

## A P P E N D I X



# Building ToCollege.net

In this appendix, we're going to cover everything you need to know in order to get ToCollege.net set up on your machine. To do this, we're going to rely on a number of popular open source tools. We won't be able to go over the setup for all of these tools exhaustively, but I'll try to do my best to give you everything you need to get up and running.

First, we're going to go over the general tools that we'll want to have in our environment. We'll target a Windows and Eclipse environment, though nothing that we're doing should prevent you from achieving the same results on another platform or with another IDE. Indeed, ToCollege.net is built and deployed on Linux servers and developed on both Mac and Windows.

After that, we'll tackle the ToCollege.net codebase. I'll show you how to get all the code for the site and how to get it compiling. Alright, let's get started.

## General Tools

In this section, we're going to look at the basic tools that we'll want to have at our disposal for developing ToCollege.net.

### Eclipse Tools

Before we begin working with the ToCollege.net code, let's update Eclipse. If you don't have it installed already, you'll want to begin by installing Eclipse 3.3, available from <http://www.eclipse.org/downloads/>. I would recommend the Java EE package, because it comes with some nice-to-have extra tools.

---

**Note** If you'd prefer not to use Eclipse, that's not a problem. Every modern IDE will be able to get you started, and if you want to use vi and compile on the command line, that will work too. For you IDE developers, you will still need a tool that helps you configure your classpath from a Maven pom.xml file. For NetBeans, a plug-in is available at <http://mevenide.codehaus.org/m2-site/mevenide2-netbeans/installation.html>, and for IntelliJ, there is one here: <http://plugins.intellij.net/plugin/?id=1166>.

---

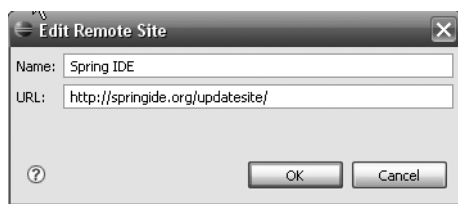
Once you have Eclipse installed, we're going to add some useful plug-ins. To install plug-ins for Eclipse, you'll need to do the following:

1. Click Help ► Software update ► Find and Install.
2. Click Search For New Features.
3. Select New Remote Site.

We'll use the Edit Remote Site dialog three times to install our three desired plug-ins. For each remote site, we'll need to enter a name and the update site where we want Eclipse to go look for plug-ins. First, we'll install SpringIDE.

## SpringIDE

The SpringIDE plug-in (available at <http://springide.org/updatesite/>) has really improved in recent versions and is now a must have for developing with Spring. This tool will give us enhanced code completion for our Spring bean files and will help us generate the nice bean graph figures that you saw in Chapters 5 and 6. Figure A-1 shows the URL you'll need.

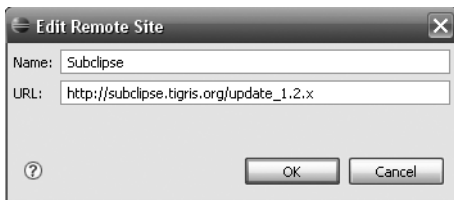


**Figure A-1.** Adding the Spring IDE plug-in

To read all about this plug-in, you can go to <http://www.springide.org/blog/> for the latest information. Next, we'll install a plug-in that will let us access the ToCollege.net code repository straight from Eclipse.

## Subclipse

Subclipse (available at [http://subclipse.tigris.org/update\\_1.2.x](http://subclipse.tigris.org/update_1.2.x)) isn't glamorous, but it's a solid tool for connecting Eclipse to a Subversion repository. The URL for Subclipse is shown in Figure A-2.

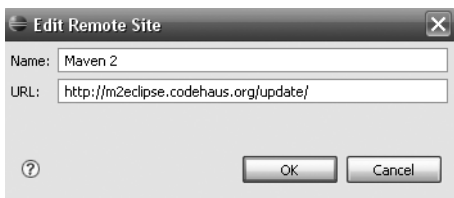


**Figure A-2.** Adding the Subclipse plug-in

The web site for all the information you need about this plug-in is <http://subclipse.tigris.org/>. With our code downloaded, we'll still need something to help us download the code dependencies and configure the build paths. That's what Maven's for.

## Maven Plug-in

The URL for adding the m2eclipse Maven plug-in (<http://m2eclipse.codehaus.org/update/>) is shown in Figure A-3.



**Figure A-3.** m2eclipse IDE plug-in

The web site for information on m2eclipse is <http://m2eclipse.codehaus.org/index.html>. There's a competing plug-in called MavenIDE, but I've been happy with m2eclipse and haven't tried switching. The URL for the MavenIDE site is <http://mevenide.codehaus.org/mevenide-ui-eclipse/features.html>. There, you'll be able to read up on the current version of this plug-in. Whichever plug-in you use, a Maven plug-in is an especially important one, as it is going to make sure that Eclipse can understand what JARs our project will need to compile.

## Other Eclipse Plug-ins

Just because we've stopped installing plug-ins doesn't mean you need to stop. It's always worth taking a look at good lists of Eclipse plug-ins. There's nothing better than finding a helpful tool. This list has a nice overview of some great Eclipse plug-ins: <http://ist.berkeley.edu/as/ag/tools/howto/eclipse-plugins.html>.

## Cygwin

Installing Cygwin, available at <http://www.cygwin.com/>, might not be necessary, but I, for one, can't imagine living without it on a Windows machine. If you're going to spend any amount of time on the command line in Windows, you really owe it to yourself to get as far away as you can from the `cmd.exe` application that comes with Windows. Better yet, Cygwin makes it easy to install tools like Subversion and `ssh` as part of your download. It's not required, so we won't go into any more detail here, but I definitely suggest that you give it a shot!

## Maven

This is the last big tool that we're going to need to install. Let me spend a couple quick seconds explaining what Maven is going to do for us, and then I'll show you how to download it.

### About Maven

The main build tool that we're going to be using is Maven. Maven is a significant step forward from older systems like Ant. Deep at the core of Maven is the concept of convention over configuration. The idea is that most Java projects follow fairly similar life cycles no matter what domain they're in. By standardizing this build cycle, Maven can perform many tasks for you with no configuration at all. Additionally, the rich ecosystem of Maven plug-ins can easily integrate at the appropriate stage of the cycle. The Maven build cycle follows:

1. *Validate*: Validate that the project is correct and that all necessary information is available.
2. *Compile*: Compile the source code of the project.
3. *Test*: Test the compiled source code using a suitable unit testing framework. These tests should not require the code to be packaged or deployed.

4. *Package*: Take the compiled code and package it in its distributable format, such as a JAR.
5. *Test integration*: Process and deploy the package, if necessary, into an environment where integration tests can be run.
6. *Verify*: Run any checks to verify the package is valid and meets quality criteria.
7. *Install*: Install the package into the local repository for use as a dependency in other projects.
8. *Deploy*: In an integration or release environment, copy the final package to the remote repository for sharing with other developers and projects.

This is the list of steps straight from the horse's mouth: <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>. The neat thing about Maven is that it really takes these life cycle phases to heart. No matter what Maven project you download, you'll be able to run `mvn test`, and the project should automatically download all of its dependencies (in the validate stage), compile (in the eponymous stage), and run all of its unit tests. This is really pretty amazing, and it regularly works like a charm.

All of this functionality doesn't come without a bit of, shall we say, cognitive overhead. There's a solid philosophy at work here, but if you've never seen it before, it's going to seem a bit foreign, which can make understanding the errors that Maven throws seem a little mysterious at first. There's a great roundup of the pros and cons of Maven on InfoQ here: <http://www.infoq.com/news/2008/01/maven-debate>. One of my favorite quotes is from Rich Hightower, which I think captures a bit of the Maven experience.

