Task-Oriented Solutions to Over 175 Common Problems

Covers Eclipse 3.0

Eclipse Cookbook™

O'REILLY®

Steve Holzner

CHAPTER 6

Using Eclipse in Teams

6.0 Introduction

Professional developers frequently work in teams, and Eclipse is up to the task. Eclipse supports the Concurrent Versions System (CVS) for this purpose. If you're working in a team, you have to coordinate your development work with others to avoid conflicts. You're all sharing the same code, which means your work of genius might be destroyed unintentionally by someone else's thoughtless efforts.

Source control precludes those kinds of problems because it controls access to shared code in a well-defined way. Besides controlling access to code, source control maintains a history of changes so that you can restore the code from earlier versions. Because it maintains a history of your code, not only can you restore code against earlier versions, but you can also compare the current code to earlier versions to see the differences at a glance.

Like much else in the Java world, CVS is an open source project. CVS first appeared in 1986, when it was a set of Unix shell scripts; it wasn't until 1989 that dedicated CVS software first appeared. Today, CVS is available on many operating systems across the board, from Unix and Linux to Windows.

For details on CVS, take a look at *http://www.cvshome.org*.

The CVS *repository* is where developers store code files to be shared. To retrieve a file from the repository, you check that file out of the repository. When you want to store your newly changed version of the file, you commit it to the repository. Refreshing your copy of the code from the repository is called *updating* it.

CVS also has slightly different terminology than Eclipse; what's a *project* to Eclipse is a *module* to CVS. Each module gets its own directory in the repository, making it easier to separate modules. Standard projects also are called *physical modules*, while *logical* or *virtual modules* are collections of related resources.

How many copies of your code are available to be checked out at once? That depends on which repository model you use. Here are the options:

Pessimistic locking
: Sequential access. With this type of locking, only one developer can check out a file at a time. When the file is checked out, the file is locked. It's possible for someone else to check out read-only copies of the file, but not to change the original. Access is sequential.

Optimistic locking
: Random access. With this type of locking, developers can check out and modify files freely. When you commit changed files, the repository software *merges* your changes automatically. If the merge operation has issues, the software will flag them and ask you to resolve the problems.

CVS uses optimistic locking by default (some CVS software also supports pessimistic locking). We'll be using optimistic locking here, which is what Eclipse supports. You use a CVS server to handle the actual file manipulation, as we'll do in this chapter.

CVS also automatically assigns a version number to each file when it's committed. When you first commit a file, it's version 1.1 (1.0 on some CVS installations). The next time, the version number is 1.2, and so on. When you update your code locally, Eclipse doesn't just overwrite your local version of a file. Instead, it merges the changes with your local file in an intelligent way. If conflicts exist, it'll insert special CVS markup to make the conflicting lines stand out, and those conflicts will have to be handled before running the code. Usually updates are smooth, but if there are a lot of conflicts because there's been a lot of work on the file or you haven't updated in some time, it can take a while to unravel.

CVS also enables you to support multiple development streams, called *branches*, in the same module. The main development stream in a module is called the *head*, and branches are forks that can diverge from that main stream. For example, a branch can represent a beta version of the project, or some new capability you're adding to your code that you want to test first.

6.1 Getting a CVS Server

Problem

You want to start working with CVS and need to install a CVS server.

Solution

You might already have a CVS server installed; if not, you can download one.

Discussion

Today, most Linux and Unix installations come with a CVS server as part of their standard distribution. To check if you have a working CVS installation, type `cvs --help` on the command line; you should see a list of help prompts. If you can't find a CVS server, download one from *http://www.cvshome.org*. On larger systems, talk to the support techs if you can't find a CVS installation.

If you're running Windows, you can find a number of CVS servers available for download. For example, the venerable CVSNT is available for free from *http://www.cvsnt.org*. Just run the executable to install it.

A variety of CVS servers are available, and they all come with their own installation instructions. I'm not going to reproduce those installation instructions here, having been burned by that in the past as new versions—with totally different installation instructions—appeared. Usually, installation is not difficult once you've downloaded the server you want. Just check the instructions that come with the download. And bear in mind that if the install is too complex, and things aren't working, other CVS servers are always available.

See Also

Chapter 4 of *Eclipse* (O'Reilly).

6.2 Creating a CVS Repository

Problem

You need to create a CVS repository to store code to share with others.

Solution

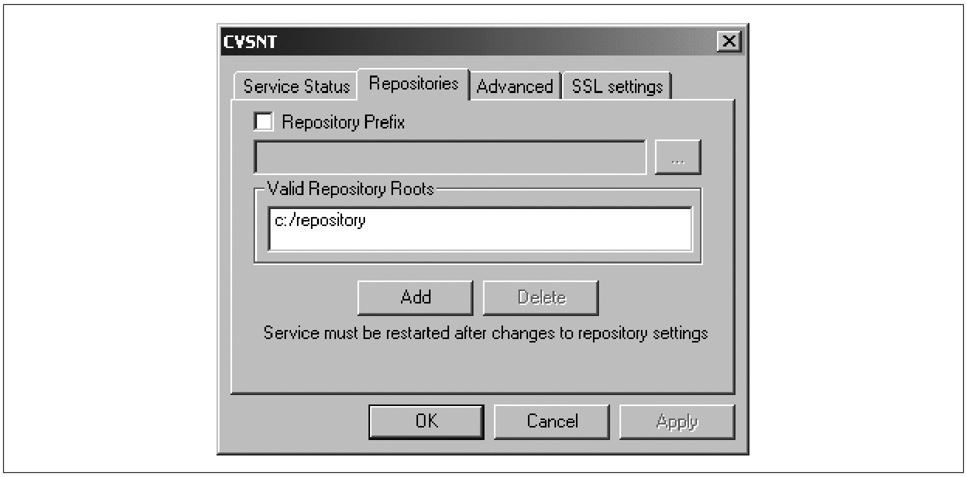
In Linux and Unix, use the command `cvs -d` *`path`* `init`, where *`path`* gives the location of the directory you want to use as the repository. In Windows CVSNT, use the Repository tab's Add button to add a new repository.

Discussion

After installing a CVS server, you need to create a repository in which to store shared code. In Linux and Unix, you can enter `cvs -d` *`path`* `init` at the command prompt, where *`path`* is the location of the repository.

When creating a repository, bear in mind that the permissions and ownership for *path* have to allow access to all members of the development team.

In Windows, CVSNT runs as a Windows service. You start it from the Start menu, selecting the Service control panel item from whatever program group you've added it to. This opens the CVSNT control panel shown in Figure 6-1. Click the Repositories tab in the CVSNT control panel, click the Add button, enter the path of the new repository directory, and click OK. In the figure, we're using the directory *c:/ repository* as the CVS repository.



c:/repository

Service must be restarted after changes to repository settings

Figure 6-1. Selecting a repository

You start the server by selecting the Service control panel item from the program group to which you've added CVSNT, opening the CVSNT control panel. Click the Start button in both the "CVS Service" and "CVS Lock Service" boxes, which will make CVSNT display the word `Running` in both of those boxes, as shown in Figure 6-2.

See Also

Recipe 6.4 on storing a project in a CVS repository.

6.3 Connecting Eclipse to a CVS Repository

Problem

You want to connect Eclipse to a CVS repository.

