Google Web Toolkit

Taking the pain out of Ajax

Ed Burnette

The Pragmatic Bookshelf

Raleigh, North Carolina Dallas, Texas

Useful Friday Links

- Source code from this book and other resources.
- Free updates to this PDF
- Errata and suggestions. To report an erratum on a page, click the link in the footer.

To see what we're up to, please visit us at

http://www.pragmaticprogrammer.com

Produced in the United States of America.

Lovingly created by gerbil #17 on 2006-11-28

Contents

Chapter 1

Introduction

The Google Web Toolkit (GWT) was unveiled to an unsuspecting public on May 18th, 2006 at the annual JavaOne conference in San Francisco. The premise behind GWT is simple: make Ajax[1] development easier by hiding browser incompatibilities from the programmer and allowing the developer to work in a familiar Java development environment.

The announcement was one of the highlights of the conference and interest continues to grow. Developers have used GWT technology in everything from games to mortgage calculators. The gwtPowered community site lists over 130 examples, articles, widgets, and other resources. Why has the Google Web Toolkit become such a hot topic?

If you've ever written a non-trivial Ajax application before, then I'm sure you can sympathize with the need to make the process easier. If not, then a little background is in order.

1.1 Life before GWT

Dynamic web applications are typically written in several different languages across two or more tiers. On the client side (the part running in the browser), you have HTML markup of course,

[1] The term *Ajax* was famously coined in February 2005 by Jesse James Garrett. Originally it was an acronym for Asynchronous Javascript And Xml. The technology has actually been around for a few years—for example it was used in Outlook Web Access in 2000—but didn't get much attention until Google popularized it with applications such as GMail and Google Maps.

plus you have some logic written in JavaScript to perform tasks like client-side validation and manipulation of the HTML document object model (DOM).

Unfortunately, slight differences in the JavaScript language between browsers, along with major differences in the DOM, make coding these clients a bit like walking through a mine field. Various libraries such as Dojo and Prototype were created to smooth out the rough edges but JavaScript/browser programming is still something of a black art. Some developers have abandoned HTML and JavaScript altogether in favor of Flash or other alternatives.

On the server side you have a web server tier and optionally a data tier. Commodity web servers such as Apache, Tomcat, Lighttpd, and IIS host your application logic, which is written in Java, PHP, Ruby, C#, Klingon (ok, maybe not Klingon), or other languages. JavaScript is not used on the server except by a few masochists. Data services are provided by databases such as MySQL, Oracle, Sql Server, and so forth. Often the actual database is hidden behind an Object/Relational (O/R) layer such as Hibernate.

Although this architecture is very flexible, its complexity makes it hard to manage. Frameworks such as Ruby on Rails grew up to reduce the complexity on the server side. Other frameworks like Java Server Faces (JSF) and Microsoft Atlas try to standardize and provide built-in implementations of client-side operations such as validation. However, substantial dynamic web applications are still much harder to write than the traditional desktop applications they're supposed to replace.

JavaScript is a red-headed step-child of a language that first appeared in the Netscape browser in 1995 as a way to script Java applets. It was adopted by Microsoft in the following year, becoming the de-facto standard for scripting inside the browser. Despite having Java in its name, it bears little resemblance to that language. The closest thing to JavaScript would be... well actually there's nothing quite like JavaScript. Some would count that as a good thing.

1.2 What GWT does for you

Google Web Toolkit unifies client and server code into a single application written in one language: Java. This has many advantages. For one thing, far more developers know Java than JavaScript or Flash. Another reason is that Java is blessed with an abundance of developer tools such as Eclipse, NetBeans, and IDEA. GWT lets you create a web application in much the same way as you would create a Swing application—creating visual components, setting up event handlers, debugging, and so forth—all within a familiar IDE.

By standardizing on one language you can share code on the client and server. For example you can run the same validation code—once on the client for immediate feedback, and once on the server for maximum security. You can even move code between tiers as you refactor your application to adapt to changing requirements.

GWT also abstracts the browser's DOM, hiding differences between browsers behind easy to extend object-oriented UI patterns. This helps make your code portable over all supported browsers.

