

Using JDeveloper with Subversion

A Developer's Guide for the JDeveloper Subversion VCS extension

10.1.3.0 July 2006

Contents

- [Introduction](#)
- [Installing the JDeveloper Subversion VCS extension](#)
- [Connecting to a Subversion repository](#)
- [Importing JDeveloper projects into Subversion](#)
- [Checking out files](#)
- [Adding and committing files](#)
- [Updating files](#)
- [Editing files](#)
- [Comparing and merging revisions of files](#)
- [Resolving conflicts](#)
- [Renaming files](#)
- [Removing files from Subversion control](#)
- [Appendix A: Installing Subversion client software](#)
- [Appendix B: Installing an additional Java helper library](#)
- [Appendix C: Connecting to a Subversion repository through a proxy server](#)
- [Appendix D: Creating a local Subversion repository](#)

Introduction

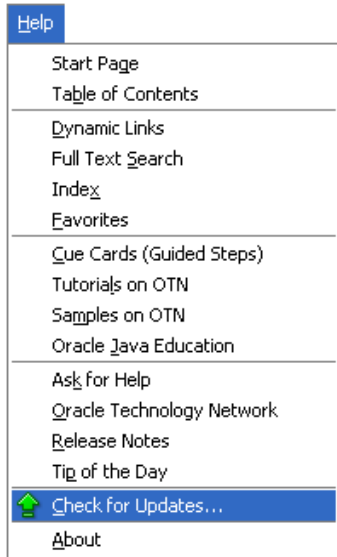
Subversion is a powerful, open source version control system designed as a replacement for the popular Concurrent Versions System (CVS). It offers many enhancements to CVS, such as versioned directories and metadata, and is architected for simpler, more flexible network access. Oracle offers an extension to the JDeveloper IDE, which automates much of the manual work involved with interacting with Subversion. For a rigorous introduction to Subversion, consult "Version Control with Subversion" at <http://svnbook.red-bean.com/>.

This guide takes you through the necessary steps to install Subversion and configure JDeveloper for use with it. You will then be taken through some common scenarios encountered when working with Subversion in JDeveloper.

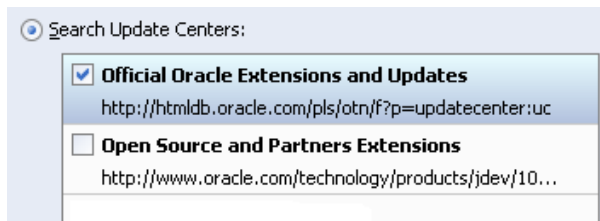
Installing the JDeveloper Subversion VCS extension

The JDeveloper Subversion VCS extension is based on JDeveloper production version 10.1.3.0.4. You will need either the J2EE or Studio Edition; the Java edition will be supported in a future version. If you have not already done so, download and install JDeveloper 10.1.3 (J2EE or Studio) from <http://otn.oracle.com/products/jdev>.

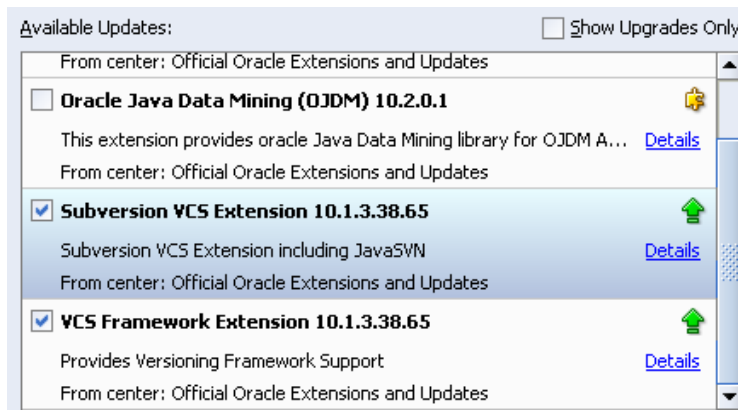
The JDeveloper Subversion VCS extension is available via the official Oracle update center. You install updates using the Check for Updates feature available from the Help menu:



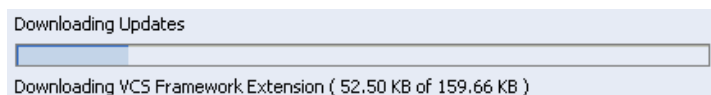
Using the Check for Updates wizard, search the Official Oracle Extensions and Updates center:



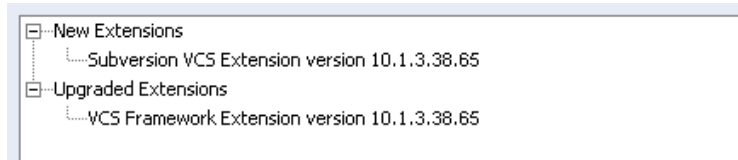
Locate and select the Subversion VCS Extension entry. Selecting this will automatically select the VCS Framework Extension update too.



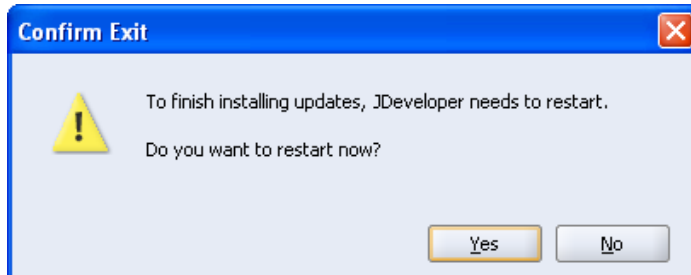
Proceed to the download stage:



After downloading the updates, you will see a summary page with information similar to the following:



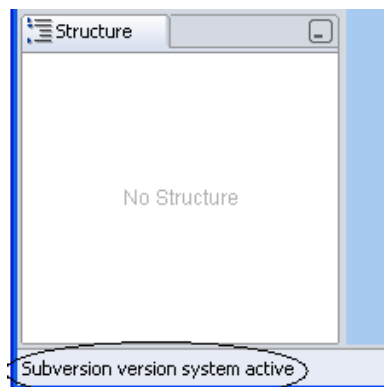
On completion of this wizard, you may be asked whether you want to automatically restart JDeveloper:



Choose "Yes".

