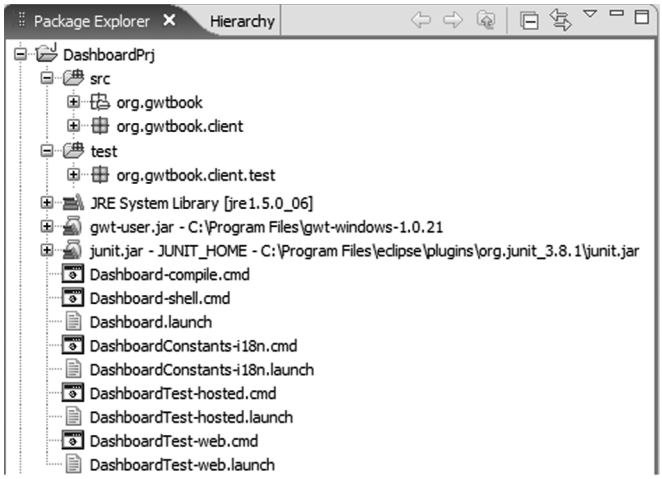

Figure 2.10 The GWT Dashboard application loaded into the Eclipse Package Explorer window, showing the directory structure, the Java and HTML files together with the associated module XML file, and the range of command-line applications and Eclipse launch configurations generated by the GWT creation tools.

Using other IDEs

With GWT, you aren't restricted to using the Eclipse IDE. If you wish, you can use a text editor and command-line execution of tools. If you do use an IDE, then you can import the structure you've just created.

You've already seen how the `applicationCreator` tool creates the correct file structure for a GWT application, so you're up and running from that point. Follow these steps if you're using an IDE for which there is no existing GWT support:

1. Create the application using `applicationCreator` (and both `i18nCreator` and `junitCreator` if you want that functionality).
2. Import the code into your IDE, preserving the file structure just produced.
3. Add the gwt-user.jar file to your IDE's classpath and any path required for auto-completion to operate correctly.
4. Hook up your IDE to execute the `application-shell` and `application-compile` scripts.
5. For debugging, hook your IDE debugger up to port 9000, which is the port on which hosted mode allows debugging connections.

Using Ant scripts will significantly increase the ease of development and testing, and you can find some references to scripts that others have built on the GWT forum.

IDEs such as NetBeans can import Eclipse projects, so this may be a smoother way to ensure that all the configuration needed for your application is imported.

The alternative is to use an IDE that has a GWT wizard built for it, such as IntelliJ, which creates the necessary files and configuration for instant use in that IDE. Finally, some extensions to Eclipse exist, such as Googlipse and GWTDesigner; they allow you to develop GWT applications graphically and thus create the necessary configurations for you at project creation time.

One way or another, you've arrived at the point where your project structure is in place and you have the ability to start building your new GWT application, either in a text editor or in an IDE. You've executed the default application to make sure everything has been created correctly, and now you're ready to change that default to your own application.

2.3 Summary

This chapter has covered the first step in developing a GWT application created using the default GWT creation tools. You now have the directory and file structure for the default GWT application, which forms the basis for the Dashboard example you'll build on throughout the rest of the book.

You may think we've used a lot of pages for what is effectively the GWT "Hello World" example, but the GWT structure isn't as simple as typing four lines of code and saying, "there you are." However, you've done the hard part; continuing to develop your application from this point is much more like writing a few lines of code and executing it. Although it may have seemed like a long path to get to this point, you've seen that with a bit of practice and knowing what you're doing, completing Stage 1 takes only a few minutes.

The key benefit of using the GWT creation tools is that they generate the directory structure and file contents in such a way that they correctly tie together. You can easily check that the structure is correct by executing the default application. This puts you in a safe position to begin replacing the default files with the files needed for your own application, and that means heading on to chapter 3.

AJAX / JAVA

GWT IN ACTION

Robert Hanson and Adam Tacy

With Ajax you can create great interfaces, but testing and team development are difficult and poor tool support adds to the hassle. Not so with GWT. The Google Web Toolkit is an open source Java framework that turns Java code into browser-neutral JavaScript and HTML. Along the way you get the power and tools of the full Java platform. GWT includes a complete widget library and good RPC support for full application development.

GWT in Action is a clearly written, comprehensive tutorial on the Google Web Toolkit. It covers the GWT development cycle, from setting up your programming environment, to building the application, then deploying it to the web server. By following a running example, you'll master widgets, panels, and events: the basic building blocks of a GWT application. Then you'll learn how to reuse existing JavaScript components, communicate via GWT-RPC, and interoperate with JSON. Along the way, you'll pick up great techniques for internationalization, testing, and deployment. This book assumes a working knowledge of Java.

What's Inside:

- Build a full-scale GWT project
- Master GWT-RPC and JSON interaction
- Create flexible and scalable applications
- Try the GWT Dashboard live at www.manning.com/GWTinAction

Robert Hanson is a US-based Internet engineer and creator of the popular open source GWT Widget Library. **Adam Tacy** is a consultant working for WM-Data in Sweden and a contributor to the GWT Widget Library.

For more information, code samples, and to purchase an ebook visit www.manning.com/GWTinAction

"... impressive quality and thoroughness. Wonderful!"

—Bernard Farrell, Software Architect, Kronos Inc.

"How to 'think in GWT.' The code: concise, efficient, thorough, and plentiful."

—Scott Stirling
Senior Consultant at USi
an AT&T Company

"Perfect for Java developers struggling with JavaScript."

—Carlo Gottiglieri
Java Developer, Sytel-Reply

"A real nitty-gritty tutorial on the rich features of GWT."

—Andrew Grothe
COO Eliptic Webwise, Inc.

ISBN-10: 1-933988-23-1
ISBN-13: 978-1-933988-23-8

$49.99 / Can $64.99