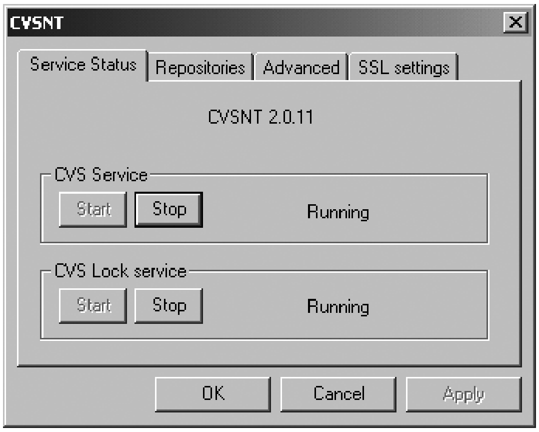


Figure 6-2. Running CVSNT

Solution

In Eclipse, open the Repositories view, right-click that view, and select New → Repository Location, opening the Add CVS Repository dialog. Enter the required information, and click OK.

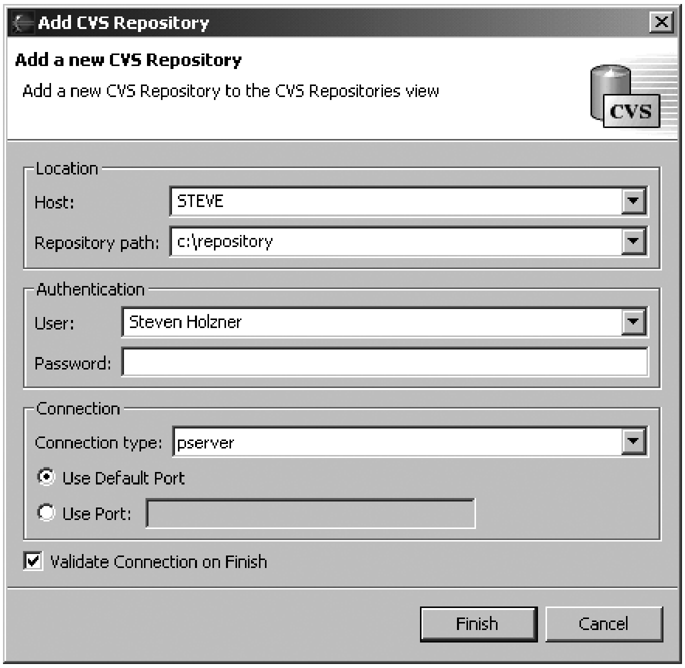
Discussion

You have to establish a connection from Eclipse through the CVS server to the CVS repository before working with that repository. First, make sure your CVS server is running.

To connect Eclipse to the CVS repository, select Window → Open Perspective → Other, and select the CVS Repository Exploring perspective. After you do this the first time, Eclipse adds this perspective to the Window → Open Perspective submenu and also adds a shortcut for this perspective to the other perspective shortcuts at the extreme left of the Eclipse window.

When the CVS Repository Exploring perspective opens, right-click the blank CVS repositories view at left, and select New → Repository Location, opening the Add CVS Repository dialog shown in Figure 6-3.

In the Add CVS Repository dialog, enter the name of the CVS server, often the name of the machine hosting the CVS server, and the path to the CVS repository. To connect to the CVS server, you'll also need to supply a username and password, as shown in Figure 6-3 (in this case we're using integrated Windows NT security, so no password is needed). You can use two connection protocols with CVS servers, SSH (secure shell) and `pserver`. We'll use `pserver` here.



Add a new CVS Repository to the CVS Repositories view

Location
Host: STEVE
Repository path: c:\repository

Authentication
User: Steven Holzner
Password:

Connection
Connection type: pserver
(●) Use Default Port
( ) Use Port:

[✓] Validate Connection on Finish

Figure 6-3. Connecting Eclipse to a CVS repository

pserver is a CVS client/server protocol that uses its own password files and connections. It's more efficient than SSH but less secure. If security is an issue, go with SSH.

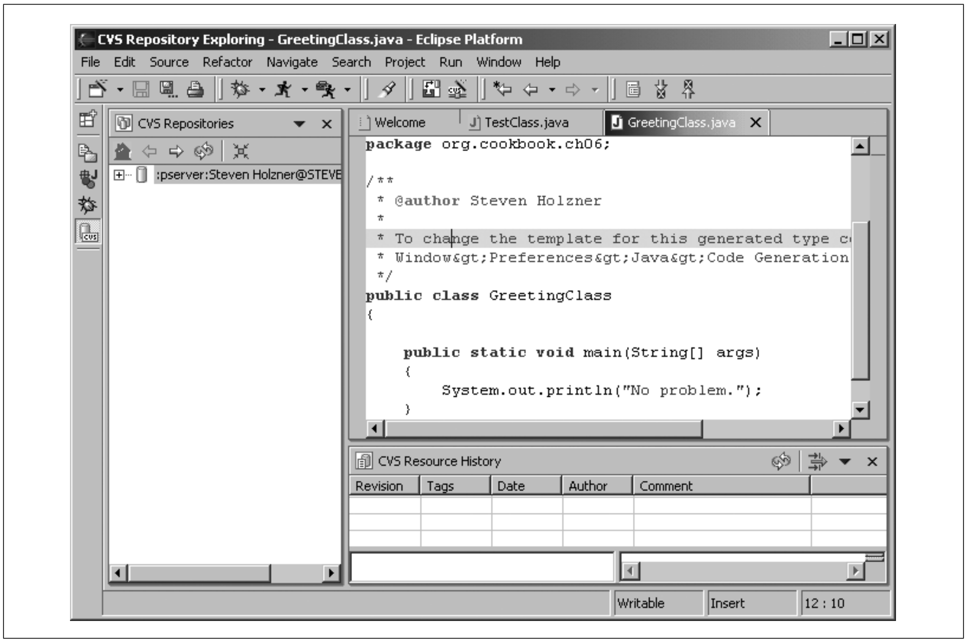
Click Finish after configuring the connection. The new connection to the CVS server should appear in the CVS Repositories view, as shown in Figure 6-4.

A public CVS server is available that gives you access to the code for Eclipse; go to *:pserver:anonymous@dev.eclipse.org:/home/eclipse*.

If you wish, you can see what commands Eclipse sends to the CVS server. To do so, open the CVS console by selecting Window → Show View → Other → CVS → CVS Console. The CVS Console view will appear (this view overlaps the standard Console view).

Eclipse 3.0

Eclipse 3.0 also supports CVS SSH2 in addition to the pserver and SSH protocols. You can enable SSH2 in the SSH2 Connection Method preference page (right-click a project and select Team → CVS → SSH2 Connection Method). All CVS server connections of type extssh will use SSH2 from that point on.



CVS Repository Exploring - GreetingClass.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
CVS Repositories
:pserver:Steven Holzner@STEVE
Welcome
TestClass.java
GreetingClass.java
CVS Resource History
Revision
Tags
Date
Author
Comment
Writable
Insert
12 : 10

```
package org.cookbook.ch06;

/**
 * @author Steven Holzner
 *
 * To change the template for this generated type c
 * Window&gt;Preferences&gt;Java&gt;Code Generation
 */
public class GreetingClass
{

    public static void main(String[] args)
    {
        System.out.println("No problem.");
    }
```

Figure 6-4. A new repository created in the CVS Repositories view

See Also

Chapter 4 of *Eclipse* (O'Reilly).

6.4 Storing an Eclipse Project in a CVS Repository

Problem

You have an Eclipse project you want to store in a CVS repository to make it available to other developers.

Solution

Right-click the project you want to share, and select Team → Share Project. Follow the directions in the Share Project with CVS Repository dialog.