*Every day I curse Maven. Every day I praise Maven. I hate it and love it all of the time. I remember although it could be better it is a far cry from using Ant. Since I travel a lot and consult/develop a lot . . . I have seen so many snarly Ant build scripts. At least with Maven, I have to just tame one beast and one philosophy. With Ant, it is random beast with many heads.*

— Hightower

([http://raibledesigns.com/rd/entry/re\\_why\\_grails\\_doesn\\_t](http://raibledesigns.com/rd/entry/re_why_grails_doesn_t))

The good news is that there's a wealth of information available about Maven. The best place to start is the 250-plus-page book published by Mergere; it's available free from <http://maven.apache.org/articles.html>.

ToCollege.net uses Maven heavily but doesn't do anything too weird with it, so it should be a good place to get started. First off, though, we've got to install it.

## Install Maven

To install Maven for your platform, go to <http://maven.apache.org/>, and follow the links to download the most recent version. We're going to install version 2.0.8 here.

---

**Note** Mac OS X Leopard users should have Maven installed by default. Try typing **mvn -version** at the command line. If it's version 2.0 or higher, you should be all set.

---

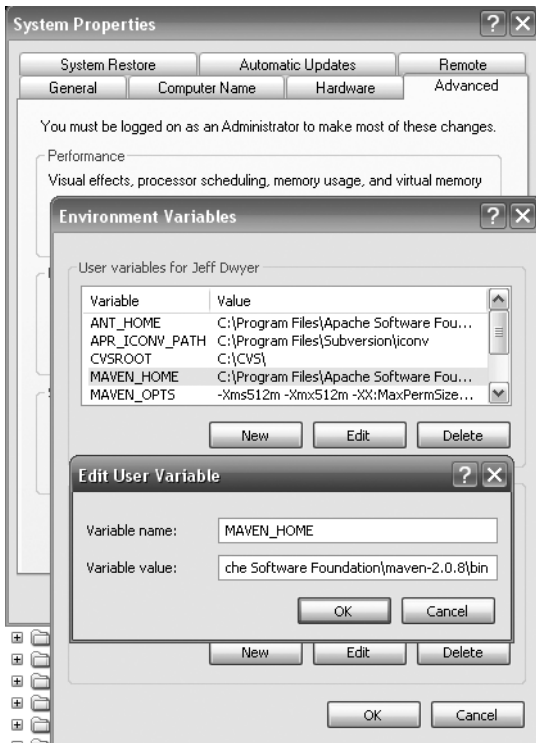
You can use the following two guides to get you up and running quickly:

- <http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
- <http://maven.apache.org/guides/getting-started/index.html>

The basic steps for setting up Maven are going to be something like the following:

1. Download and unzip `apache-maven-2.0.8-bin.zip`.
2. Copy it to `C:\Program Files\Apache Software Foundation\maven-2.0.8`.
3. Edit your environment variables (press Start + Pause/Break to bring up the System Properties window, and from there, click Environment Variables). Add `MAVEN_HOME` as the "Variable name" and `C:\Program Files\Apache Software Foundation\maven-2.0.8\bin` for the "Variable value".
4. Add `;%MAVEN_HOME%` to the `PATH` environment variable (the `PATH` variable should already exist).

Figure A-4 shows the dialogs for editing your environment variables on Windows.



**Figure A-4.** Setting up Maven environment variables

By doing this, we've enabled the Maven executable, which lives in the `bin/` directory, to be run without specifying the full path. To test that we've done this correctly, we can open Cygwin and type `mvn -version`.

You should be greeted with something that looks like Figure A-5.

```
$ mvn -version
Maven version: 2.0.8
Java version: 1.5.0_05
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
```

**Figure A-5.** Verifying that you've installed Maven correctly

Excellent. With Maven installed, we're ready to get down to business and pull down the ToCollege.net source code.

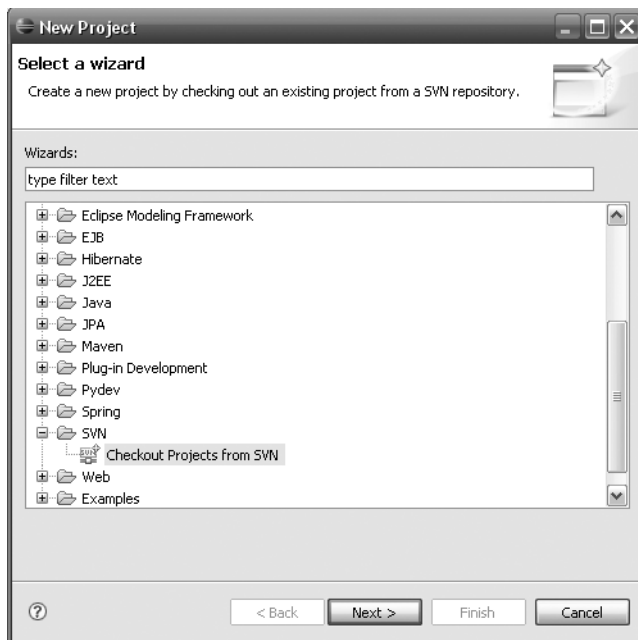
**Note** Maven will default to putting your local repository (the downloaded JAR files) in your user profile under a directory called `.m2/repository/`. To change this location, you can use the `M2_REPO` environment variable.

## The ToCollege.net Codebase

The ToCollege.net source code is hosted on Google code hosting at <http://code.google.com/p/tocollege-net/>. This site has a nice Subversion source code browser, issue tracker, and wiki, as well as links to the ToCollege.net web site, blog, and other resources.

### Check Out the Source Code

To manually check out the code, you can go to <http://code.google.com/p/tocollege-net/source/checkout> and follow the directions listed there. Since we're going to want to develop the code inside our IDE, we're going to use the Subclipse plug-in we installed earlier to download the source directly into our workspace. Let's go through the checkout process. First, we need to go to the file menu and select **Create** ► **New Project** to open the dialog shown in Figure A-6.



**Figure A-6.** *The New Project dialog*

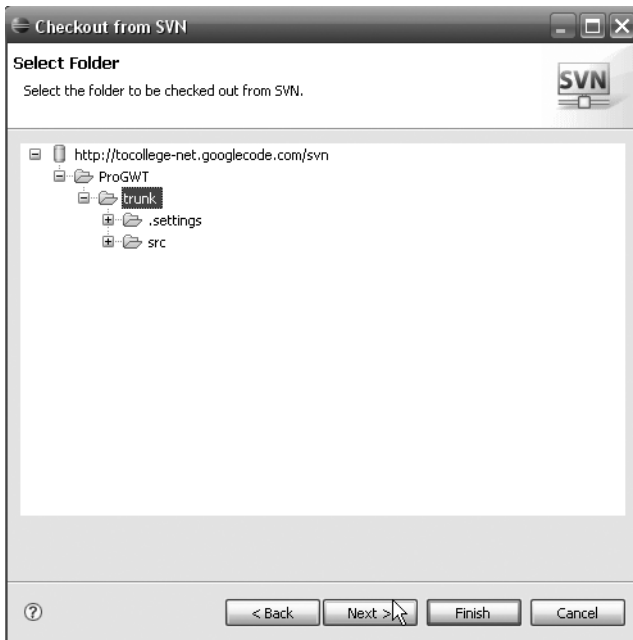
Our Subclipse plug-in has added a new entry called Checkout Projects from SVN. That's what we want, so click Next. In this next field, shown in Figure A-7, we're prompted for the URL of the repository.



**Figure A-7.** *Specifying the Google hosting repository*

Note here that we don't want to enter the same thing that the Google code site tells us to. More precisely, we don't want the ending `/trunk/` in this field. Instead, we'll select our project in the next dialog, shown in Figure A-8. It's basically the same thing; we're just breaking it up into two parts. The first dialog is asking for just the repository location, and `/trunk/` specifies the revision within this repository. If we included `/trunk/` in the URL and then wanted to check out a different revision or tag, we would need to create a whole new repository.