If this sounds too good to be true, well, it is a little bit. You still have to be careful not to introduce browser-specific dependencies. As tech guru Joel Spolsky likes to say, all abstractions are leaky. Occasionally you may have to delve into CSS/DOM/JavaScript to address browser quirks in non-trivial programs. But with GWT this is the exception rather than the rule.

1.3 About this book

This book provides you with a thorough introduction to the Google Web Toolkit. From installation, through your first application, to UI

components and Remote Procedure calls, you'll learn the ins and outs of the framework. Some knowledge of Java programming and HTML is assumed, but you don't have to be an expert in web programming.

History

This section lists all the updates made to the first edition of this book.

- P1.1 (27nov2006): Updated for GWT 1.2.22. Added I18N chapter.
- P1 (11sep2006): Updated for GWT 1.1.10.
- P0 (23aug2006): Original for GWT 1.1.0.

Ok, enough talk—let's get started with your first GWT application!

Chapter 2

Getting Started

Getting started developing with Google Web Toolkit is easy. In this chapter I'll show you how to set up a few things, and then you can jump right in and create a working application using the scaffolding GWT provides.

2.1 Supported platforms

Development of GWT applications is supported on Windows, Linux, and MacOSX (as of GWT 1.2). All the examples in this book were done on Windows.

GWT applications may be deployed in web servers running on any operating system, and viewed on any modern desktop browser (IE6, IE7, Firefox, Opera, and so on).

2.2 Installing

Before you start coding you need to install Java, an IDE, and GWT itself.

Java 1.4.2+

First you need a copy of Java. Although GWT works with Java 1.4.2 and newer, you might as well get the latest Sun JDK 5.0 or 6.0 update from the Sun download site To verify you have the right version, run this command from your shell window:

```
C:\> java -version
java version "1.5.0_07"
```

```
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_07-b03)
Java HotSpot(TM) Client VM (build 1.5.0_07-b03, mixed mode, sharing)
```

Sun has a taken a page from Microsoft's playbook and bundled their NetBeans IDE in the 5.0 JDK. However this is sometimes an older version of the JDK, and this kind of bundling should be discouraged anyway. Fortunately, you can still get just the plain JDK without NetBeans and save yourself 70MB of extra downloading at the same time. Unless you really want NetBeans of course.

Eclipse

Second, you need a copy of the Eclipse IDE. While you can use other Java IDEs such as NetBeans or IDEA, the Google developers use Eclipse and so do I, so that's what I'll be using for the remainder of this book. Go to the Eclipse downloads page, pick 3.2 (or later), and then get the Eclipse package for your platform (Windows, Linux, Mac, etc.). You can either get the full SDK (the one that has all the sources and programmer documentation), or for a smaller download you can just get the Platform Runtime Binary plus the JDT Runtime Binary.

For an easier Eclipse download experience, you could try the Eclipse on demand site, sponsored by Yoxos, or Easy Eclipse, sponsored by nexB.

GWT

Next, download the Google Web Toolkit SDK (1.2.22 or later). Unzip the Google Web Toolkit onto your machine. No special install is needed. Now you're ready to create your first project.

2.3 Create scaffolding

At a command prompt, run these commands (substituting the appropriate paths for your system):

```
C:\> mkdir c:\gwt-projects\MyProject
```

```
C:\> cd c:\gwt-projects\MyProject

C:\gwt-projects\MyProject> projectCreator -eclipse MyProject
Created directory C:\gwt-projects\MyProject\src
Created file C:\gwt-projects\MyProject\.project
Created file C:\gwt-projects\MyProject\.classpath

C:\gwt-projects\MyProject> applicationCreator -eclipse MyProject\
   com.xyz.client.MyApp
Created directory C:\gwt-projects\MyProject\src\com\xyz
Created directory C:\gwt-projects\MyProject\src\com\xyz\client
Created directory C:\gwt-projects\MyProject\src\com\xyz\public
Created file C:\gwt-projects\MyProject\src\com\xyz\MyApp.gwt.xml
Created file C:\gwt-projects\MyProject\src\com\xyz\public\MyApp.html
Created file C:\gwt-projects\MyProject\src\com\xyz\client\MyApp.java
Created file C:\gwt-projects\MyProject\MyApp.launch
Created file C:\gwt-projects\MyProject\MyApp-shell.cmd
Created file C:\gwt-projects\MyProject\MyApp-compile.cmd
```