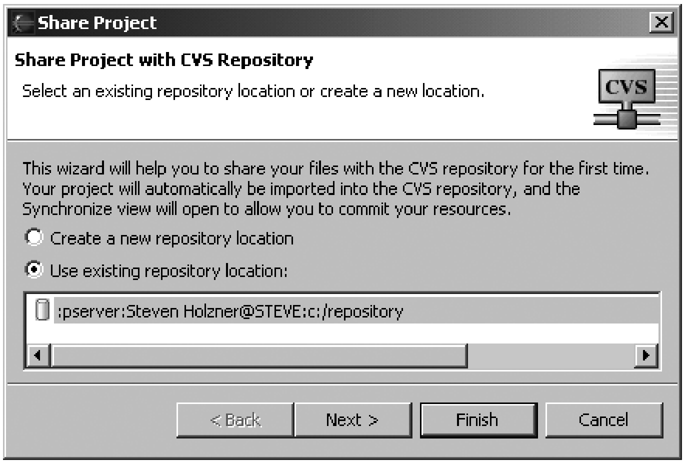
Discussion

As an example, we'll create a project here and add it to a CVS repository. The code for this example project, *GreetingApp*, appears in Example 6-1. All this code does is display the message `No problem.`.

Example 6-1. The GreetingApp project

```
package org.cookbook.ch06;

public class GreetingClass
{

    public static void main(String[] args)
    {
        System.out.println("No problem.");
    }
}
```

To add this project to the CVS repository, open the Java perspective, right-click the project, and select Team → Share Project. This displays the Share Project with CVS Repository dialog, as shown in Figure 6-5.


Share Project

Share Project with CVS Repository

Select an existing repository location or create a new location.

This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Synchronize view will open to allow you to commit your resources.

○ Create a new repository location

● Use existing repository location:

:pserver:Steven Holzner@STEVE:c:/repository

Figure 6-5. Sharing a project with a CVS repository

Make sure the "Use existing repository location" radio button is selected, and select the repository you want to use. Click Finish to add the project to the CVS repository. This creates a CVS module with the same name as the Eclipse project.

If you want to give the created CVS module a different name, click Next instead of Finish, enter the name of the CVS module you want to create, enter a new module name, and click Finish.

This adds the project to the CVS repository and also opens a Synchronize view in Eclipse which overlaps with the Console view (more on how to work with the Synchronize view later in this chapter; because there's nothing to synchronize with at this point, it's not of much use to us now).

See Also

Recipe 6.5 on committing files to the CVS repository.

6.5 Committing Files to the CVS Repository

Problem

You've edited a file, saved your changes, and want to send it to the CVS repository so that others can access it.

Solution

Right-click a file, and select Team → Commit.

Discussion

In Recipe 6.4, you saw how to add a project to a CVS repository. To share your code, you have to check in code files. This requires two steps: first, you add a file to the CVS repository, which registers the file with the CVS server but doesn't actually upload it; then you commit the file, which uploads it to the repository.

Technically, the way to send files to the CVS repository is to add them to Eclipse's version control and then commit them. You do that by right-clicking the files and selecting Team → Add to Version Control. Then select Team → Commit to commit the files.

However, Eclipse gives you a shortcut here. To commit all the files in a project, right-click the project, and select Team → Commit. When you do, Eclipse displays the Add to CVS Version Control dialog. Click the Details button and check the checkboxes matching the files you want to add to CVS version control; Eclipse will list all the files in the project, including your Java source files and the *.project* and *.classpath* files. Then click Yes.

If you want to check in and check out projects as Eclipse projects, be sure to commit the *.project* and *.classpath* files.

Eclipse will prompt you for a comment for the set of files you're committing, giving you the chance to label those files. In this case, just enter some text, such as `The Greeting App`, as shown in Figure 6-6, and click OK.

You also can simply right-click an individual file and select Team → Commit. If the file is not yet under version control, Eclipse will ask if you want to add it; click Yes. Eclipse will display the same Commit dialog shown in Figure 6-6, enabling you to enter a comment for the file before it's committed.

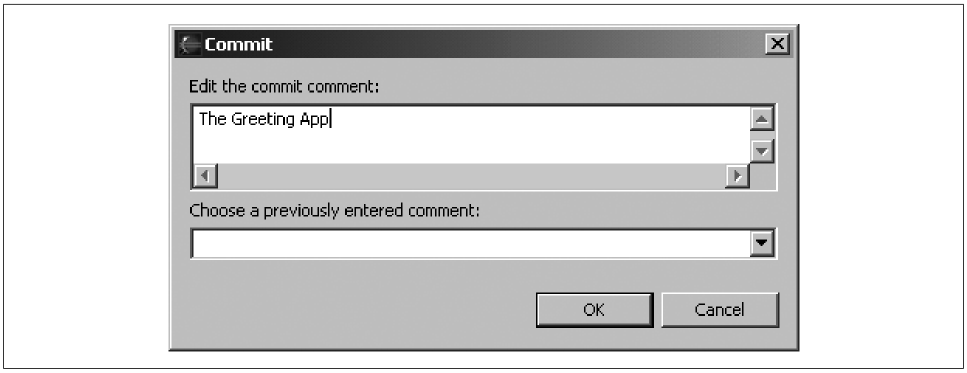


Figure 6-6. Committing files

When the file is committed, it's uploaded to the CVS repository and given a version number. Eclipse also will use a special decoration for files under version control if you tell it to do so; see Recipe 6.6.

See Also

Recipe 6.6 on visually labeling files under version control; Chapter 4 of *Eclipse* (O'Reilly).

6.6 Visually Labeling Files Under Version Control

Problem

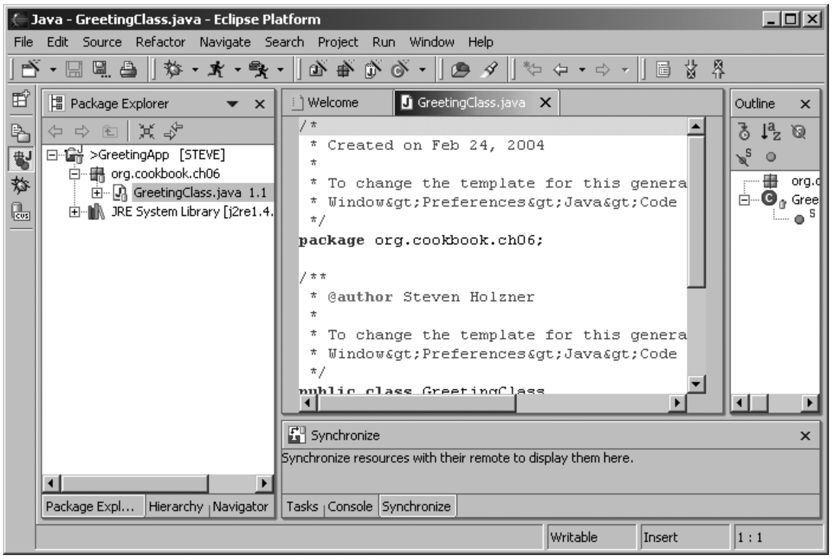
You want to see at a glance what files in a project are under version control.

Solution

Turn on CVS decorations by selecting Window → Preferences → Workbench → Label Decorations, check the CVS checkbox, and click OK.

Discussion

To make Eclipse indicate which files are under version control, you turn on CVS label decorations. Select Window → Preferences → Workbench → Label Decorations to open the Preferences dialog, then check the "CVS" checkbox, and click OK. This makes Eclipse add a gold cylinder to the icons of files under CVS version control, as shown in the *GreetingApp* project in Figure 6-7. Note also that files in the repository will have a CVS version number showing; that version is 1.1 here.


Java - GreetingClass.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
>GreetingApp [STEVE]
org.cookbook.ch06
GreetingClass.java 1.1
JRE System Library [j2re1.4.
Welcome
GreetingClass.java
Outline
org.c
Gree
Synchronize
Package Expl... Hierarchy Navigator
Tasks Console Synchronize
Writable Insert 1 : 1

```
/*
 * Created on Feb 24, 2004
 *
 * To change the template for this genera
 * Window&gt;Preferences&gt;Java&gt;Code
 */
package org.cookbook.ch06;

/**
 * @author Steven Holzner
 *
 * To change the template for this genera
 * Window&gt;Preferences&gt;Java&gt;Code
 */
public class GreetingClass
```

Synchronize resources with their remote to display them here.