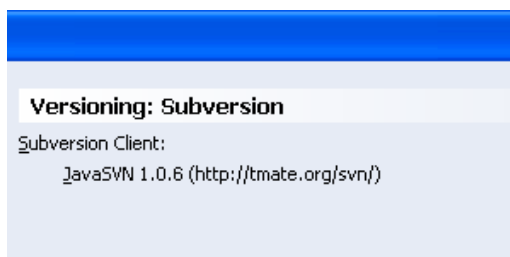
When JDeveloper has restarted, open the **Versioning** menu, choose **Select Version System** and then **Subversion**.

If configuration was successful, you will see a status message below the structure pane:



Assuming success, view the preferences for Subversion to check that the JavaSVN client library has been installed as expected: Open the Preferences dialog (**Tools > Preferences**) and in the left pane select **Versioning | Subversion**.

With the extension correctly installed, you will see something like:



Connecting to a Subversion repository

The Subversion repository holds all of your versioned data.

If you have previously been using Subversion as your versioning system, you will already have a repository that you can connect to through JDeveloper. In the Subversion Navigator (**View > Subversion Navigator**), right-click the Subversion node and select **New Repository Connection**. In the Create Subversion Connection dialog, enter the repository location, using an appropriate access method protocol (<http://>, <https://>, <svn://>, <svn+ssh://>). You can also use the <file:///> protocol, but to do so you will need to install additional software, as explained in [Appendix B](#).

TIP: If you want to follow the examples given in this Developer's Guide, you will need to install a local Subversion repository named `c:/svn-repos` and connect to it using the `svn://` protocol. The JDeveloper Subversion VCS extension will allow you to install a local repository if you also have separate Subversion client software installed. See [Creating a local Subversion repository](#).

If you want to save the details of your Subversion repository connections so that you can use them later or on another machine, you can export them to a text file. You can then recreate the connections by importing the details from the file. To export, right-click the Subversion node in the Subversion Navigator and choose **Export Connections**. Complete the Export Subversion Connections dialog, then click **OK**. To reimport connection details from a file, right-click the Subversion node and choose **Import Connections**. Through the Import Subversion Connections dialog, browse to the file that contains the connection details that you wish to import, then click **OK**.

Importing JDeveloper projects into Subversion

To provide examples to use with this guide, create a new Java application and project, with one Java package and two simple Java classes. Specify the location of the project as `c:\jdev1013\jdev\mywork\test`.




Files (such as these) that you have created in JDeveloper before using Subversion control must be imported into the Subversion repository, and then checked out from it. These checked out files become your "working copies".

You import existing JDeveloper projects using JDeveloper's Import to Subversion wizard.

Before you begin, you need to know the location of the Subversion repository where you would like to store your JDeveloper projects. For the example used in this guide, a URL is used, as follows:

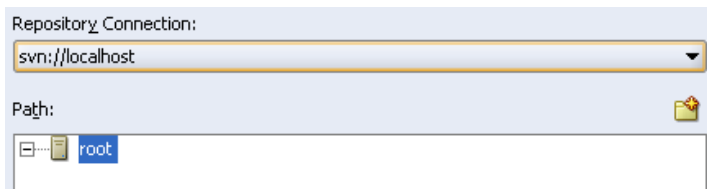
```
svn://localhost/
```

Another piece of information that could be useful is the location of the directory that holds your JDeveloper application, although this will be set from the context in which you invoke the wizard. For the example used in this guide, this is:

```
c:\jdev1013\jdev\mywork\test
```

Launch the wizard by right-clicking the JDeveloper application or project that you want to import (in this example, TestApplication), then selecting **Versioning > Import Files**. This opens the Import to Subversion wizard.

On the Destination page, make sure that the Repository Connection box contains the URL (or connection name, if you gave it one) for the repository that you created earlier:



In general, you should select the location in the repository you want to import to. For the purposes of this guide, you created a new, empty, repository so for now you can leave the Path selection as 'root'.

On the Source page, make sure that the Source Directory box contains the location of the directory that holds your JDeveloper application which, for the example in this guide, should be:



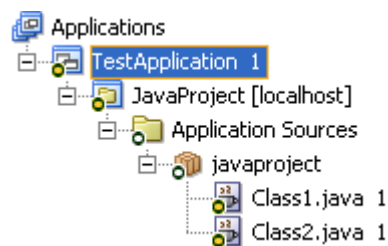
You do not have to make any changes on the Filters page.

On the Options page make sure that the Perform Checkout box is selected:



Complete the import by clicking **Finish**.

Unfold the TestApplication node and the nodes beneath it to see this view.



Because you allowed the wizard to check out the imported files, the files are shown in the navigator, and they have a version number next to them.

In Subversion, the version number refers to the entire repository, not to individual files within it. A version number next to a file therefore represents the latest revision in which that file was modified. Upon import into a new, empty repository, all files will have their version number set to "1". When importing into a prepopulated repository, a version number will be assigned depending on the existing contents of the repository, but will be the same for all the newly imported files.

In JDeveloper, you can see the version information for a file by right-clicking it in the Application Navigator and choosing **Versioning > Properties**.

If you have set the preference to show navigator state overlay icons (see **Tools > Preferences | Versioning | Subversion | General**) you will also see small symbols indicating the version status of the files. For example, an orange circle indicates that a file is unmodified since it was last updated from the repository.

The files now visible in the Application Navigator are your Subversion "working copy" files and are ready to be worked on.