The projectCreator and applicationCreator commands are two shell scripts that are supplied as part of GWT, so you'll need to specify the path to them or add the GWT directory to your system PATH variable. projectCreator builds the scaffolding for a generic GWT project, and applicationCreator adds a simple GWT application that you can build upon. MyProject, MyApp, and com.xyz are just example names; you can use anything you want. However the .client part of the package name is important; we'll come back to that later.

2.4 Running and debugging

At this point you're ready to try out the application.

Figure 2.1: Hello world GWT application

Running outside Eclipse

First, let's run the app outside of the IDE by using one of the handy shell scripts that the scaffolding provided:

```
C:\gwt-projects\MyProject> MyApp-shell
```

If everything is working correctly two windows will appear: The GWT development shell (this is kind of like a console window) and a web browser window. See Figure 2.1

Verify the application works by clicking the `Click me` button—the text *Hello World!* will appear. Congratulations, you've just created and run your first GWT application.

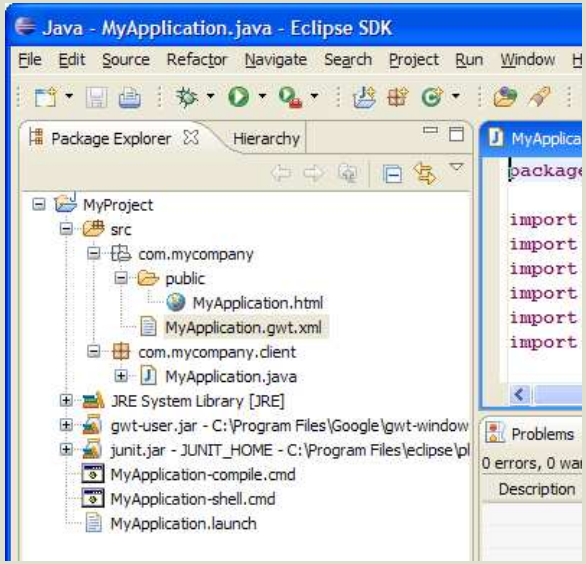

Figure 2.2: GWT project in Eclipse

Running inside Eclipse

Now close the two GWT windows, start up Eclipse, and import this project into your workspace (File → Import → Existing Projects Into Workspace). The project will build, and if all is successful you will end up with something like Figure 2.2 .

Now select Run → Debug..., and click on the launch configuration titled MyApp (under Java Application). Then click on Debug. The two GWT windows should appear again, just like in Figure 2.1, on the preceding page

Debugging

Ok, now for the neat part. Leave the application running and switch back to the Eclipse window. Set a breakpoint in the onClick() method

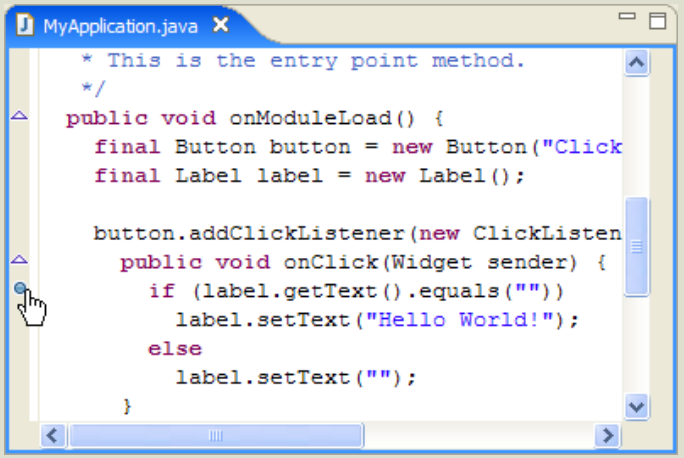

Figure 2.3: Setting a breakpoint

in MyApp.java by double-clicking the gutter area next to the line (see Figure 2.3).