You can see that with our repository there is a ProGWT project. Under that project is a trunk directory. That's what you want to select for checking out. This is basic Subversion functionality, but if you're used to CVS, it can be a little weird. For an overview of Subversion's trunk, tags, and branches, take a look at <http://svnbook.red-bean.com/en/1.1/ch04s07.html>. By checking out the trunk, you'll get the latest ToCollege.net code. All that's left to do is specify a project name in our Eclipse project, as shown in Figure A-9.

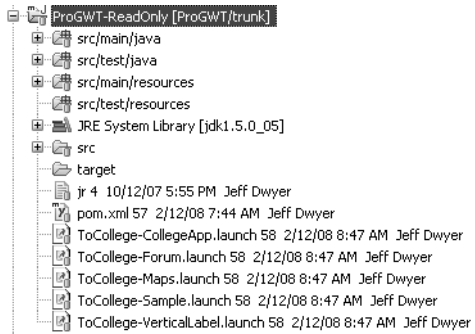


**Figure A-8.** *Select the trunk directory*



**Figure A-9.** *Picking a name for our project*

We'll call it ProGWT-ReadOnly to refer to the fact that we won't be able to commit any changes we make back to the repository. There's certainly nothing stopping us from making any changes, however. Click Next, and Subclipse should pull down all the code you'll need.

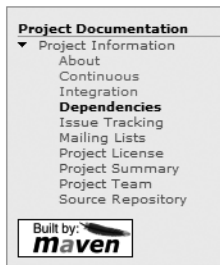


**Figure A-10.** *ProGWT-ReadOnly in Eclipse*

Fantastic! All the source code is downloaded, and we're ready to get this thing compiling!

## Compiling with Maven

As I said before, Maven is going to run our build process. This means that it will be responsible for downloading all of the JAR files that our project has dependencies on. The list is pretty substantial. Figure A-11 shows the first part of the list.



## Project Dependencies

### compile

The following is a list of compile dependencies for this project. These dependencies are required to compile and run the application:

GroupId	ArtifactId	Version	Classifier	Type
aspectj	aspectjrt	1.5.0	-	jar
aspectj	aspectjweaver	1.5.2	-	jar
c3p0	c3p0	0.9.1	-	jar
com.allen_sauer.gwt.dnd	gwt-dnd	2.0.7	-	jar
com.allen_sauer.gwt.log	gwt-log	1.5.1	-	jar
com.google	gwt-google-apis	1.5.0.build99	-	jar
com.google	gwt-incubator	0.0.1-20080117	-	jar
com.google	gwt-servlet	1.5.0.build1806	-	jar
com.ibm.icu	icu4j	3.4.4	-	jar
commons-beanutils	commons-beanutils	1.7.0	-	jar
commons-lang	commons-lang	2.1	-	jar
javax.mail	mail	1.4	-	jar
mysql	mysql-connector-java	5.0.5	-	jar
opensymphony	sitemesh	2.3	-	jar
org.apache.lucene	lucene-highlighter	2.2.0	-	jar
org.apache.lucene	lucene-snowball	2.2.0	-	jar

**Figure A-11.** Some of ToCollege.net's dependencies

This nice printout was created by simply running the `mvn site` command on ToCollege.net. All told, there are about 33 direct dependencies for ToCollege.net. There's a little secret hiding within that sentence however. I said *direct* dependencies. Many of the JARs listed here actually rely on other JARs to function; they are what Maven calls transitive dependencies. Happily, just as the `mvn site` command was able to print us out this wonderful list, it can also print out a list of all of our project's transitive dependencies as well (see Figure A-12).

## Project Dependency Graph

### Dependency Tree

- com.apress.progwt:ProGWT:war
  - com.google.gwt-user:jar
  - com.google.gwt-dev-windows:jar
  - com.google.gwt-servlet:jar
  - com.google.gwt-google-apis:jar
  - com.google.gwt-incubator:jar
  - com.allen\_sauer.gwt.dnd.gwt-dnd:jar
  - com.allen\_sauer.gwt.log.gwt-log:jar
  - org.json:json:jar
  - org.springframework:spring:jar
  - aspectj:aspectjrt:jar
  - aspectj:aspectjweaver:jar
  - org.springframework:spring-webmvc:jar
    - org.springframework:spring-context-support:jar
    - org.springframework:spring-web:jar
  - org.springframework.security:spring-security-openid:jar
    - org.springframework.security:spring-security-core:jar
    - org.openid4java:openid4java:jar
  - org.freemarker:freemarker:jar
  - opensymphony:sitemesh:jar
  - org.springframework:spring-orm:jar
  - org.springframework:spring-jdbc:jar
    - org.springframework:spring-context:jar
    - org.springframework:spring-tx:jar
  - org.springframework:spring-aop:jar
    - aopalliance:aopalliance:jar
    - org.springframework:spring-beans:jar
    - org.springframework:spring-core:jar
  - org.hibernate:hibernate:jar
    - net.sf.ehcache:ehcache:jar
    - javax.transaction:jta:jar
    - asm:asm-attribs:jar
    - dom4j:dom4j:jar
    - antlr:antlr:jar
    - cglib:cglib:jar
    - asm:asm:jar
    - commons-collections:commons-collections:jar

**Figure A-12.** *Some of ToCollege.net's transitive dependencies*

Now, this list of JARs stretches to about 60 files. If you'd like to go to all of the web sites implicated here and download the latest versions, you're more than welcome. I, for one, am going to let Maven do my dirty work and grab all of these automatically.

---

**Tip** To simply generate this tree, run the `mvn dependency:tree` command.

---

## Install-All

Before we let Maven install all these dependencies, we're going to give it a leg up. Let's drop into the command line to run a batch file that comes with ToCollege.net:

```
cd workspace/ProGWT-ReadOnly
cd setup/maven
./install-all
```

This `install-all` program is a little batch file that is going to make our life easier. While the number of Maven JARs that are available in central locations is amazing, there are some that aren't available. For this reason, there are a few pesky JARs in this directory that need to be installed manually. This script simply runs those Maven installs manually. Open the batch file to see exactly what's going on here. All we're doing is copying the JAR files you see in this directory into our local Maven repository in the appropriate directory structure by using the `mvn install` command.

---

**Note** In an enterprise setting, the way to handle this is by setting up your own repository. See <http://maven.apache.org/guides/introduction/introduction-to-repositories.html> for more details.

---

Time to let Maven do its thing!

## Running mvn compile

Let's see this transitive dependency thing in action. All we need to do is run the following two commands:

1. `cd workspace/ProGWT-ReadOnly`
2. `mvn compile`

When we do this, Maven will begin to download files from the Internet that it needs to compile, which will look something like Figure A-13.

```
$ mvn compile
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building Unnamed - com.apress.progwt:ProGWT:war:1.0-SNAPSHOT
[INFO] task-segment: [compile]
[INFO] -----
[INFO] artifact org.codehaus.mojo:cobertura-maven-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/codehaus/mojo/cobertura-maven-plugin/2.2/cobertura-
5K downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/mojo/mojo/16/mojo-16.pom
8K downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/mojo/cobertura-maven-plugin/2.2/cobertura-
28K downloaded
Downloading: http://repo1.maven.org/maven2/org/mortbay/jetty/maven-jetty-plugin/6.1.2rc2/maven-jet
4K downloaded
Downloading: http://repo1.maven.org/maven2/org/mortbay/jetty/project/6.1.2rc2/project-6.1.2rc2.pom
11K downloaded
Downloading: http://repo1.maven.org/maven2/org/mortbay/jetty/maven-jetty-plugin/6.1.2rc2/maven-jet
36K downloaded
```

**Figure A-13.** *Maven begins to fetch resources.*

As you can see, Maven is going to start downloading our dependencies—all of them. If you haven't used Maven before, this is going to take a good little while since you'll be starting with an empty repository. All told, there are about 150MB of JARs required to get ToCollege.net off the ground. If this seems like a lot of work, consider all the legwork that Maven is saving you. Without Maven, we'd more than likely be doing the familiar Ant build do-si-do:

1. Try to build.
2. Get an error with a "can't find class name" exception.
3. Try to guess what JAR this missing class comes from.
4. Find the web site of the JAR, download the file, and add it to the lib directory.
5. Go back to step 1.