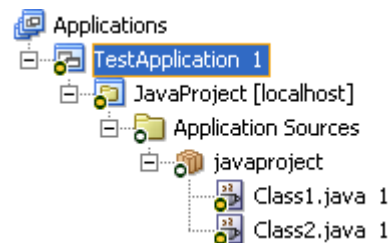
Checking out files

"Checking out" is a task you perform when you wish to create the initial local copies of files and folders that are being stored within a Subversion repository. It is these local copies, stored outside the Subversion repository, that you work on.

If you have used the Import to Subversion wizard with the Perform Checkout wizard selected, the imported files will already have been checked out.

As an example of checking out files independently of the import operation, in the Subversion Navigator, right-click the connection node (in the example, `svn://localhost`) and select **Check Out**. Confirm that you want to check out from the root. A location for the checked out files is suggested. To avoid confusion with the files checked out as part of the import operation, change the destination to `c:/work` and click OK.

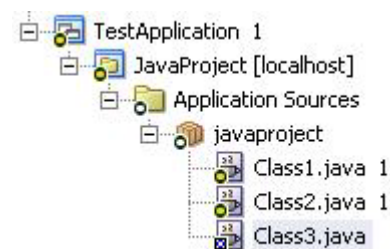
The files that you checked out of the Subversion repository will now appear in the Application Navigator, with a version number next to them.



Adding and committing files

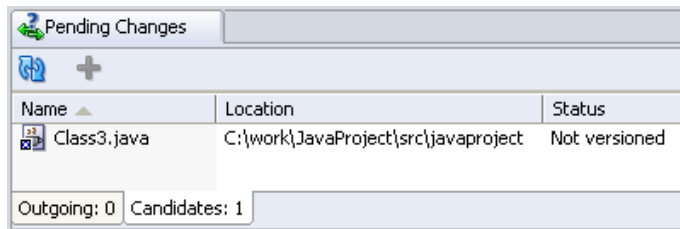
When you create new files within JDeveloper, you must add them to Subversion control if you want to use versioning features with them and if you want to make them available to other developers. You do this by using JDeveloper's Add command, followed by the Commit command.

In the Application Navigator, create a new Java file, `Class3.java`, as part of your versioned application.




You will see that the navigator icon for the new file is a diagonal white cross on a blue background. This indicates that the file is currently unknown to the version control system and that you should add it.

Before doing so, take a look at a JDeveloper tool that will help you keep track of any changes that should be made to the repository. If you do not already have the Pending Changes window open, open it by selecting **Versioning > Pending Changes**. The Candidates tab of this window will be orange, indicating that it contains a change that you have not yet seen. Click on the tab and note that the file you created, `Class3.java`, is listed with the status of "Not versioned".

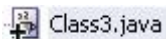


You can add Class3.java to Subversion control from here. In a future release, the Pending Changes window will also tell you whether changes have been made to the file in the repository, giving you the chance to update your local copy if you wish to.

To add Class3.java to Subversion control, do one of the following:

- Select it in the Application Navigator and choose **Versioning > Add**.
- Right-click it in the Pending Changes window and choose **Add**.
- Select it in the Pending Changes window and click the **Add** icon .

Any of these methods will open a dialog box through which you confirm that the file should be added. Once added, the navigator icon changes to a small black cross:



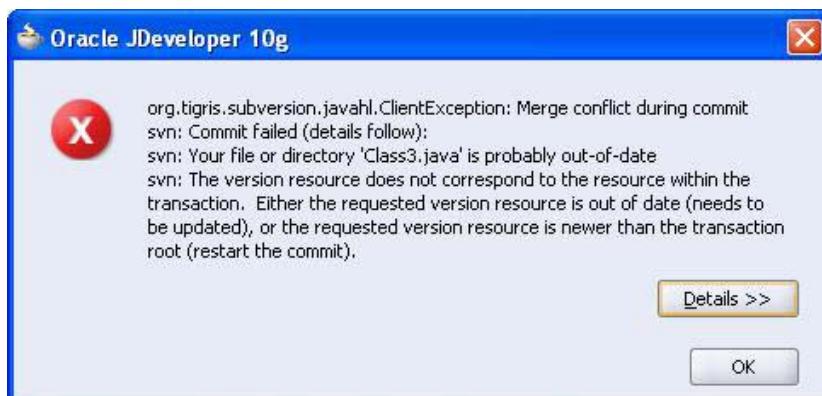
This indicates that the file is known to Subversion but is not yet part of the repository. To make the file part of the repository you use JDeveloper's Commit command: select Class3.java in the Application Navigator and choose **Versioning > Commit**. This opens the Commit Resources dialog. You should definitely use the Comments box in this dialog to describe actions that you carry out on the file. In this case, you could add the comment "Initial commit of Class3.java". These comments will be visible when you view the versioning history of a file, and will help you and other developers on the project to identify one version from another.

TIP: In a live development environment, to avoid unnecessary conflicts and the subsequent work in resolving them, you should update your files before committing them.

Updating files



"Updating" is the process of bringing your local files up to date with those in the Subversion repository. You would do this because another developer has committed his changes to the repository. You can update your files using JDeveloper's Update command.

You will know that there is a more up-to-date version of a file in the repository when you attempt to commit your local version of it. A message will be shown and you will not be able to proceed with the commit.



You can update your entire working copy, single directories or packages, or individual files.


To update files, select them or their parent directory in the Application Navigator, then choose **Versioning > Update**.

If a file can be updated without causing a conflict, the navigator icon will change to an orange circle  and the file content will be updated with content from the repository version. If, when a file is updated, there is a conflict, the navigator icon will change to an exclamation point  and the file content will be changed to show both the original content and the content from the repository. The conflicting content will be separated by conflict markers (see [Resolving Conflicts](#)).

Editing files

You can edit files from your working copy directly in JDeveloper, or by using an external editing tool. Both binary and text files can be processed by the versioning system, so you can use text editors, word processors, graphics programs, or any other tool that you would normally use.

You do not need to register that you wish to edit a file (in other words, you do not have to "obtain an edit" as you would in other versioning systems). When you make changes, Subversion automatically detects that the file has been changed, and the versioning status of the file will be updated in JDeveloper to reflect this.