Figure 6-7. Files under version control

See Also

Chapter 4 of *Eclipse* (O'Reilly).

6.7 Examining the CVS Repository

Problem

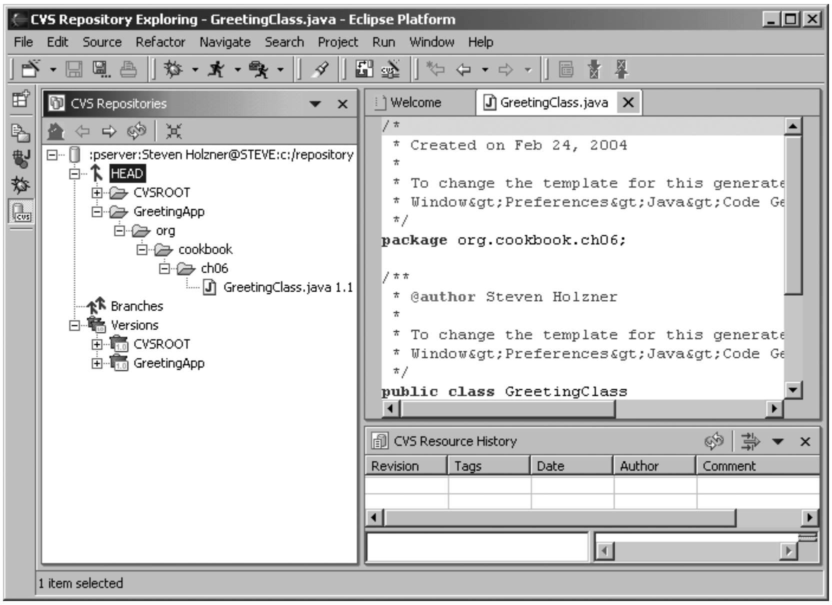
You want to explore the CVS repository from inside Eclipse.

Solution

Use the CVS Repository Exploring perspective. Open this perspective by selecting Window → Open Perspective → Other → CVS Repository Exploring. Or, if you've opened this perspective in the past—which means Eclipse will have added it to the Open Perspective menu—select Window → Open Perspective → CVS Repository Exploring.

Discussion

The CVS Repository Exploring perspective enables you to see what's inside the CVS repository. For example, you can see the entire *GreetingApp* project, all the way down to the *GreetingClass.java* file, in the CVS Repositories view at left in Figure 6-8.



```
/*
 * Created on Feb 24, 2004
 *
 * To change the template for this generate
 * Window&gt;Preferences&gt;Java&gt;Code Ge
 */
package org.cookbook.ch06;

/**
 * @author Steven Holzner
 *
 * To change the template for this generate
 * Window&gt;Preferences&gt;Java&gt;Code Ge
 */
public class GreetingClass
```

Figure 6-8. The CVS repository perspective

This project, *GreetingApp*, is in the repository's `HEAD` section, which is the main development stream.

Also note the *CVSROOT* directory, which holds CVS administrative data; the Branches node, which holds any files in other branches of development; and the Versions node, which holds explicitly labeled versions.

Eclipse 3.0

In Eclipse 3.0, you can determine who's responsible for a bug. You right-click a file and select Team → Show Annotation to open a CVS Annotation view showing the selected file in the CVS Repositories view. When you select a line in the editor, the CVS Annotation view reveals who released that edit to the file.

See Also

Recipe 6.8 on checking projects out of a CVS repository; Recipe 6.13 on creating CVS branches; Chapter 4 of *Eclipse* (O'Reilly).

6.8 Checking Projects Out of a CVS Repository

Problem

Someone wants to check out a project of yours from the CVS repository.

Solution

In Eclipse, right-click a file in the CVS Repositories view, and click Check Out As. Then use the Check Out dialog to check out the item.

Discussion

If other people want to check out a module that you've stored in a CVS repository, they have to create a connection to the repository as we did earlier in this chapter. To do this, they should right-click the CVS Repositories view; select New → Repository Location; and enter the name of the CVS server, the location of the repository, the username, the password, and the type of connection.

They can then open the CVS Repositories view to explore the files in the repository. To check out the *GreetingApp* module, they can right-click the module in the Repositories view and click Check Out As from the context menu. Eclipse will open the New Project dialog and automatically create a new project corresponding to the CVS module.

If you're sharing an Eclipse project, and each CVS module has its own Eclipse *.project* file, you can select Check Out As Project from the Repositories view's context menu, which checks out an Eclipse project and adds it to the Package Explorer. Note that if your code isn't in a project of a kind that Eclipse can recognize, it will ask you what type of project to create.

See Also

Chapter 4 of *Eclipse* (O'Reilly).

6.9 Updating Your Code from a CVS Repository

Problem

You want to update your local code with the code in a CVS repository.

Solution

Right-click the file, and select Team → Update, then resolve any conflicts. Alternatively, if you just want to replace your version with what's in the CVS repository, right-click the file and select Replace With → Latest From HEAD.

Discussion

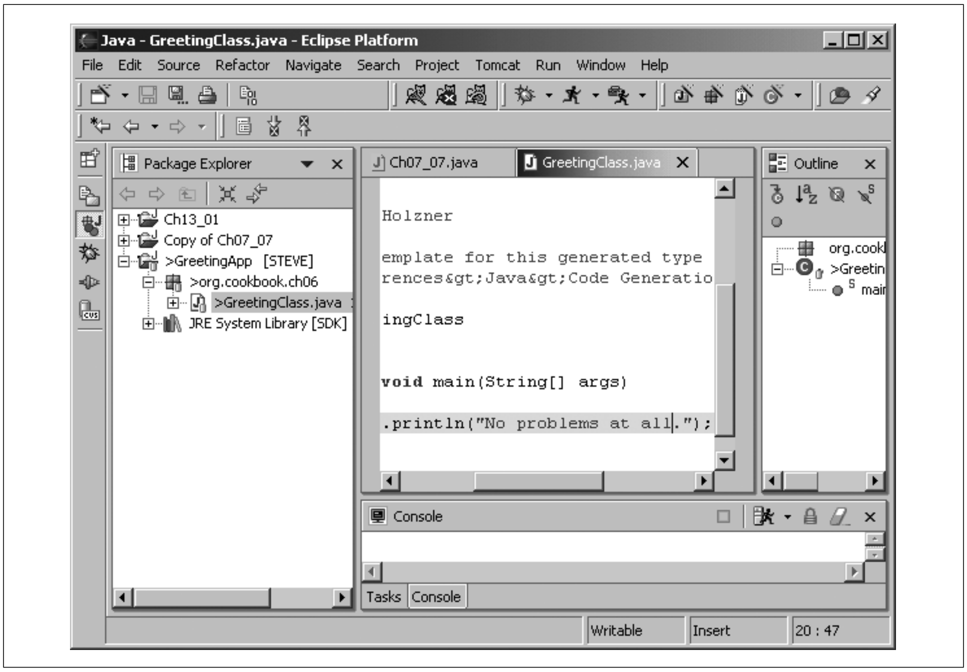
For example, say that someone checked out your code and changed this line:

```
public static void main(String[] args)
{
    System.out.println("No problem.");
}
```

to this:

```
public static void main(String[] args)
{
    System.out.println("No problems at all.");
}
```

When she makes these changes in his version of Eclipse, a > character appears in front of files that haven't yet been committed, as shown in Figure 6-9.