Then switch to the application window and click the `Click me` button again. Eclipse will stop at the line in your Java code where you put the breakpoint. You can single step, examine variables, and so forth.

Hmm, that's Java code, yet you're writing an Ajax application that will eventually be deployed in pure JavaScript. All the power of your Java development environment—Eclipse, the debugger, refactoring, source management, and so on—is suddenly available in the Ajax world. Can you begin to see the potential of this technology? In the next chapter we'll take a look behind the curtain to reveal how the magic is done.

Chapter 3

Hosted vs. Web Mode

In the previous chapter, when you invoked a GWT application you were using what Google calls *hosted mode*. Hosted mode is only used during development. When in production, your application will be running in *web mode*. Before going any further in using GWT you need to understand the difference between the two. Note that as of this writing, hosted mode is only available on Windows and Linux.

3.1 Hosted mode

Think of hosted mode as training wheels for your GWT application. It's a hybrid development environment unique to GWT that lets your code run as real Java code, but still inside a browser. Execution in hosted mode is controlled by the Google Web Toolkit *development shell* (the background window in Figure 2.1, on page 8).

The development shell is actually an Eclipse Rich Client application, consisting of the shell console, a tomcat server, and one or more hosted browsers.

The *hosted browser* (the front window in Figure 2.1) has two connections back to the development shell. One is just a regular http connection to get the web pages, .css files, images, and other resources. All these are handled by the embedded Tomcat server using a servlet called com.google.gwt.dev.shell.GWTShellServlet.

The second connection is a back-door that intercepts all interactions inside the hosted browser and routes them not to JavaScript but to Java code in the shell. That Java code in turn calls your real client Java code, which was compiled to bytecode by your IDE. The exact

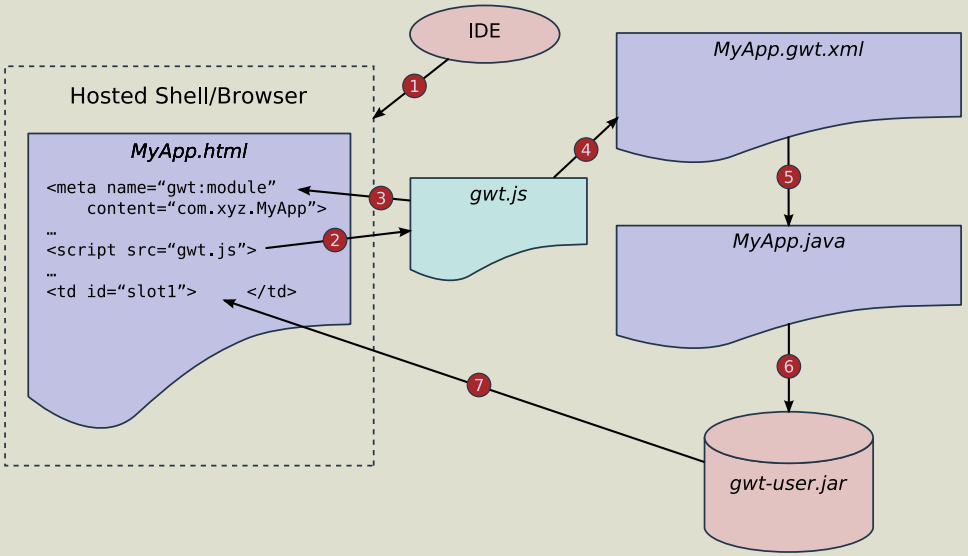

Figure 3.1: How a GWT page is loaded in hosted mode

details of how this is done are hidden in the shell code, which is not open source.

Figure 3.1 shows how a page is loaded in hosted mode:

1. The Shell program opens a hosted browser window, which loads MyApp.html.
2. MyApp.html loads gwt.js with a *<script>* tag.
3. gwt.js scans MyApp.html and parses out the *<meta name='gwt-module'>* to get the module name.
4. GWT reads the module file (MyApp.gwt.xml) to find the name of the EntryPoint class (MyApp).
5. The MyApp class is instantiated and its onModuleLoad() method

Joe Asks...

Is GWT open source?