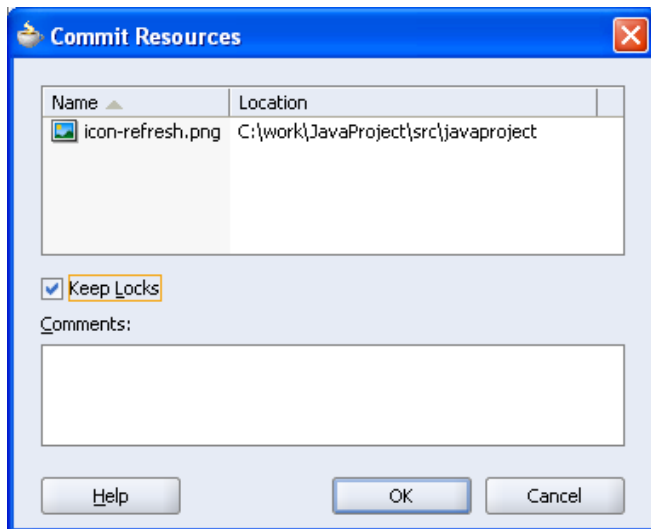
TIP: To be certain that you are seeing the most up-to-date status of files in JDeveloper, click the Refresh button  on the Application Navigator toolbar.

Where more than one developer is going to be working on the same file, there has to be a system to prevent one developer's work unintentionally overwriting another's. The preferred solution to this with Subversion is the **copy-modify-merge model**. In this model, each developer working on a file takes a copy of it, modifies it, then merges it with the other developers' copies. This may sound as if it would cause frequent conflicts in the content, but in practice this is rarely the case. When conflicts do occur, there is an efficient process for resolving them (see [Resolving Conflicts](#)).

An alternative to the above is the **lock-modify-unlock model**, in which a developer locks a file, modifies it, adds it back to the repository, then unlocks it. Locking a file does not prevent another developer from working on a copy of the file: it prevents the content of that copy from being returned to the repository until the lock is removed. So there is still potential for wasted effort in this model.

The lock-modify-unlock model is more suitable for working with files in binary formats, where it is impossible to merge conflicting changes. It ensures that developers take it in strict turns to submit changes to a file.

In this release of JDeveloper, support for locking is limited to retaining locks on files that you are about to commit. You must previously have locked the files using the Subversion lock command. In a future release, locking and unlocking will be fully supported.



Retaining a lock will mean that other developers will still not be able to commit changes they made to their copies of the files.

To lock a file using the Subversion lock command, enter the following at a command prompt:

```
svn lock [file name] -m "[message]"
```

To unlock a file, use:

```
svn unlock [file name]
```

Comparing and merging revisions of files

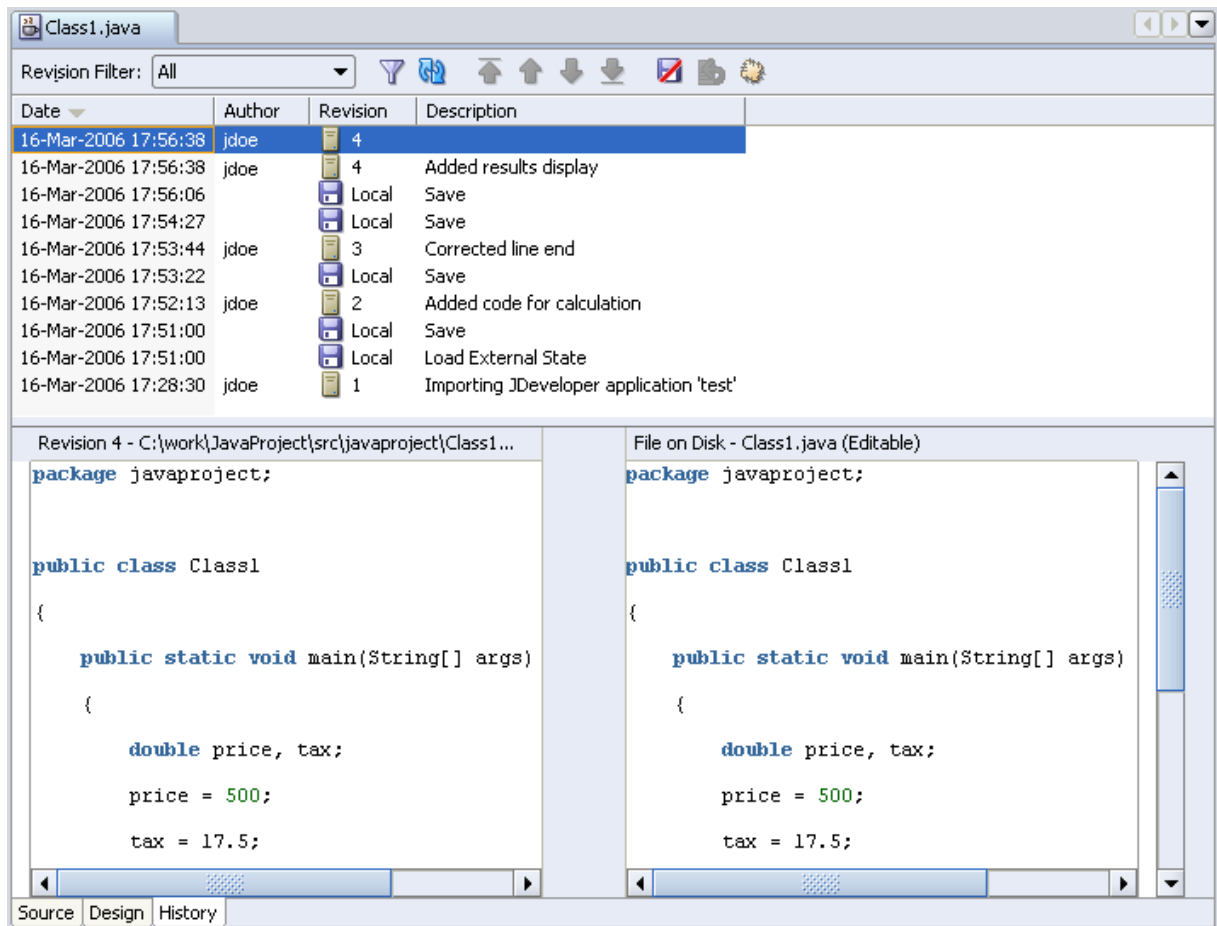
JDeveloper provides two tools for comparing files, the local history tool and the version history tool.