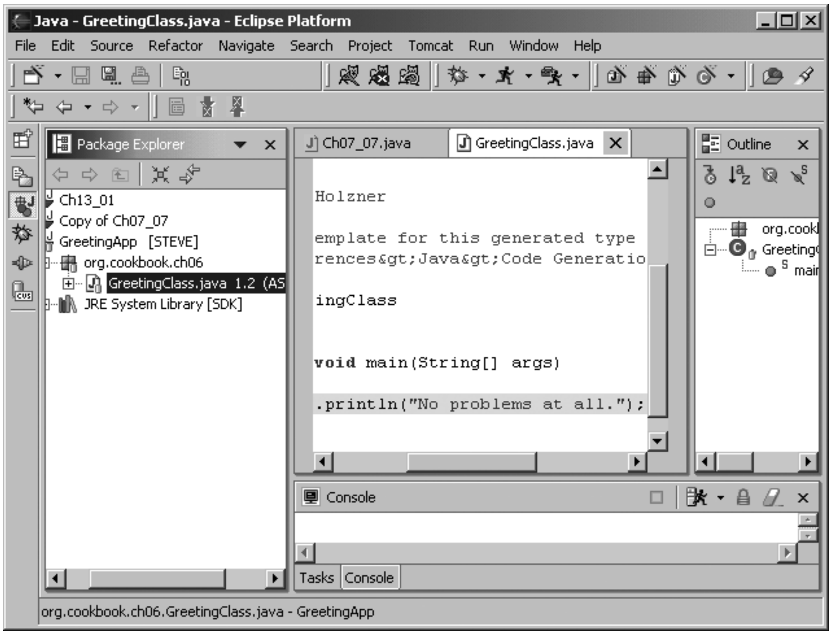


```
Holzner

emplate for this generated type
rences&gt;Java&gt;Code Generatio

ingClass


void main(String[] args)

.println("No problems at all.");
```

Figure 6-9. Outgoing changes ready

When she commits her changes, the latest version of *GreetingClass.java* in the CVS repository changes from 1.1 to 1.2, as shown in Figure 6-10.

To update your code with the most recent version of the code in the repository (which is now Version 1.2, as stored by the other developer), right-click the project or file in your version of Eclipse and select Team → Update. Doing so upgrades your version of the project's files to version 1.2—if there's no conflict.


Java - GreetingClass.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help
Package Explorer
Ch13_01
Copy of Ch07_07
GreetingApp [STEVE]
org.cookbook.ch06
GreetingClass.java 1.2 (AS
JRE System Library [SDK]
Ch07_07.java
GreetingClass.java
Outline
org.cook
Greeting
main
Console
Tasks Console
org.cookbook.ch06.GreetingClass.java - GreetingApp

```
Holzner

emplate for this generated type
rences&gt;Java&gt;Code Generatio

ingClass


void main(String[] args)

.println("No problems at all.");
```

Figure 6-10. A new version in the CVS repository

If you've changed this line of code yourself to something such as this:

```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
```

there will be a conflict with the new line in the new version, version 1.2, of the file. Eclipse will mark that conflict by listing both versions in your code with some added CVS markup such as this:

```
public static void main(String[] args) {
<<<<<<< GreetingClass.java
    System.out.println("No problems here.");
=======
    System.out.println("No problems at all.");
>>>>>>> 1.2
    }
}
```

It's up to you to handle these conflicts in your code (Eclipse is not going to compile your code until the CVS markup has been dealt with and removed). Letting Eclipse handle updates like this is one way to handle updates from the CVS repository, but if the changes are substantial, it's best to *synchronize* with the repository, an issue we discuss in Recipe 6.10.

Note that if you just want to *replace* your version of a file with the latest version of the file in the main development stream for the project in the CVS repository, right-click the file and select Replace With → Latest From HEAD.

6.10 Synchronizing Your Code with the CVS Repository

Problem

You've got a lot of changes from the version of a file in the CVS repository and want to get your code up to speed.

Solution

Right-click the project and select Team → Synchronize with Repository. Then take a look at the synchronization issues that Eclipse displays side by side.

Discussion

Synchronizing with the repository enables you to compare changes that have been made side by side in an easier format than the update merge format. For instance, say that the version of the code in the repository uses this code:

```
public static void main(String[] args)
{
   System.out.println("No problems at all.");
}
```

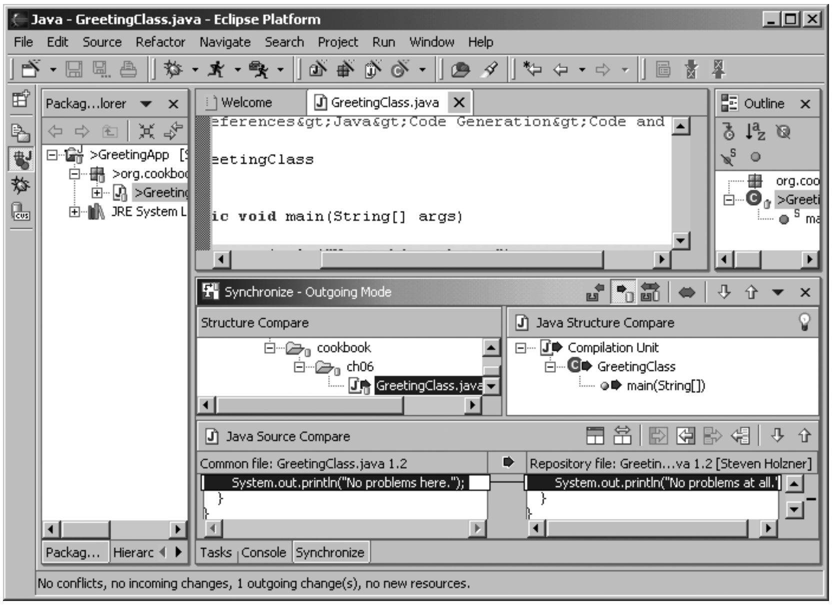
But you've changed that line of code to this:

```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
```

To synchronize your code with the version of the file in the repository, right-click the project and select Team → Synchronize with Repository. Then double-click the *GreetingClass.java* node in the Structure Compare view to take a look at the synchronization issues for that file. You can see the results in Figure 6-11.

Note the side-by-side comparison going on at the bottom of Figure 6-11, where your local file is being compared to that in the CVS repository. As shown in the figure, lines appear connecting the differences in the files.

You can use the up and down arrow buttons at right in the Java Source Compare view to navigate between changes. You also can use the two arrow buttons next to the navigation buttons to accept or make changes. The button with the left-facing arrow copies the current change from the repository to your local code, and the button with the right-facing arrow copies the current change from your local file to the



```
eferences&gt;Java&gt;Code Generation&gt;Code and
eetingClass

ic void main(String[] args)
```

```
    System.out.println("No problems here.");
  }
}
```

```
    System.out.println("No problems at all.");
  }
}
```

No conflicts, no incoming changes, 1 outgoing change(s), no new resources.

Figure 6-11. Synchronizing code

repository. After you've synchronized your version of the code with that in the repository, commit your changes to the repository.

Eclipse 3.0

Eclipse 3.0 has a handy way to look at changes in the local document as compared to the version of that document in the CVS repository. Right-click the Quick Diff ruler, and set the Set QuickDiff Reference item to CVS Repository. This makes the Quick-Diff bar compare recent changes to those in the CVS repository—very handy.

See Also

Recipe 4.6 on comparing files against local history; Recipe 4.7 on restoring elements and files from local history; Recipe 6.9 on updating code from a repository; Chapter 4 of *Eclipse* (O'Reilly).

6.11 Creating Code Patches

Problem

You need to coordinate your development with another team of developers using a patch they can install to update their code.