Oh, and did I mention that there's no good way to know what version of each JAR we should use? With Maven, all of our dependencies are laid out for us, version numbers included.

The good news is that once Maven downloads these files to your repository, they're there. So while downloading does take a while to happen, in the future, Maven will need to download only new JARs when you upgrade. Want to upgrade from FreeMarker 2.3.10 to 2.3.11? All you need to do is change the <version> tag in your pom.xml file, as shown in Listing A-1, and Maven will go fetch the new JAR.

**Listing A-1.** *ProGWT/pom.xml*

```
<dependency>
  <groupId>org.freemarker</groupId>
  <artifactId>freemarker</artifactId>
  <version>2.3.11</version>
</dependency>
```

Not bad eh? Enough bragging about the capabilities of Maven. There's one real hurdle we need to clear before we're ready to go. Currently, GWT isn't in any central repositories, so Maven isn't going to be able to find it. Let me teach you how to install JARs yourself so that we can get GWT working and actually make ToCollege.net run.

## Installing GWT

This book was written concurrently with the development of GWT 1.5. This version is a huge step forward for GWT, but it is taking a little longer to make it out the door than I initially thought it would. At the time of this writing, GWT 1.5 has just released milestone 1. While this isn't an official GWT release, it's the next best thing. This JAR isn't meant for production use, but by the time this book is in stores, 1.5 should be released.

---

**Note** Stay tuned to the <http://code.google.com/p/tocollege-net/> web site where we'll keep track of everything you need to know to compile with the latest versions of GWT.

---

## Download GWT for Your Platform

The GWT download list can be found at <http://code.google.com/p/google-web-toolkit/downloads/list>. GWT 1.5 milestone 1 was released on four platforms; to get started, you'll just need to download the zip file that is appropriate for the operating system that you're working on:

- Use `gwt-mac_10.5-0.0.2030.zip` for Mac OS X 10.5 (Leopard).
- Use `gwt-mac_10.4-0.0.2030.zip` for Mac OS X 10.4 (Tiger).
- Use `gwt-linux-0.0.2030.zip` for Linux.
- Use `gwt-windows-0.0.2030.zip` for Windows.

Once you've downloaded this archive, unzip it.

The next step is to run a number of `mvn install` commands to copy the three main GWT JARs into the appropriate Maven repository directories. I've saved these commands to some script in the ToCollege.net Setup directory to help you out:

```
cd workspace/ProGWT-ReadOnly/Setup/maven/gwt
chmod a+x install*
cp install-windows /downloads/gwt-windows-0.0.2030/
cd /downloads/gwt-windows-0.0.2030/
./install-windows 1.5.0-M1
```

Here, you can see that we first make sure that all of these scripts are have execution privileges by running the `chmod a+x` command. Next, we simply copy the script into the folder that was created when we unzipped the GWT download. Finally, we run the script, passing in “1.5.0-M1” as the version parameter. This parameter will be used as the Maven artifact version.

This little install batch file means we that we don't need to remember the Maven command and type it in by hand. It's essentially a single line like this, which takes the version number as a parameter:

```
mvn install:install-file -DgroupId=com.google -DartifactId=gwt-google-apis -Dversion=$1 -Dpackaging=jar -Dfile=build\lib\gwt-google-apis.jar -DgeneratePom=true
```

Our install script will actually run a line like the preceding one three times, once for each of the three JARs that we'll need for GWT. After that, the install script will copy the platform-specific development resources into the repository. For Windows, this is two `.dll` files. For Linux and Mac platforms, this will be a number of `.lib*` files. Feel free to open the install file to see precisely what's going on behind the scenes.

## Maven Profiles

One trick that we should cover is how we're using Maven profiles to make sure that we compile with the correct JARs on the classpath for each platform. For Windows, we want to install the file `gwt-dev-windows.jar`. For Mac, we'd like to install `gwt-dev-mac.jar`. So the question is, “How do we change our classpath depending on which operating system we're using?” We could directly edit the `pom.xml` file to include the correct version, but this would be annoying when it came to checking the file into version control. Happily, there's a better way to do this. It's called Maven profiles.

Basically, the idea of profiles in Maven is that a single `pom.xml` can specify different options for different environments. The neat thing is how many ways Maven gives us to specify our environment. Environments can be anything from environment variables to parameters that you pass in on the command line to automatic detection of the JDK or

operating system attributes. For a full look at profiles, check out <http://maven.apache.org/guides/introduction/introduction-to-profiles.html>. Operating system detection is just what we're looking for, so let's see this in action. The goal here is to include the right artifactId for the right operating system. To do that, we'll make the artifactId a variable called `gwt-dev`, and then we'll set the value of this variable in the profile as shown in Listing A-2.

**Listing A-2.** *Operating System Profiles in pom.xml*

```
<profiles>
  <profile>
    <id>windows</id>
    <activation>
      <os>
        <family>windows</family>
      </os>
    </activation>
    <properties>
      <gwt-dev>gwt-dev-windows</gwt-dev>
    </properties>
  </profile>
  <!--profile for mac and linux elided-->
</profiles>
<dependencies>
  <dependency>
    <groupId>com.google</groupId>
    <artifactId>gwt-user</artifactId>
    <version>1.5.0-M1</version>
  </dependency>
  <dependency>
    <groupId>com.google</groupId>
    <artifactId>${gwt-dev}</artifactId>
    <version>1.5.0-M1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.google</groupId>
    <artifactId>gwt-servlet</artifactId>
    <version>1.5.0-M1</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

You can see that this is pretty straightforward. We describe a profile called `windows` and specify that it activates when the operating system family is “windows”. This will happen automatically, and when it does, the property `gwt-dev` will be created with the value `gwt-dev-windows`. All we need to do then is reference that variable in our `artifactId`.

You shouldn't have to change anything here to get ToCollege.net to work, but it's a neat demonstration of how Maven can be easily customized. You should keep Maven profiles in mind as you look at changing between development, test, and deployment environments.

### Compiling GWT from Source

An alternative way to install GWT is to install directly from the GWT source. If you want to get the latest GWT updates and can't wait for releases, this is what you'll need to do.

---

**Note** If you're happy waiting for official releases, you can safely skip this section. I just added this for completeness, since GWT 1.5 isn't officially released at the time of this writing.

---

Of course, GWT is open source, so we can easily download the source, compile it, and install the resulting JARs to our Maven repository if we want to be on the bleeding edge. Use the following commands to make this happen:

1. `mkdir gwt`
2. `cd gwt`
3. `svn checkout http://google-web-toolkit.googlecode.com/svn/trunk/ gwt-trunk`
4. `svn checkout http://gwt-google-apis.googlecode.com/svn/trunk/  
gwt-google-apis-read-only`

First, we're going to make a directory on our machine that will house all of the Google source code. Then, we'll check out the GWT source code. You can see that we're checking out the HEAD revision here. If we want to check out a particular build number, we could just as easily do that by adding an `-r 2030` argument (using revision 2030 as an example) to our command. Finally, we check out the HEAD revision of the GWT Google APIs.

With the code checked out, let's copy some helper scripts into the GWT directories. These will just save us some typing when we go to install our JARs into our Maven repository. The helper scripts are located in the `Setup/maven/gwt` directory of ToCollege.net's source code. To copy the two scripts over, you should be able to use a series of commands like the following:

1. `cd workspace/ProGWT-ReadOnly/Setup/maven/gwt`
2. `cp -r gwt-google-apis-read-only/ /cygdrive/c/gwt/`
3. `cp -r gwt-trunk/ /cygdrive/c/gwt/`

Take a look in the directories you're copying if you want to see what's going on. All we're doing is copying some very simple scripts over. With those in place, let's build GWT:

```
cd gwt-trunk
ant
./install-windows 1.5.0-SNAPSHOT
```

GWT compiles with Ant, so all we need to do is run `ant`. Once this completes, GWT will be compiled, and three JARs will have been created. Running this install script will run a Maven `install` command, which will copy these JARs to the local repository. Now, all we need to do is compile the GWT Google APIs.