Using the local history tool

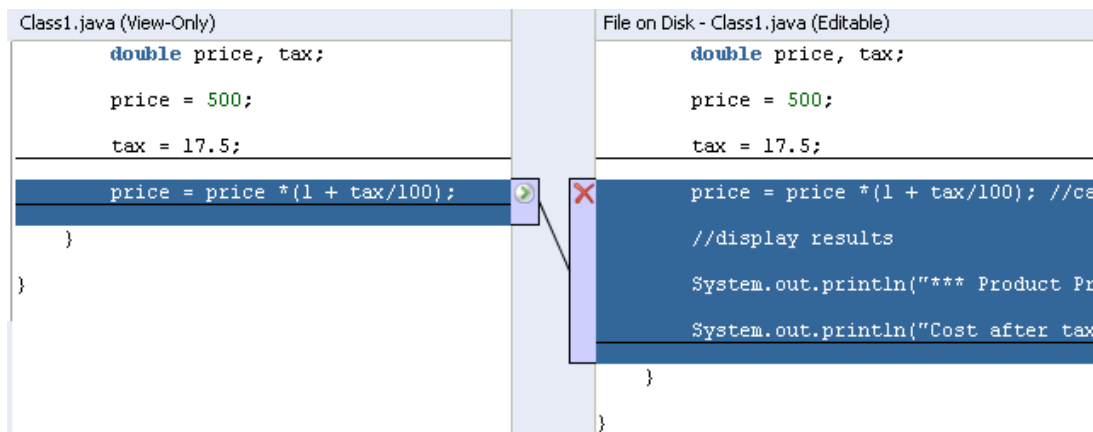
Using the local history tool you can:

- View the content of earlier versions of a file.
- Merge the content of earlier versions into the latest one.
- Resolve conflicts between earlier versions and the latest one.

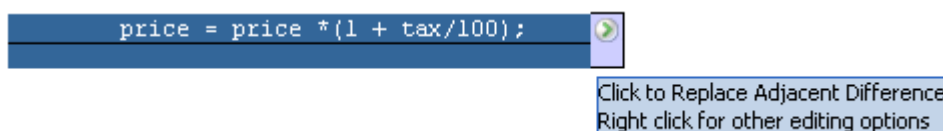
To see the history of a particular file, open the file by double-clicking it, then click the History tab.



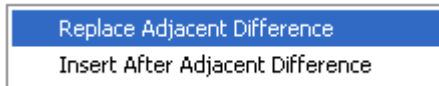
The left panel contains the content of the currently selected revision. You cannot edit this content. The right panel contains the content of the most recent revision (the same as you can see on the Source tab). You can edit the content in the right panel. If you do, your changes are instantly reflected on the Source tab. For the most recent revision, there will be no difference between the left and right panels. For earlier revisions, differences are shown by linked boxes that outline any lines that conflict.



Symbols in the margin between the two panels indicate the suggested action for resolving the conflict. Hover the mouse pointer over the symbol to display a tooltip that explains the suggested action.



If there are any alternate actions, you can implement them from the symbol's context menu (right-click the symbol).



Using the version history tool

Use the version history tool to compare any version of a file with any other version of that file. This is useful for seeing what changes other developers have made to a file. The file versions shown are read-only: you can compare the content of two versions of a file but you cannot edit either of them. The version history tool cannot therefore be used to merge versions of files.


To view and compare versions of a file, select the file in the Applications Navigator and choose **Versioning > Version History**.

The version history tool is similar to the local history tool described earlier, with the following differences:

It has two version selection panels instead of one, so that you can select any combination of versions to compare. (Note, however, that only the left panel has difference navigation controls.)

Although the differences between the versions are shown by highlighting and linked boxes, there are no action icons or context menus associated with them.

Resolving conflicts

Conflicts arise when two files have identical line positions containing different content. You will probably learn that your local copy of a file conflicts with the one in the repository when you try to commit it. In this case, the commit is prohibited and, after you have updated your local file from the repository, an exclamation point  is shown in the file's navigator icon.

If conflicts have arisen in one of your files, you will have to resolve them before you can commit the file to the repository. A merge tool is provided in JDeveloper through which you can resolve the conflicts in text-based files. There is no easy way to resolve conflicts in binary files, so JDeveloper lets you indicate to Subversion that a conflict is resolved even when no changes have been made.

Conflicts in text-based files

In a text-based file with conflicts, the file content will be changed to show both the original content and the content from the file in the repository. The conflicting content will be separated by conflict markers. The simple example below shows a conflict arising from the addition of differing comments on the same line of a Java file.

```

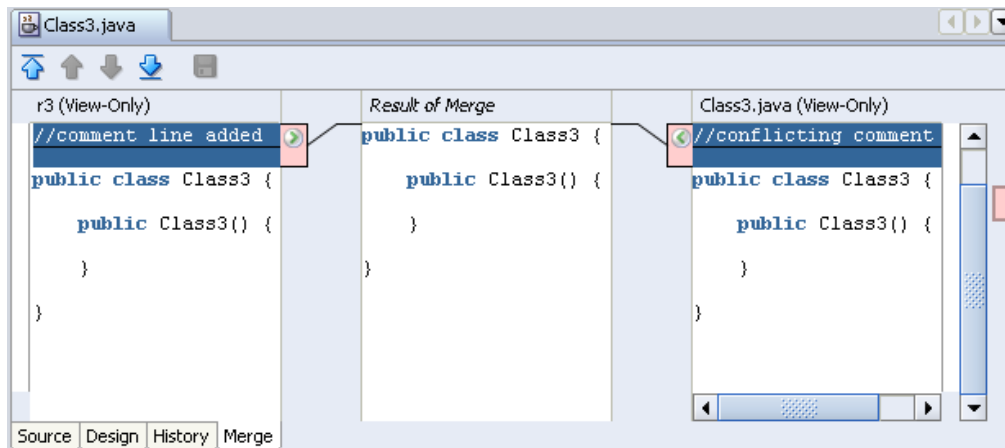
<<<<<<< .mine
//conflicting comment line

=====
//comment line added
>>>>>>> .r321