Solution

Create a code patch so that they can update their code. (Note that this is a *code patch*, not a binary patch. Eclipse can use this patch to update source code to match another version.)

Discussion

Say your version of the code displays the text "No problems here.":

```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
```

But the code the other team is using from the CVS repository displays "No problems at all.":

```
public static void main(String[] args)
{
    System.out.println("No problems at all.");
}
```

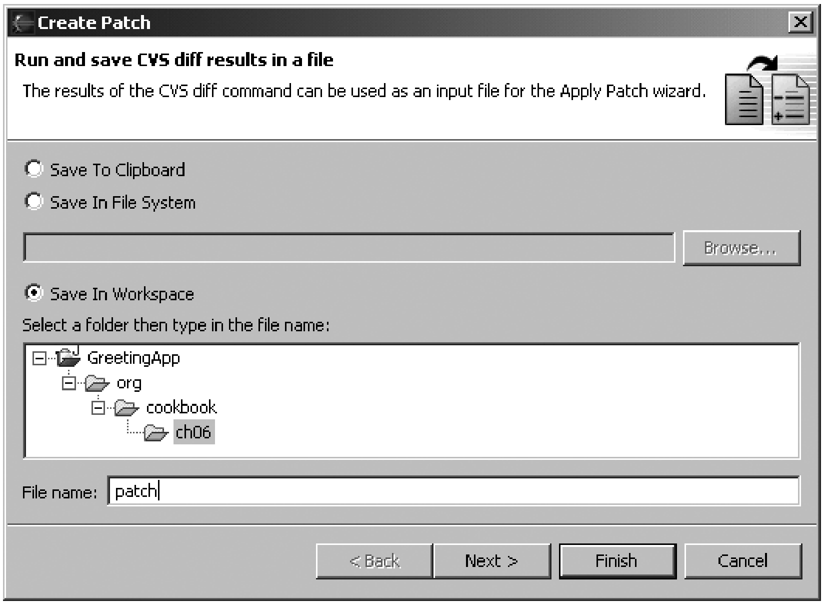
To update the other developers without changing version numbers, you can create a code patch. To create a code patch, Eclipse compares your local code to what's in the repository and creates a patch file holding the differences.

To create a code patch using your local version of a file as the version to which the patch will update the version in the repository, save your file locally, right-click it, and select Team → Create Patch, opening the dialog shown in Figure 6-12.

In this example, we'll save the file named *patch* in the current workspace, as shown in Figure 6-12. Click the Finish button to save the patch.

This creates the text file named *patch*. Here's what that file looks like; you can see the line to remove marked with a - and the line to add marked with a +:

```
Index: GreetingClass.java
===================================================================
RCS file: c:/repository/GreetingApp/org/cookbook/ch06/GreetingClass.java,v
retrieving revision 1.2
diff -u -r1.2 GreetingClass.java
--- GreetingClass.java    25 Feb 2004 16:34:07 -0000    1.2
+++ GreetingClass.java    25 Feb 2004 18:12:18 -0000
@@ -17,6 +17,6 @@

 public static void main(String[] args)
 {
-    System.out.println("No problems at all.");
+    System.out.println("No problems here.");
     }
 }
```


Create Patch

Run and save CVS diff results in a file

The results of the CVS diff command can be used as an input file for the Apply Patch wizard.

Save To Clipboard
Save In File System
Save In Workspace [✓]

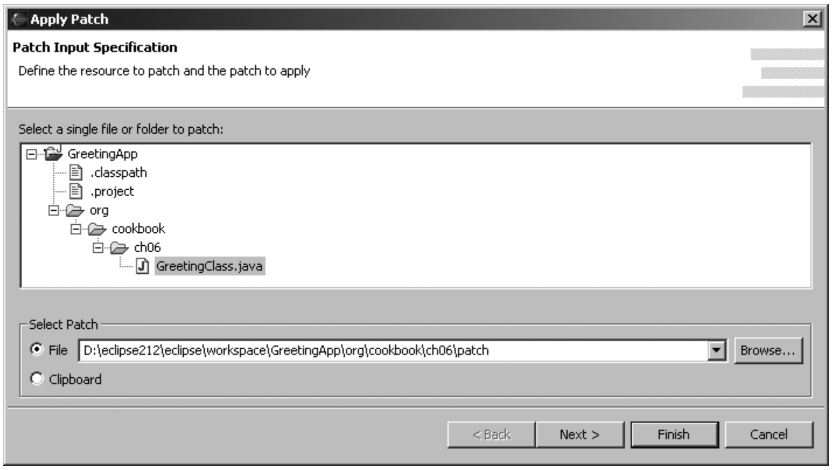
Select a folder then type in the file name:

GreetingApp
org
cookbook
ch06

File name: patch

Figure 6-12. Creating a new patch

To apply the new patch to code that has not yet been patched, right-click the file to be updated in Eclipse and select Team → Apply Patch, opening the dialog shown in Figure 6-13.


Apply Patch

Patch Input Specification

Define the resource to patch and the patch to apply

Select a single file or folder to patch:

GreetingApp
.classpath
.project
org
cookbook
ch06
GreetingClass.java

Select Patch
File [✓] D:\eclipse212\eclipse\workspace\GreetingApp\org\cookbook\ch06\patch
Clipboard

Figure 6-13. The Apply Patch dialog

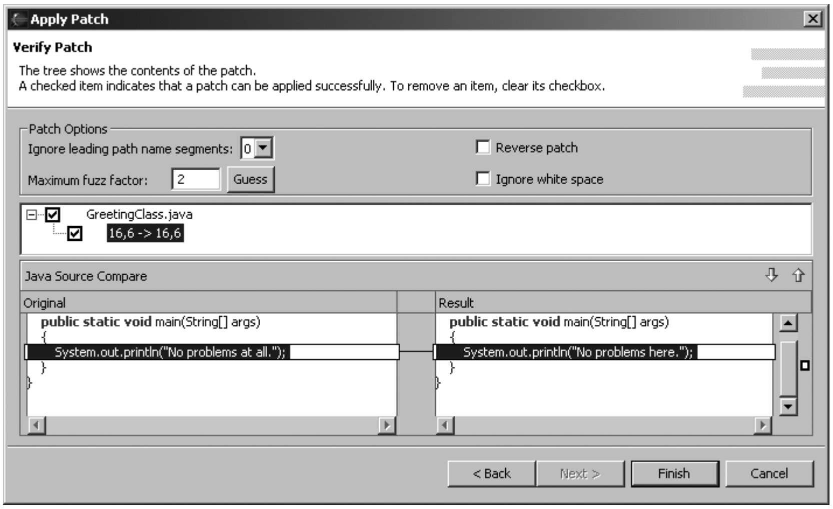
Click Next to open the dialog shown in Figure 6-14. In this dialog you can review the changes the patch will create in the local version of the file. As shown in the figure, Eclipse will change the line:

```
System.out.println("No problems at all.");
```

to:

```
System.out.println("No problems here.");
```

To apply the patch, click Finish.


Apply Patch

Verify Patch

The tree shows the contents of the patch.
A checked item indicates that a patch can be applied successfully. To remove an item, clear its checkbox.

Patch Options
Ignore leading path name segments: 0
Maximum fuzz factor: 2
☐ Reverse patch
☐ Ignore white space

☑ GreetingClass.java
☑ 16,6 -> 16,6

Java Source Compare

Original:
```
public static void main(String[] args)
{
    System.out.println("No problems at all.");
}
}
```

Result:
```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
}
```

Figure 6-14. Accepting a patch

Applying the patch makes this change to the code in the other team's installation of Eclipse, as shown in Figure 6-15. Note that the version number of the file was not changed, but the file was updated with the new code.