```
cd ../gwt-google-apis-read-only
ant
./install 1.5.0-SNAPSHOT
```

This just does the same old thing. With these in place, we'll be able to include GWT code in our `ToCollege.net` project with the dependencies shown in Listing A-3.

**Listing A-3.** *ProGWT/pom.xml*

```
<dependency>
  <groupId>com.google</groupId>
  <artifactId>gwt-user</artifactId>
  <version>1.5.0-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.google</groupId>
  <artifactId>gwt-dev-windows</artifactId>
  <version>1.5.0-SNAPSHOT </version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.google</groupId>
  <artifactId>gwt-servlet</artifactId>
  <version>1.5.0-SNAPSHOT </version>
</dependency>
```

```
<dependency>
  <groupId>com.google</groupId>
  <artifactId>gwt-google-apis</artifactId>
  <version>1.5.0-SNAPSHOT</version>
</dependency>
```

Fantastic! Whether you're happy with the GWT official releases or you're running right off the GWT trunk, Maven should now have absolutely everything it needs for ToCollege.net to run.

## Manually Installing JARs

Before we move on, let's just take a quick look at a general purpose script that we can use to install any old JAR that we come across. Because the `mvn install` syntax is admittedly a pain, I find it useful to have a little script so I don't need to remember the exact syntax. On Linux (or Cygwin), I just save the line below to a file called `install`:

```
mvn install:install-file -DgroupId=$1 -DartifactId=$2 -Dversion=$3 -Dpackaging=jar
-Dfile=$4 -DgeneratePom=true
```

Now, from a command prompt, we can simply type the following line, inserting the `groupId`, `artifactId`, version number, and JAR location as parameters. Say we download a Compass JAR and want to add it to the repository. All we need to do is run this:

```
./install org.opensymphony compass 1.2.0 compass.jar
```

And Compass will be available for Maven.

## Other GWT Libraries

If you plan on installing other GWT libraries, you're going to need to first install them to Maven. The best way to do this is to download the third-party JAR of the library, then copy the simple install script that I just showed you into the same directory as the library and run it with the appropriate parameters. You'll need to do this if you want to use other available GWT libraries such as those listed on these sites:

- <http://www.gwt-site.com/top-5-gwt-libraries/>
- <http://www.infoq.com/news/2008/01/gwt-frameworks>

Once you install the libraries to your Maven repository, you'll be all set to add them to your classpath by adding them to your `pom.xml` file. With no further ado, let's see some of this code in action!

## Run the Sample Calculator

Now that ToCollege.net compiles. Let's give her a whirl. We won't be able to run the full ToCollege.net application yet, because we haven't set up the database, but that doesn't mean we can't see the calculator from Chapter 2.

The ToCollege.net source comes with a number of preset launch configurations so that you can run samples in hosted mode right out of Eclipse. All you need to do is right-click the launch file, as shown in Figure A-14.

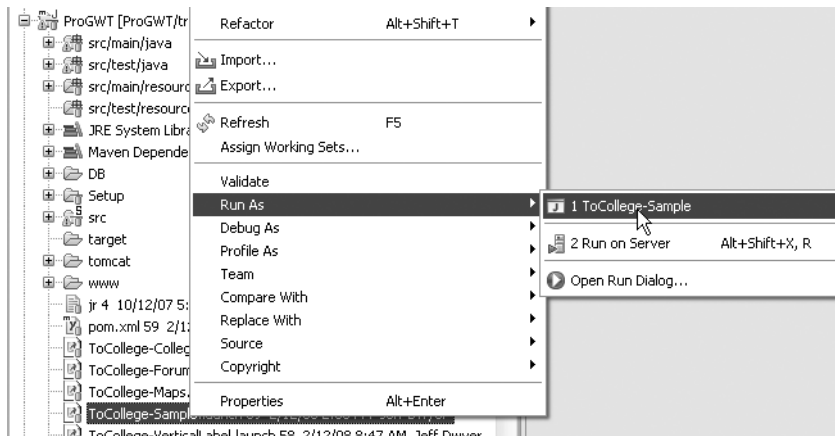


Figure A-14. Running the sample launch configuration

After that, you'll be able to bring up the calculator (see Figure A-15).

---

**Note** Mac users have their own special launch configurations. These launch configurations (ToCollege-Sample-Mac.launch) pass a special `-XstartOnFirstThread` argument to the Java VM. This is a Mac OS X requirement for GWT hosted mode.

---

Excellent—that's simple GWT integration finished. Now, let's set up the database and get our server side up and running.

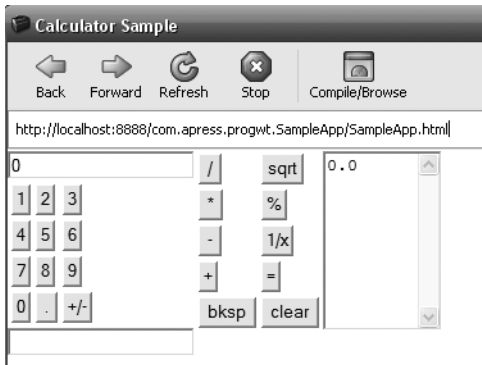


Figure A-15. The sample running in hosted mode

## Set Up the Database

To help you quickly set up your environment, I'm going to recommend that you install WAMP. The only strict requirement for ToCollege.net is that you set up a functioning MySQL database, but WAMP is an easy way to get some great tools installed as well. To install WAMP, go to <http://www.wampserver.com/en/>, and follow the directions there. WAMP will install current and compatible versions of

- Apache
- MySQL
- PHP
- phpMyAdmin

Again, all you really need to get ToCollege.net running is a MySQL database, but I find that the phpMyAdmin interface makes simple database management tasks much easier. Moreover, WAMP is so darn easy to use that it's worth installing even if you never use anything except its MySQL launcher.

---

**Tip** Mac users, you can use the excellent MAMP (<http://www.mamp.info/>), which is just like its Windows counterpart. The only thing to note is that, by default, MAMP runs on port 8888, which conflicted with GWT hosted mode. To change this, I simply went to the MAMP control panel and selected Run on Default Ports, which changed Apache to run on port 80 and MySQL to run on port 3306.

---

During setup, use whatever username and password you'd like. We'll go over where you need to reference them in a moment.

When you've got WAMP installed, you should be able to get to a phpMyAdmin screen like Figure A-16.

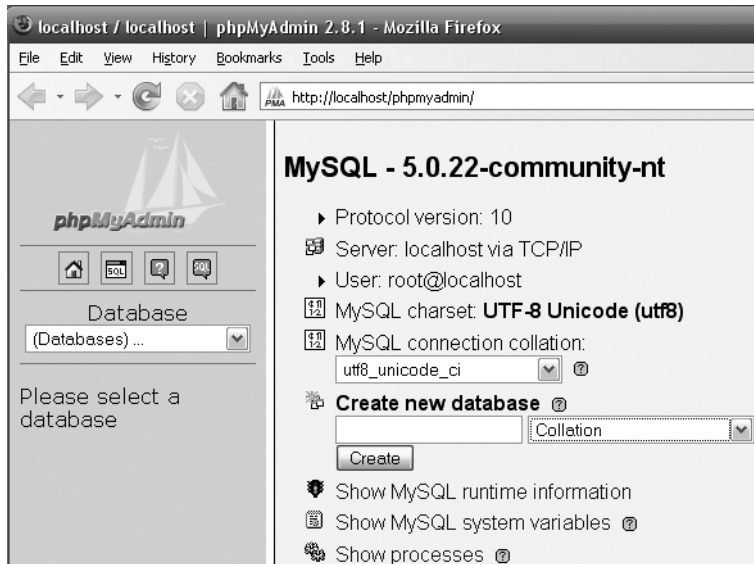


Figure A-16. *phpMyAdmin*

Here, you'll want to create a new database of collation type `utf8_bin`. All you need to do is select that from the list and type in the database name: **progtw**.

Once you've done that, you should create a user named `progtw` for the host `localhost` with password `progtw`. You'll want to give this user permissions for all CRUD operations on the `progtw` database you just created. Now, we're ready to create our tables.

The tables needed to run ToCollege.net are available in the folder `ProGWT/Setup/db/`. Open the `schema.sql` file in Eclipse to see the ToCollege.net schema. This is simply a big text file of SQL commands, specifically all of the `CREATE TABLE` commands that you'll need to have your own ToCollege.net database. Select all of this file, and click the SQL tab in phpMyAdmin. Paste in the SQL, and click execute. You should be treated to something that looks like Figure A-17.

	Table	Action	Records	Type	Collation
<input type="checkbox"/>	applications		0	InnoDB	utf8_bin
<input type="checkbox"/>	application_cons		0	InnoDB	utf8_bin
<input type="checkbox"/>	application_process_map		0	InnoDB	utf8_bin
<input type="checkbox"/>	application_pros		0	InnoDB	utf8_bin
<input type="checkbox"/>	application_ratings_map		0	InnoDB	utf8_bin
<input type="checkbox"/>	bar		0	InnoDB	utf8_bin
<input type="checkbox"/>	foo		0	InnoDB	utf8_bin
<input type="checkbox"/>	forumposts		0	InnoDB	utf8_bin
<input type="checkbox"/>	mailing_list		0	InnoDB	utf8_bin
<input type="checkbox"/>	processtypes		0	InnoDB	utf8_bin
<input type="checkbox"/>	ratingtypes		0	InnoDB	utf8_bin
<input type="checkbox"/>	schools		0	InnoDB	utf8_bin
<input type="checkbox"/>	users		0	InnoDB	utf8_bin
<input type="checkbox"/>	user_priorities_map		0	InnoDB	utf8_bin
<input type="checkbox"/>	user_processtype		0	InnoDB	utf8_bin
<input type="checkbox"/>	user_ratingtype		0	InnoDB	utf8_bin
	<b>16 table(s)</b>	<b>Sum</b>	<b>0</b>	<b>InnoDB</b>	<b>utf8_bin</b>

Figure A-17. *phpMyAdmin showing our new database tables*

Now, we've got all our tables but no data. Let's do the same copy and paste maneuver to get some initialization data. Open the `init.sql` file, and paste it into the SQL entry blank, and you should be all set. This initialization data will give us 21 schools and 2 users to work with. The first user is `test` and has the password `testaroo`, and the second is `unit-test` and has the password `unit-test`. Of course, you'll be able to add more users through the regular interface.

## Environment Setup

Let's take a look at the two properties files that ToCollege.net uses to configure its operation.

## The config.properties File

The main properties file for ToCollege.net is `config.properties`. In this file, we'll define all of the properties that make ToCollege.net run. The first section, shown in Listing A-4, contains the properties that are the same no matter what platform we're deploying to.

### Listing A-4. *src/main/resources/config.properties*