```

The start of the conflicting content is shown by open angle brackets, and the end is shown by closing angle brackets, followed by a revision number. Your content is shown before a line of "equals" signs. The content from the repository is shown after the line of "equals" signs.



To open the merge tool so that you can resolve the conflict, select the file in the Application Navigator and choose **Versioning > Resolve Conflicts**.



The left panel contains the content of the version in the repository. The panel heading shows the revision number. You cannot edit this content. The right panel contains the content of the most recent local version. This content also cannot be edited. The center panel contains the results of the merge. The content will be updated as you resolve the individual conflicts. You can also edit the content directly. Symbols in the margins between the three panels indicate the suggested action for resolving each conflict. Hover the mouse pointer over the symbol to display a tooltip that explains the suggested action. If there are any alternate actions, you can implement them from the symbol's context menu. Accepting an initial suggested action may cause the appearance of additional suggested actions.

To complete the merge, you must save the changes that have been made, using the  Save button.

Conflicts in binary files

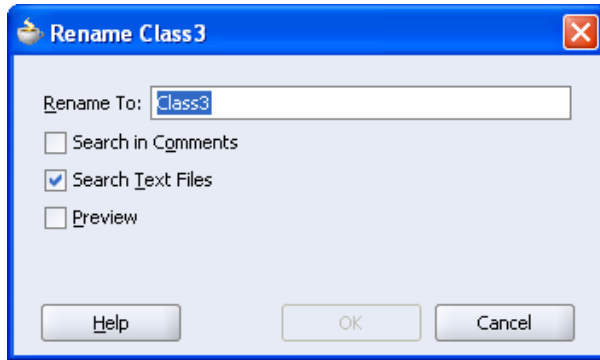
Because it is not easy to resolve conflicts in binary files (such as graphics files), you can indicate to Subversion that the conflicted file is the one that should be regarded as correct, even though you have made no changes to it. To do this, select the file in the Application Navigator, then select **Versioning > Mark Resolved**. The file's navigator icon changes from an exclamation point  to an asterisk  to indicate a modified file. When you commit this file (**Versioning > Commit**) the Subversion repository should accept it without raising a conflict message.

Renaming files

Subversion allows the renaming of versioned source files, and JDeveloper supports this.

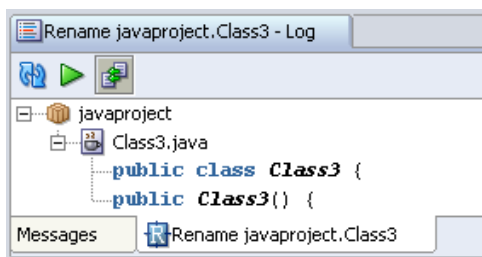
When you rename a source file in JDeveloper, all occurrences of the name in that file and in other source files will be renamed.


To rename the Java source file Class3.java to Class4.java, select Class3.java in the Application Navigator and choose **Refactor > Rename**. This opens the Rename dialog.



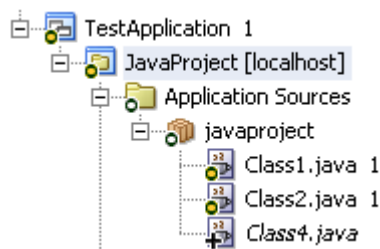
Overwrite the existing name (Class3) with the new name (Class4). You do not need to specify the file extension. If you want instances of the name changed in text files that are part of the project, check the Search Text Files box.

TIP: If you would like to view a list of the usages of the name before any changes are made, check the Preview box. The preview will be displayed in the Log window when you click **OK**. The log displays a collapsible tree of packages and Java files. Under each file one or more usages are listed. Double-click on a usage to view it in an Edit window. Use the preview to inspect and modify or exclude individual usages, before completing the rename operation.

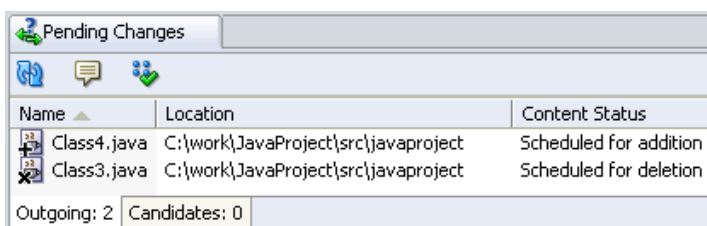


If you selected the Preview option, complete the rename by clicking the Do Refactoring button  in the Log window.

If you did not select the Preview option, the file name changes were made when you clicked **OK**. The Application Navigator now shows that the newly named file needs to be saved (its name is italicized) and committed (the navigator icon includes a small black cross).



In the Pending Changes window, a file with the old file name is shown as scheduled for deletion, and a file with the new file name is shown as scheduled for addition.



When you attempt to commit the files, you will be told that Class4.java has been modified. You will be prompted to save the file, which you should do.