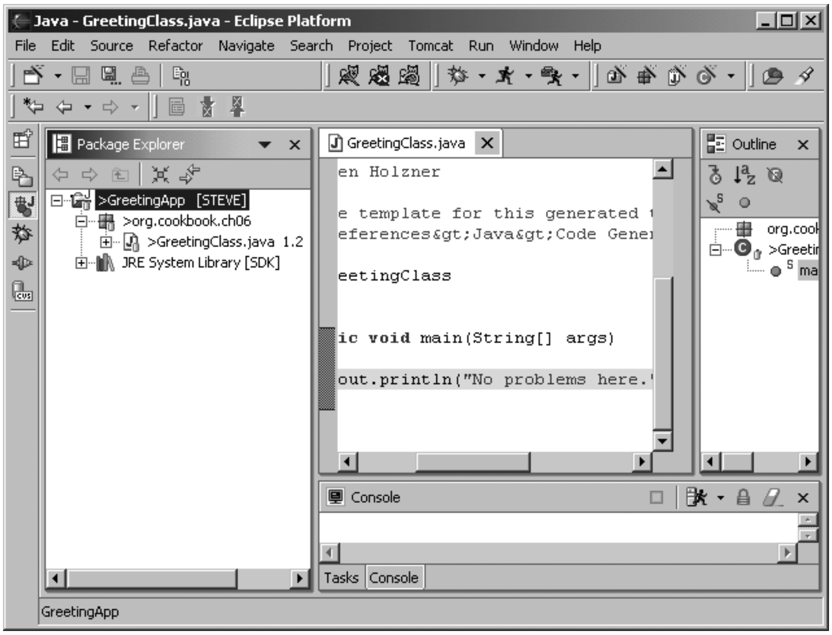
6.12 Naming Code Versions

Problem

You've got a milestone build of your project, and you want to save it by name in the CVS repository for easy reference later on.

Solution

Tag the project with a version name by right-clicking it and selecting Team → Tag As Version.



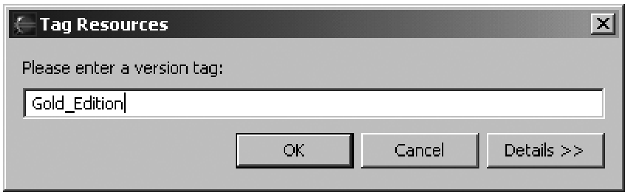
```
en Holzner

e template for this generated t
eferences&gt;Java&gt;Code Gene

eetingClass


ic void main(String[] args)

out.println("No problems here.'
```

Figure 6-15. Applying a code patch

Discussion

If you've created a milestone build of your project, you might want to save it by name. Doing so makes CVS store the tagged version so that you can access it by name later.

Right-clicking a project under version control, and selecting Team → Tag As Version opens the dialog shown in Figure 6-16. In this case, we're going to name our current version of the project `Gold_Edition`, as shown in the figure.

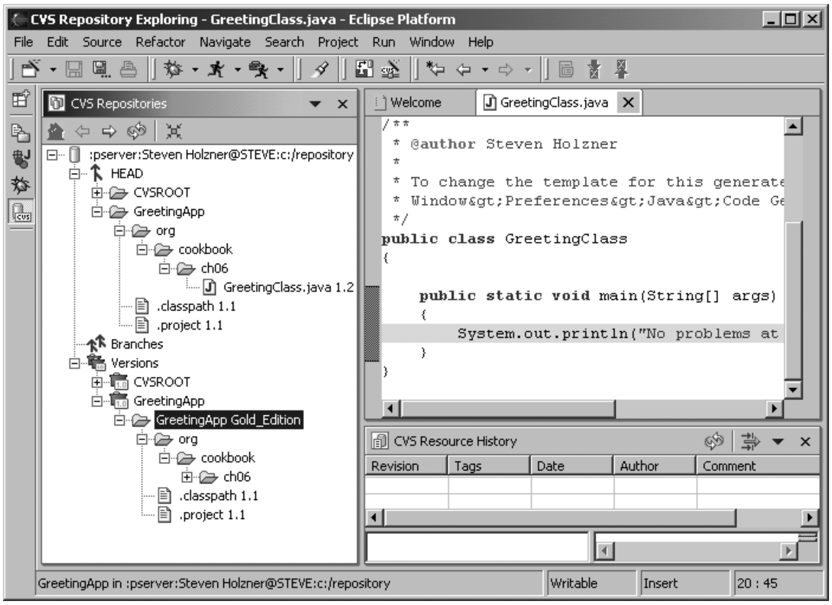

Tag Resources

Please enter a version tag:

Gold_Edition

Figure 6-16. Tagging a version of your code

Note that version labels must start with a letter, and they cannot include spaces or these characters: `` `$,.:;@|' ``. After tagging the current version with this name, you can find it in the Versions node in the CVS Repositories view, as shown in Figure 6-17.



```
/**
 * @author Steven Holzner
 *
 * To change the template for this generate
 * Window&gt;Preferences&gt;Java&gt;Code Ge
 */
public class GreetingClass
{


    public static void main(String[] args)
    {
        System.out.println("No problems at
    }
}
```

Figure 6-17. A new tagged version

You can check out a tagged version of a module by right-clicking it in the CVS Repositories view and selecting context menu items such as Check Out as Project, as with any other CVS module. Alternatively, you can right-click a project in the Package Explorer and select Replace With → Another Branch or Version, opening the dialog shown in Figure 6-18. Select the version with which you want to replace the current project, and click OK.

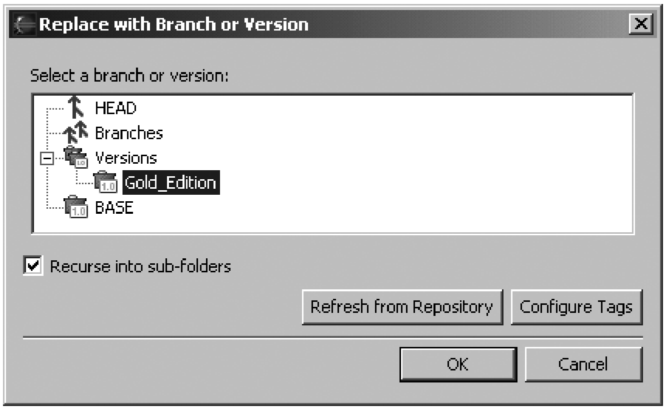


Figure 6-18. Accessing a milestone build

See Also

Recipe 6.4 on storing an Eclipse project in CVS; Recipe 6.5 on committing files to the repository.

6.13 Creating CVS Branches

Problem

You want to develop a new version of your code, such as a beta version, by creating a new branch in your development tree.

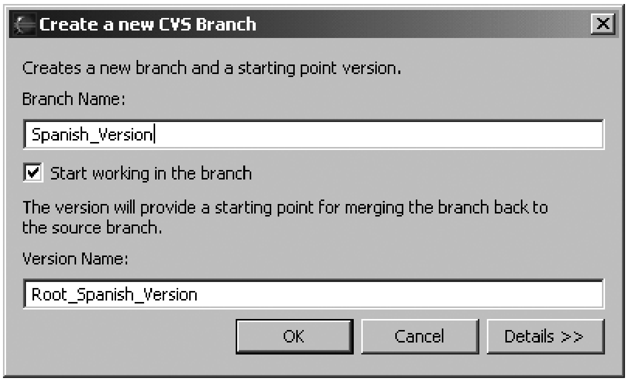
Solution

Add a new branch to your project's development tree by selecting Team → Branch.

Discussion

CVS also enables you to create new branches in your code's development tree. Such branches can act as alternate streams of development for your code; e.g., you might want to develop a new version of your code that uses prompts in another language.