```
db.dialect=org.hibernate.dialect.MySQLInnoDBDialect
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/progwt
gears.betaManifestVersion=1
users.maxUsers=3
users.startInvitations=2
```

You can see that we have a number of basic properties here. Note especially the `users.maxUsers` property. This is the total number of users that you'll be able to add to the system before is says, "We're full!" After that, it diverts signup requests to the mailing list. This is a nice feature for our beta software to have. To change this limit, you'll just need to increase this number.

The second section of this file contains properties that are referenced with our `HostPrecedingPropertyPlaceholderConfigurer` class. This class replaces a property like `${HOST.compass.repo}` with `${sonata.compass.repo}` (if your machine is named `sonata`). This is helpful because many times you'll want different properties for different machines. You can see in Listing A-5 that we are specifying different paths for Windows, Mac, and Linux distributions. The windows machine is named `sonata`; the Mac is named `silvia`, and the Linux box is named simply `linux`.

### Listing A-5. *src/main/resources/config.properties*

```
#windows sample, machine name 'sonata' our 'development' options
sonata.compass.repo=file://DB/compass/toCollege
sonata.openID.trustRoot=http://localhost:8080
sonata.gwt.serializeEverything=true
sonata.gears.localserver.dir=C:\\workspace\\ProGWT\\target\\ProGWT-1.0-SNAPSHOT\\
com.apress.progwt.Interactive
sonata.gears.localserver.url=http://localhost:8080/com.apress.progwt.Interactive/

#mac sample, machine name 'silvia' our 'development' options
silvia.local.compass.repo=file:///var/tmp/DB/compass/toCollege
silvia.local.openID.trustRoot=http://localhost:8080
silvia.local.gwt.serializeEverything=true
```

```
silvia.local.gears.localserver.dir=~/workspace/ProGWT/target/ProGWT-1.0-SNAPSHOT/➤
com.apress.progwt.Interactive
silvia.local.gears.localserver.url=http://localhost:8080/➤
com.apress.progwt.Interactive/

#linux sample, machine name 'linux', our 'deployment' options
linux.compass.repo=file:///var/tmp/DB/compass/toCollege
linux.openID.trustRoot=http://www.tocollege.net
linux.gwt.serializeEverything=false
linux.gears.localserver.dir=/tomcat/webapps/ROOT/com.apress.progwt.Interactive/
linux.gears.localserver.url=http://www.tocollege.net/com.apress.progwt.Interactive/
```

To get this working for your machine, you'll need to add your machine to this file. The easiest thing to do is to just change the host name in the section of properties that is appropriate for the platform that you're working on. The main thing that you'll need to have working is the `compass.repo` property.

If you're not sure what the name of your machine is, don't worry. When you run the application and these properties are not set, you will get log message that inform you of the name of all `HostProceedingProperties` that haven't been set. This warning will look like this:

```
Please define property: silvia.local.compass.repo
```

The first part of this property name is the host name that you'll need to use. All you need to do is copy this line into the properties file and give it a value; then, you should be all set.

This method of host-specific properties makes it easy to configure your application to run differently in different environments. Instead of having to modify one central `config.properties` file and worry about whether or not to commit it to your source code repository, you can easily leave all the configurations in and commit them all. There's one thing left to do however. This file is still not a good place to put the database password. For the real secure stuff, we'll specify a secondary properties file.

## The `env.properties` File

When I went to share the ToCollege.net code with you I ran into an issue. How was I going to avoid sharing my database password with you as well? The password is naturally stored in a properties file, but I'd like to commit the properties file so that you can see what it contains. My solution is to have a second properties file that I copy over the default file as part of my deployment process. To make this file as small as possible, I've separated it from the `config.properties` file.

---

**Note** Another good way to avoid this problem is to just use JNDI. See <http://www.infoq.com/articles/spring-2.5-part-1> for some good tips on this. Indeed, JNDI is probably the best way to do this, but I didn't want you to have to setup JNDI datasources just to get these samples to run.

---

Let's take a look at the file contents now. You may need to edit these entries to make sure that our code will use the correct username and password to access this database. Open the `env.properties` file in `src/main/resources`. It will contain the entries shown in Listing A-6.

**Listing A-6.** `src/main/resources/env.properties`