The short answer is yes. All the libraries and JavaScript you need to deploy your code are covered under the Apache license. Two development time pieces – the development shell and the Java to JavaScript compiler – are not open source at this time. But according to GWT tech leader Bruce Johnson, even these parts may be opened in the future.

is called. Your application begins.

6. Your application code makes calls into the GWT user library (gwt-user.jar), which is also Java code.

7. Code in gwt-user.jar manipulates the hosted browser's DOM to add UI components to the web page, and redirects all browser events back to the Java application code using special hooks in the browser.

Because real Java code is running, you can use Java tools like the Eclipse debugger, findbugs, pmd, JUnit, and so forth. It's almost as if you were developing a rich client program with Swing or SWT because it's Java end-to-end.

Once you've debugged and unit tested your code the next step is to compile it into a form that can be run inside a regular browser (not one that has been hijacked by the development shell). That's where web mode comes in.

3.2 Web mode

When you click the Compile/Browse button in the hosted browser, the GWT compiler translates your .client package into JavaScript and opens a normal web browser to view the application. At this point pages are still served by the shell's Tomcat instance, but they could just as easily come from the file system or a normal web server.

Another way to invoke the GWT compiler is with the shell script provided by the scaffolding (MyApp-compile). You could also write an Ant script to do it if you prefer. For example to maintain the gwtpowered.org site I have an ant script that does the compile and then

copies everything to my hosting provider. You can find the source at http://code.google.com/p/gwtpowered.

However you invoke it, the GWT compiler combines your code with a JavaScript version of the GWT API (the equivalent of gwt-user.jar) in one JavaScript file. This code and several supporting files are placed in the www directory inside your project. Everything from your public directory is copied there as well. The table below explains what all the files do:

The GWT compiler actually creates several different browser-specific versions of compiled JavaScript code, for several different classes of browsers (IE, Firefox, etc.). Google calls these cache files. The filenames use long unguessable hex codes. Only one of these is loaded, depending on your browser.

The .cache file is cached by the client to improve load time for future visits. When the app is modified and recompiled the .cache file name will be different so the browser will download it again. Any old .cache files will be ignored.

Future versions of GWT might create more or fewer of these cache files. As support for new browsers is added to GWT, all you have to do is recompile your application with the newer compiler to support them

Filename	**Description**
long-hex-name.cache.html	Compiled JavaScript
long-hex-name.cache.xml	Implementation defined
module-name.nocache.html	Cache file selection
gwt.js	Common GWT bootstrap code
history.html	Contents of history IFrame
MyApp.html	Main page, copied from public
tree*.gif	+/- images used by the Tree widget

The flow of execution during a page load in web mode (see Figure 3.2, on the following page) is a bit different than in hosted mode. Here's a breakdown of what happens:

1. The web browser loads MyApp.html.
2. MyApp.html loads gwt.js with a <*script*> tag.
3. gwt.js scans MyApp.html and parses out the <*meta name='gwt-module'*> to get the module name.
4. gwt.js modifies the page to include an <*iframe*> that causes the source file *module-name*.nocache.html to be loaded.

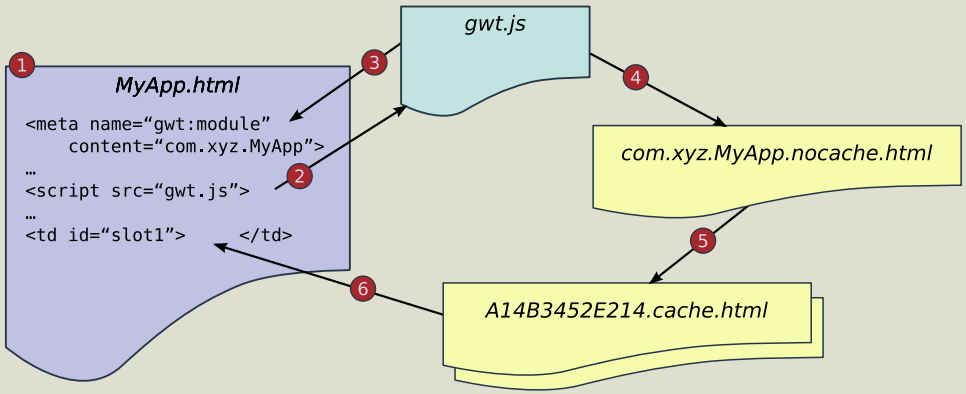

Figure 3.2: How a GWT page is loaded in web mode

5. JavaScript inside the file *module-name*.nocache.html looks at the browser's userAgent field to determine what kind of browser the user is running (IE6, Mozilla, Opera, etc.). Then it selects the correct code (cache file) for that browser type and redirects the <*iframe*> there.