To create a branch, right-click a project and select Team → Branch, which opens the Create a new CVS Branch dialog shown in Figure 6-19. In this example, we'll name the new branch `Spanish_Version`, as shown in the figure. At the same time, you can create a new version name for your code that will act as a reference, giving Eclipse a reference point for merging the branch into the main stream if you want to do that; Eclipse will suggest the name `Root_Spanish_Version` here.


Create a new CVS Branch

Creates a new branch and a starting point version.

Branch Name: Spanish_Version

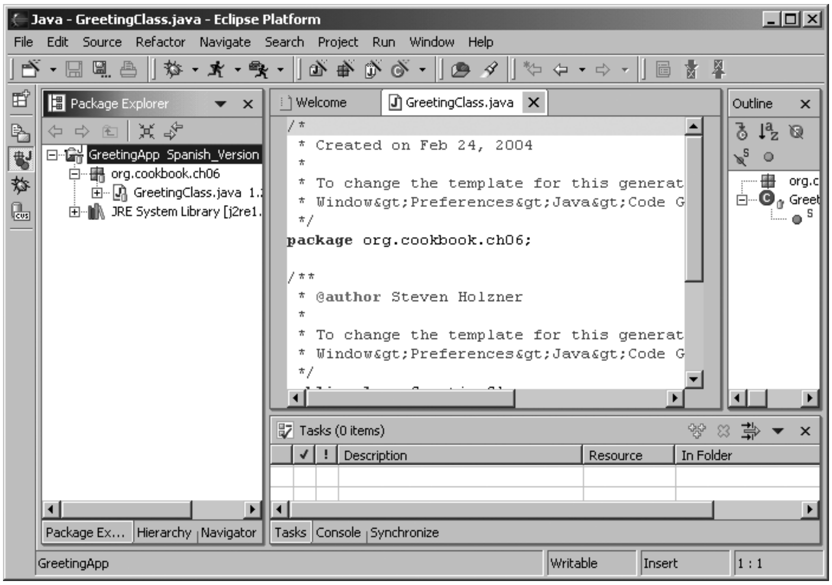
[✓] Start working in the branch

The version will provide a starting point for merging the branch back to the source branch.

Version Name: Root_Spanish_Version

Figure 6-19. Creating the Spanish_Version branch

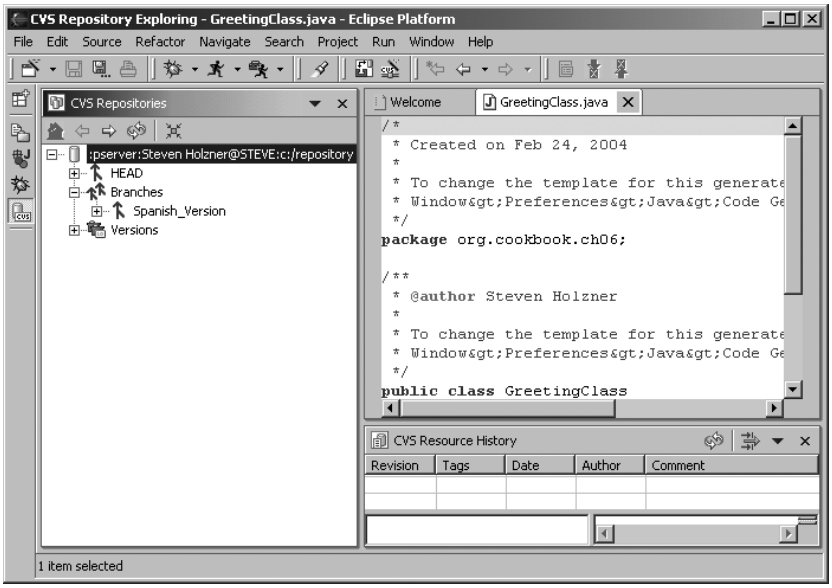
Clicking OK here opens the new branch in Eclipse, as shown in Figure 6-20 (note the project name in the Package Explorer).



```
/*
 * Created on Feb 24, 2004
 *
 * To change the template for this generat
 * Window&gt;Preferences&gt;Java&gt;Code G
 */
package org.cookbook.ch06;

/**
 * @author Steven Holzner
 *
 * To change the template for this generat
 * Window&gt;Preferences&gt;Java&gt;Code G
 */
```

Figure 6-20. A new branch

You can check out the new branch from the CVS Repositories view, as shown in Figure 6-21.

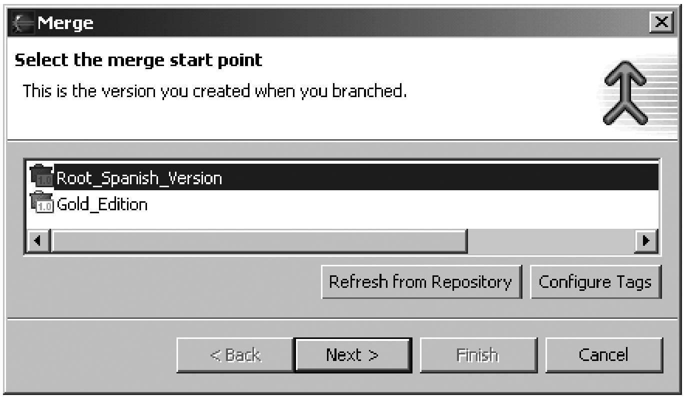


CVS Repository Exploring - GreetingClass.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
CVS Repositories
:pserver:Steven Holzner@STEVE:c:/repository
HEAD
Branches
Spanish_Version
Versions
Welcome
GreetingClass.java
CVS Resource History
Revision
Tags
Date
Author
Comment
1 item selected

```
/*
 * Created on Feb 24, 2004
 *
 * To change the template for this generate
 * Window&gt;Preferences&gt;Java&gt;Code Ge
 */
package org.cookbook.ch06;

/**
 * @author Steven Holzner
 *
 * To change the template for this generate
 * Window&gt;Preferences&gt;Java&gt;Code Ge
 */
public class GreetingClass
```

Figure 6-21. The new branch in the CVS Repositories view

You can merge branches back into the main development stream when needed. To do that, right-click the branch in the Package Explorer, and select Team → Merge, which opens the Merge dialog shown in Figure 6-22. Select the merge point for this operation—in this case, `Root_Spanish_Version`—and click Next.

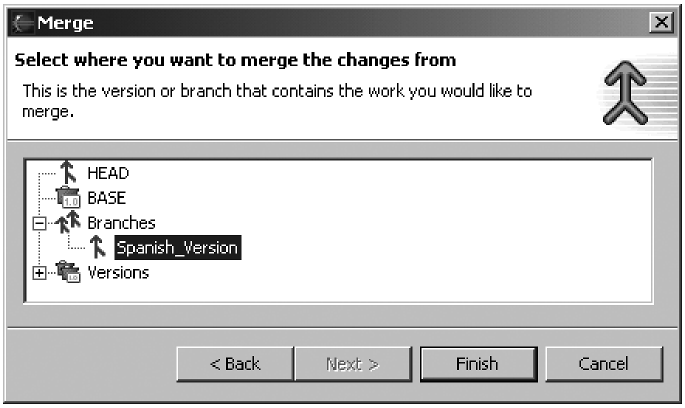

Select the merge start point

This is the version you created when you branched.

- Root_Spanish_Version
- Gold_Edition

Figure 6-22. Selecting a root version to merge to

The next dialog enables you to select the branch from which you want to merge. In this case, select `Spanish_Version`, as shown in Figure 6-23.


Select where you want to merge the changes from

This is the version or branch that contains the work you would like to merge.

- HEAD
- BASE
- Branches
  - Spanish_Version
- Versions

Figure 6-23. Merging a branch

Clicking Finish in this dialog completes the merge operation.