```
env.jdbc.user=progtw
env.jdbc.password=progtw
env.invitations.masterkey=change_this
env.invitations.salt=CHANGE_THIS_SALT
env.security.anonymous.key=CHANGE_THIS_1
env.security.remembersme.key=CHANGE_THIS_2
```

For right now, all you need to do is ensure that the `user` and `password` properties will work for the database that you just set up. This file is separated from the other properties in an effort to get some separation for truly sensitive configuration data. The idea is that when you deploy your application, you should copy a suitable `env.properties` file over this file. It's not a perfect solution, since it's still easy to commit the local database password, but that's probably not too big a deal. Just *be sure that this password isn't the same as the real live database password*, and you should have a decent first crack at protecting your database password. For ToCollege.net, it sufficed as a way to make sure that I didn't actually open source live database passwords! Sometimes, openness just goes too far.

## Running ToCollege.net

OK, we're ready for the big show. Let's fire up the server. Typically, this would mean installing Tomcat or Jetty, compiling our project, creating a WAR file, copying the WAR file, and restarting the server. With the magic of Maven, all you need to do is type this at the command line:

```
./run_jetty
```

And you're off to the races. This little script simply runs `mvn jetty:run` with one special parameter, which we'll look at in a moment. The real work is in the `jetty:run` command. This amazing little plug-in will compile our project, fire up a Jetty server

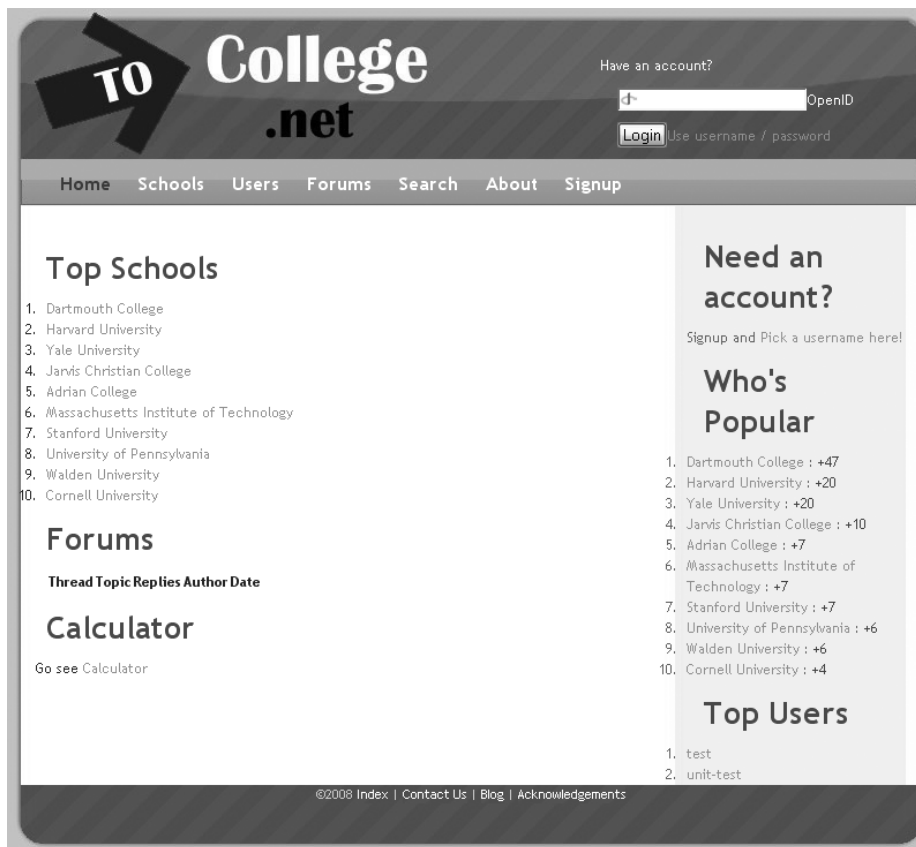
behind the scenes, and set the Jetty server working directory to our project. To read all about the plug-in, go to <http://www.mortbay.org/maven-plugin/>. The best part about it is that this server starts up amazingly quickly and will do a great job of recognizing changes to the underlying filesystem. This means that whenever we edit our templates, we won't need to restart the server at all. We can just reload the page.

---

**Note** If you get [ERROR] BUILD ERROR, . . . [INFO] trouble creating working compiler, this signals a problem with our GWT compiler plug-in. Just run it again, and it will work the second time (and thereafter). You'll need to do this after every time you run `mvn clean`, since the problem is with creating directories.

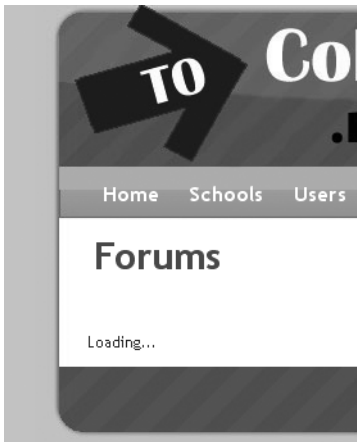
---

Let's see what this looks like. Turn your browser to <http://localhost:8080/>, and you should see something like Figure A-18.



**Figure A-18.** *It's alive!*

You can see that Spring MVC is set up correctly. It's selecting the test data out of the database and rendering it using our FreeMarker templates. Now, let's click over to the Forums page, which uses GWT (see Figure A-19).



**Figure A-19.** GWT is not really loading.

You can see that our GWT widget isn't loading. How come? Well, if you look at the source or use Firebug to check out the request for the page, you'll see that the page isn't able to find the `com.apress.progwt.Interactive/com.apress.progwt.Interactive.nocache.js` JavaScript code that we need to start GWT. This is just a path problem. By default, Jetty will start up and assume that `src/main/webapp` is the root of our web application. That's great because it means that we'll be able to change our FreeMarker templates in place and reload immediately. The only problem is that our GWT compiler plug-in will put the GWT files under the target directory. Jetty doesn't look there by default, so it won't be able to serve the GWT resources. Our solution is to just run things in two modes. One mode when we want to see GWT and one when we're just working with FreeMarker templates. If you open `pom.xml`, you'll see the Jetty plug-in, just like in Listing A-7.

**Listing A-7.** *ProGWT/pom.xml*

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <version>6.1.7</version>
  <!--skipped log dependency-->
  <configuration>
```

```
<webAppSourceDirectory>${webdir}</webAppSourceDirectory>
<scanIntervalSeconds>9999</scanIntervalSeconds>
<contextPath>/</contextPath>
<systemProperties>
  <systemProperty>
    <name>maven.test.skip</name>
    <value>>true</value>
  </systemProperty>
</systemProperties>
</configuration>
</plugin>
```

Notice the `webAppSourceDirectory` entry and the `webdir` parameter it takes. All that our scripts do is supply a location for this parameter. To run Jetty normally, as we did previously, we'll just invoke it as in Listing A-8.

#### Listing A-8. *run\_jetty*

```
mvn -Dwebdir=src/main/webapp jetty:run
```

If we want to see our GWT code right in the web site, we'll need to change this directory to be the directory that we compiled the GWT files to, as shown in Listing A-9.

#### Listing A-9. *run\_jetty\_gwt*

```
mvn -Dwebdir=target/ProGWT-1.0-SNAPSHOT jetty:run
```

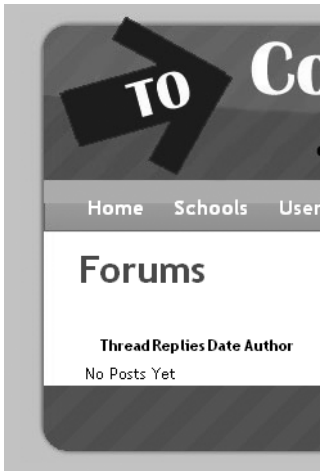
This might seem like it would be a bit of a pain, but I find that in practice I'm usually doing one thing or the other, and it's really not that bad. The Jetty startup time is so fast that there's really not much of a difference either way.