6. The JavaScript equivalent of your onModuleLoad() method is executed, and the rest of your application goes from there. Manipulations to the browser DOM are performed with ordinary dynamic HTML calls in the compiled JavaScript.

3.3 Obfuscation

By default, the GWT Java to JavaScript compiler will produce obfuscated output. Code that has been *obfuscated* is smaller than human-readable code, and is harder to reverse-engineer. It's very difficult to debug, though. Should you ever need to debug the JavaScript that GWT produces, you can turn off obfuscation with command line

parameters on the GWT compiler (for example as arguments to the MyApp-compile.cmd script). Use the -style pretty option to produce good looking output with readable names and indentation. To see full Java types as part of the names, use the -style detailed option instead.

3.4 Deployment

All the examples up to now have been dependent on the hosted shell's Tomcat server to serve up all the application's files. However in web mode they can be delivered by any web server or even (for testing) the local file system. To try this out, copy the entire www directory to another location in your file system and bring up a regular browser on your starting HTML page. The application should work exactly the same as before.[1]

For simple programs like this there is no interaction with the server because none of your code is running there. We'll see some more complicated programs in Chapter 5, *Remote Procedure Calls*, on page 32 that do require more than copying a directory, but for now let's see what fun we can have with GWT's user interface components.

[1] History doesn't work on the local file system in IE6. But why are you using IE anyway?

Chapter 4

User Interface

One thing you'll notice when developing a GWT application is that it's much like developing a desktop application with Swing, SWT, or even Visual Basic. You create controls such as buttons, lists, and tables, you add them to parents, and you interact with them via listeners. You lay them out in a certain arrangement and try to make it look nice at any font size and screen resolution. The main difference is that your GWT app will appear in a web browser, so there has to be an HTML page involved somewhere.

Traditional web applications are structured as a series of HTML pages with some kind of navigation between them. For example you might have an inventory page, an ordering page, and a confirmation page. In a GWT application, however, you stay on one page the whole time. Instead of changing web pages, you change the contents of the one page to reflect the current state. For example you might have three different panels for inventory, ordering, and confirmation within the page, and show only one at any given time. This gives the user a smoother, more responsive experience compared to the old way.

4.1 Tying into HTML

If you look in your project under src/com/xyz/public you'll find a file called MyApp.html. This is the canvas in which the GWT user interface will be hosted.

Every GWT application lives inside a single HTML page. It could be a static page like this one, or a page generated with a server-

side framework like JSP, Struts, Ruby on Rails, etc.. To keep things simple we'll just look at static pages for the rest of this book.

The fact that MyApp.html is in the public directory means it will be copied verbatim into the final deployment area on the server (see Section 3.2, *Web mode*, on page 13). If you have any images, style sheets, etc., then they need to go somewhere in this same directory.

Near the top of the HTML page is a required meta tag that associates this page with a GWT *module*.

Download MyProject/src/com/xyz/public/MyApp.html

```
<meta name='gwt:module' content='com.xyz.MyApp'>
```

A GWT module is a collection of client-side application code and resources you supply. The module named `com.mycompany.MyApp` is defined in the module file src/com/mycompany/MyApp.gwt.xml.

Download MyProject/src/com/xyz/MyApp.gwt.xml

```
<module>

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

  <!-- Specify the app entry point class.                   -->
  <entry-point class='com.xyz.client.MyApp' />

</module>
```

Here you can see the name of your Java class. Logically, when the HTML page is loaded, GWT looks up the meta tag, reads the xml file to get the class name, and starts calling code in the EntryPoint class. As of GWT 1.1 you can also have GWT inject .css files and other resources with module directives.