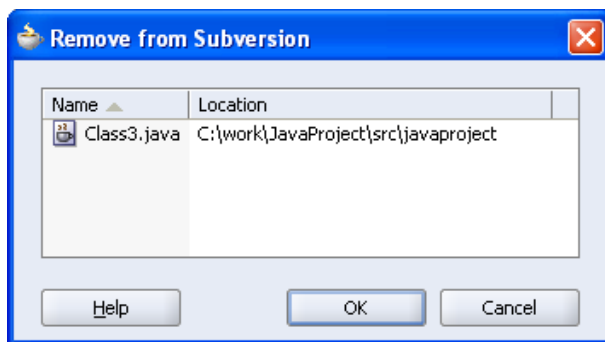
If you look at the local history of Class4.java (open the file in JDeveloper and click the History tab), you will see that the file's history details continue from that of Class3.java.

TIP: If, before you commit, you change your mind about renaming a file, you can use the Revert command. First save the renamed file - you will not be able to proceed without doing so. Then revert the renamed file (**Versioning > Revert**). This will create a new file with the name used for the renamed file (in the above example, Class4.java). You can delete this new file if you wish. Now go to the Pending Changes window, find the file with the original name, and revert it. The original file will be restored to the Application Navigator and shown as up to date. Note, however, that references to the file in other files will also have been renamed and that these will not have been changed back to the original name. You will have to search for these and change them manually.

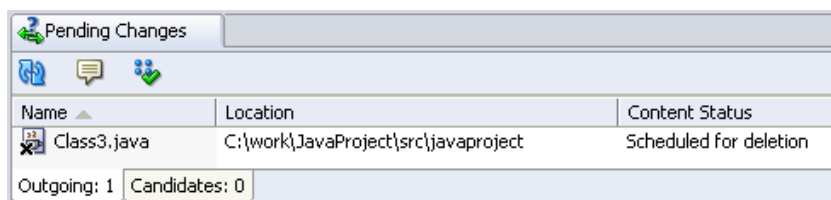
Removing files from Subversion control

CAUTION: This process erases files from disk when removing them from Subversion control.

To remove a file from Subversion control, select the file in the navigator and choose **Versioning > Remove**.



When you click the **OK** button, the Outgoing tab of the Pending Changes window shows the file as scheduled for deletion:



TIP: If you change your mind about removing the file at this stage, select the file in the Pending Changes window and choose **Versioning > Revert**.

To complete the removal of the file from Subversion control, select it in the Pending Changes window and choose **Versioning > Commit**. Add your comments to the Commit Resources dialog, and click **OK**. The file is removed from view in the Subversion repository, from your local working copy, and from all views in JDeveloper.

Appendix A: Installing Subversion client software

You do not have to install any Subversion software additional to the JDeveloper Subversion VCS extension, except in the following circumstances:

You wish to create a local Subversion repository using the JDeveloper Subversion VCS extension.

You wish to use a Java binding (helper library) other than JavaSVN, which is the one supplied with the extension.

You wish to connect to a Subversion repository through a proxy server (see [Connecting to a Subversion repository through a proxy server](#)).

In all of the above cases, you will need to install separate Subversion client software. If you wish to use an alternate Java binding, you will additionally have to install the binding software (see [Installing the JavaHL library](#)).

If you need to install Subversion client software, you can obtain the necessary Subversion source code, and binaries for certain platforms, from <http://subversion.tigris.org>. You should install Subversion 1.3.2. If you are using an operating system with a package management system, such as some distributions of GNU/Linux or other popular Unix systems, you can either consult your OS vendor or compile the client software from the source code.

For Microsoft Windows, download the Subversion installer, svn-1.3.2-setup.exe, from <http://subversion.tigris.org>. Run the installer and place the Subversion client in a convenient location, for example c:\subversion. To check the installation, open a command prompt and type `svn help`. If you don't see a list of subcommands, check that your system path includes the location where you installed Subversion (for example, `echo %PATH%` should contain c:\subversion\bin).

Appendix B: Installing an additional Java helper library

JDeveloper requires a helper library to allow it to communicate with Subversion repositories and working copies. The helper library JavaSVN (see <http://tmate.org>) is installed automatically with the JDeveloper Subversion VCS extension. You can additionally install Subversion's own library, referred to as JavaHL. The JavaHL binding has the advantage of being developed and maintained by the makers of Subversion and thus allows repository access via a wide range of protocols (<http://>, <https://>, <file:///>, <svn://>, <svn+ssh://>). If you install JavaHL, JDeveloper will let you choose between using JavaHL and using JavaSVN.

Installing the JavaHL library

JDeveloper works with version 1.3.2 of the JavaHL library.

For Microsoft Windows:

1. First, ensure the bin directory of your client installation (see [Installing Subversion client software](#)), is on your system path.
 2. Download the JavaHL binary (svn-win32-1.3.2_javahl.zip) from <http://subversion.tigris.org>.
 3. Using an appropriate tool, e.g. WinZip, extract it to a temporary location, e.g. c:\temp
 4. Locate the extracted directory in Windows Explorer, e.g. c:\temp\svn-win32-1.3.2
 5. Copy the libsvnjavahl-1.dll file to the bin of your Subversion client installation, e.g. c:\subversion\bin
- Your client installation's bin directory should contain the following files:

Name	Size	Type
intl3_svn.dll	69 KB	Application Extension
libapr.dll	125 KB	Application Extension
libapriconv.dll	37 KB	Application Extension
libaprutil.dll	165 KB	Application Extension
libdb43.dll	692 KB	Application Extension
libeay32.dll	1,036...	Application Extension
libsvnjavahl-1.dll	989 KB	Application Extension
mod_authz_svn.so	109 KB	SO File
mod_dav_svn.so	481 KB	SO File
ssleay32.dll	196 KB	Application Extension
svn.exe	913 KB	Application
svnadmin.exe	449 KB	Application
svndumpfilter.exe	397 KB	Application
svnlook.exe	421 KB	Application
svnserve.exe	469 KB	Application
svnversion.exe	757 KB	Application

TIP: Check to see if other versions of the above files are on your computer. If so, make sure that any entry in the system path that points to them is later in the path than the entry for the Subversion files. If they are earlier in the path, Subversion will attempt to use them and, if they are incompatible, an error will occur. For example, if an incompatible version of the file `ssleay32.dll` is used, you will see the message "SSL negotiation failed: SSL disabled due to library version mismatch".