---

**Tip** If you're using GWT hosted mode, you can just use the `./run_jetty` command and still get support for changing templates without a reload. Hosted mode will take care of all the GWT bits and will only be using the server for the RPC services it provides. These services work the same no matter how we invoke Jetty.

---

Just to prove that we're keeping it real here, let's look at that forum page after we've run the `./run_jetty_gwt` command (see Figure A-20).



**Figure A-20.** GWT loaded in server mode

Fantastic! Obviously, there are no forum posts yet, but you can see that we've made it past the loading message, and our GWT widget has started up correctly.

## Initialize the Search Engine

In Chapter 9, we went over our use of the Lucene full text search engine, and you saw how Lucene would use a Hibernate interceptor to keep the search indexes up to date. This functionality starts automatically, but there is one thing we'll need to do to initialize Lucene: run the indexing process. While our interceptors are sufficient to keep everything up to date once we start, right now, we've got 20 schools in the database, and we need to index them. This is a simple call in Java Land, but to make this easier to use, I've created a page at <http://localhost:8080/site/secure/extreme/scripts.html> where you'll be able to perform this index simply by clicking.

Of course, we don't want everyone to be able to do this. That's why this `secure/extreme` URL is protected and available to only users that have supervisor access. Luckily, the test user that we created in the SQL initialization scripts is set up as a supervisor, so you can just login as test with password testaroo, and you'll have access to this scripts file. Click "Index search", and you'll be all set.

---

**Note** For more details on supervisor authority in ToCollege.net, see the `getAuthorities()` method in the `ServerSideUser` class and the `applicationContext-acegi-security.xml` file.

---

## Mac-Specific Launch Configuration

If you're on Mac OS X, there are a couple of things you'll need to be aware of. We already noted that to run hosted mode you'll need to include a `-XstartOnFirstThread` argument to the JVM. ToCollege.net takes care of this in its Mac-specific launch profiles. There's a second Mac-specific feature in the launch configuration that we haven't touched on yet.

The second thing you're going to need to take into account is a particular security "feature" of the Mac browser. In our Windows setup, hosted mode runs on port 8888, and Jetty runs on port 8080. Our GWT application makes calls from one to the other without any problem. When we move to a Mac, however, this same configuration triggers a security exception, because the ports don't match. I was unable to find a way to get around this restriction, but there's an easy solution. We can simply run in `noserver` mode. This mode tells GWT hosted mode not to bother using an internal Tomcat server and to simply rely on an existing server to supply the GWT host page. Since we were already running a server, this is no big deal. We just need to let GWT know where the server is. No-server mode is achieved with the following arguments to our GWT hosted mode:

```
-out www -noserver -port 8080 com.apress.progwt.Interactive/std/CollegeApp.html
```

These arguments are specified in the file `ToCollege-CollegeApp-Mac.launch`. The important one is the final argument, which tells GWT where the host page is located. To make this work, you'll need to run the `run_jetty_gwt` command instead of the `run_jetty` command so that this directory actually is where it's supposed to be.

## Developing with ToCollege.net

At this point, you should be all set to work with the ToCollege.net codebase. Feel free to make modifications and tweaks to see what happens. In general, you'll want to just execute `./jetty_run` and then work in Eclipse hosted mode. That way, you'll be able to get all of the wonderful hosted mode debugging support that GWT provides you. Whenever you need to change the server, you can just kill that process and restart. Every few hours of development, it's a good idea to execute `./jetty_run_gwt` and open the project in different browser, just so you can be sure that everything is looking like you want it to across browsers, and you're not going down any IE-specific CSS roads.

---

**Tip** Anytime you run `mvn clean`, you should do an Eclipse clean operation as well. Eclipse will assume that it's the only one working with your files, and it can get quite confused if you clean out the target directory without telling it.

---

## Creating a WAR file

To create a WAR file from ToCollege.net, all you need to do is run `mvn package`. Because Maven knows so much about the configuration of our project, it will be able to properly copy all of the files from the `resource` and `webapp` directories into the WAR, as well as all of the JARs from the repository and the compiled classes. We don't need any custom Ant scripts to put this WAR together. This is another nice example of the utility of Maven.

It is sometimes handy to be able to create WAR files from the repository even if all your tests aren't passing. Typically, Maven insists that we have all of our tests working in order to do this, but we can beg forgiveness for our sins with the following Maven parameter:

```
mvn -Dmaven.test.skip=true package
```

This is a handy line to know about, and it will tell Maven to skip the testing phase of the life cycle altogether.

## Integrating Maven and GWT Compilation

One thing that we needed to add to our Maven `pom.xml` file in order to get this working was to schedule a GWT compile as part of the build process. That's not part of the default Maven process, but it's no problem for a plug-in to hook in to this life cycle and extend it.

---

**Note** There's nothing you need to *do* here to get ToCollege.net running. We're just looking at the guts of integrating GWT compilation into a Maven build process.

---

There are a couple maven and GWT plug-ins to choose from. We'll use an extension of the `xi8ix maven-gwt` plug-in (<http://code.google.com/p/xi8ix>). I like this plug-in because it's dead simple. After reading the MOJO code, it's easy to tell just what's happening inside. While the `gwt-maven` plug-in available at <http://code.google.com/p/gwt-maven/> looks like it would get the job done as well, but I'm not a fan of how GWT JARs are included in the project. It does have some advanced functionality, however, so it's definitely worth a look.

All you would normally need to do to get the `xi8ix` plug-in set up is to follow the directions at <http://code.google.com/p/xi8ix/wiki/Building>. This basically comes down to running the following commands:

1. `svn checkout http://xi8ix.googlecode.com/svn/trunk/ xi8ix`
2. `cd xi8ix/xi8ix-gwtc/trunk`
3. `mvn install`

There are just a few issues with this. First, this plug-in has requirements on GWT 1.4.60, and we'd like to use GWT 1.5 builds. Additionally, there are some unresolved open issues with this project for which we'll need fixes. Specifically, we need the ability to use more memory for our builds. I've tried to upload patches for this to the project, but it seems that it's not being maintained. What I've done in lieu of getting these changes incorporated is to include an updated JAR in the ToCollege.net code. We installed this when we ran `install-all` previously. Let's take a look at the code in the `ToCollege.net` `pom.xml` that adds and configures this plug-in; see Listing A-10.

**Listing A-10.** *ProGWT/pom.xml*

```
<build>
  <plugins>
    <plugin>
      <groupId>org.xi8ix</groupId>
      <artifactId>xi8ix-gwtc</artifactId>
      <version>1.3.3.2</version>
      <executions>
        <execution>
          <phase>compile</phase>
          <goals>
            <goal>gwtc</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <modules>
          <module>com.apress.progwt.Interactive</module>
        </modules>
        <maxMemory>512</maxMemory>
        <webappDirectory>
          ${project.build.directory}/${project.build.finalName}
        </webappDirectory>
      </configuration>
    </plugin>
  </plugins>
</build>
```

This is all we need to do to latch into the Maven build process and perform some work of our own. The build will kick out a directory called `com.apress.progwt.Interactive` with all of the GWT compiled classes inside it. You can see that we needed to specify the module that we wanted to compile. This plug-in has support for multiple modules, so

just add more `<module>` tags as required. Finally, you can see the `maxMemory` line, which was the impetus for our customization of this plug-in.

## Summary

Hopefully, this appendix has got you started with ToCollege.net, and hopefully, you'll appreciate having 16,000 lines of functioning source code at your disposal. Remember to check out the ToCollege.net site from time to time to see what's happening. This is a living project, and I'm planning to keep it up to date as GWT and the host of technologies that we're using moves forward. I'll try to do a good job of refining and improving the code as I go along so that you can see how the site evolves. Of course, since the site is version controlled, you'll always be able to check out the version that is just as described in this book.

That's it! Thanks for reading the book and congratulations on making it through. Please feel free to ask questions on the ToCollege.net site.