For GNU/Linux and other UNIX based operating systems:

If your OS distribution uses a package management system, consult the documentation for that system to ensure you install the JavaHL library objects.

If you have compiled your Subversion client from the source code, you may need to take extra steps to ensure the JavaHL library objects are built.

Typically, this will involve issuing the command: `make javahl && make install-javahl`

Configuring JDeveloper for the JavaHL library

If you have installed the JavaHL library, double-check that the OS-native objects are available on your system path:

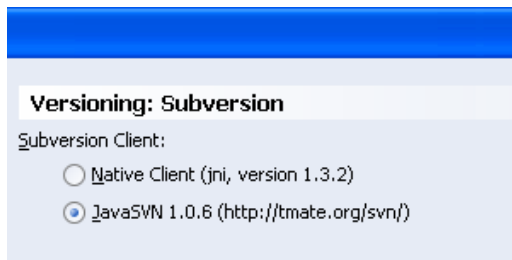
For Microsoft Windows, if you followed the steps outlined previously in this document the DLLs should already be on your system path. Satisfy yourself that this is the case by starting a command prompt, echoing the path variable (`echo %PATH`) and verifying that it contains the path to the DLLs (crucially, `libsvnjavahl-1.dll`).

For GNU/Linux based systems you can run `'ldconfig -v | grep svn'` as root. If you don't see any appropriate entries, consult your vendor's documentation on how to rectify the situation.

Start or restart JDeveloper.

Open the Preferences dialog (**Tools > Preferences**) and in the left pane select **Versioning | Subversion**.

If the JavaHL library has been installed correctly, you will see something like:



To use the JavaHL library, select the **Native Client** option.

Note that if you subsequently update the JDeveloper Subversion VCS extension from the Oracle Update Center, the above preference will be reset to JavaSVN.

Appendix C: Connecting to a Subversion repository through a proxy server

If you wish to connect to a Subversion repository through a proxy server, you must first install separate Subversion client software. See [Installing Subversion client software](#).

Once you have installed the Subversion client software, you will have a Subversion subdirectory in your Windows Application Data directory. To find the Application Data directory, at the `c:/` prompt type `cd %APPDATA%`. Then open the Subversion subdirectory. (On Linux the equivalent subdirectory will be in `~/.subversion`, where `~` is the home directory.)

In the Subversion subdirectory will be a file named `servers`. Open this file with a text editor and find the `[global]` section. Remove the comment marker (`#`) from the line `http-proxy-host` and overwrite the placeholder proxy information with the details of the proxy server that you use. Remove the comment marker (`#`) from the line `http-proxy-port` and overwrite the placeholder port information with the port number for the proxy server. If you wish to exclude certain URLs from using the proxy server, remove the comment marker (`#`) from the line `http-proxy-exceptions` and overwrite the placeholder URLs with URLs that you wish to exclude.

Add additional `http-proxy-host` and `http-proxy-port` lines with details of any other proxy servers that you use.

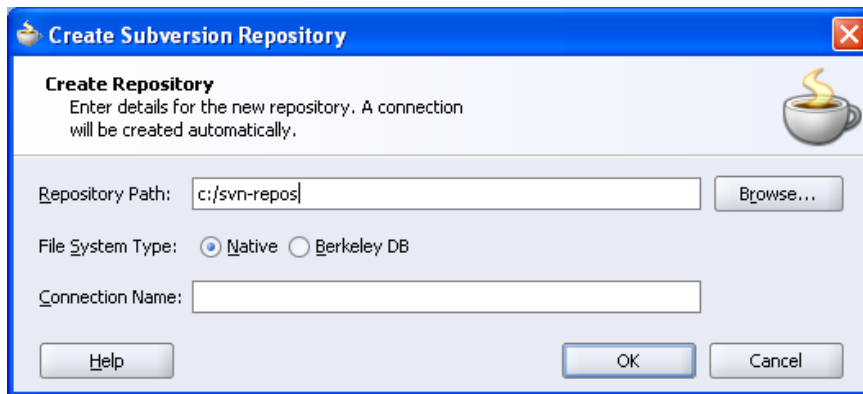
It is important that the proxy server supports all the http methods used by Subversion. Some proxy servers do not support the following methods by default: PROPFIND, REPORT, MERGE, MKACTIVITY, CHECKOUT. If you experience problems with using a proxy server to access a Subversion repository, ask the server's system administrator to change the configuration to support these http methods.

Appendix D: Creating a local Subversion repository

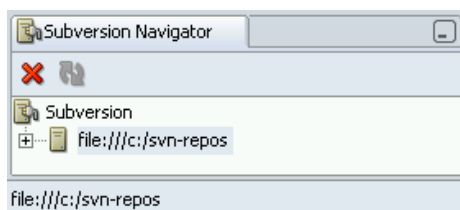
The JDeveloper Subversion VCS extension includes a feature for creating local Subversion repositories. Because of restrictions in the supporting software currently supplied with the extension, you must install separate Subversion client software before you can use JDeveloper to create a local repository. Instructions for installing Subversion client software are given in [Installing Subversion client software](#).

To create a local Subversion repository, choose **Versioning > Create Repository**.

In the Create Subversion Repository dialog, a location and file system type are suggested for your new repository. To provide an example to use in this guide, overwrite the repository path with `c:/svn-repos`. You can leave the file system type as the default Native, and you do not have to provide a connection name.



When the repository is created, a connection to it will be created automatically. You can see this in the Subversion Navigator.



If you are using JavaSVN you will need to change the access method protocol from `file:///` to `http://`, `https://`, `svn://` or `svn+ssh://`. These protocols are described in the Subversion documentation.

TIP: The `svn://` protocol is particularly useful for a small team or individual developer where security is not a big concern, and is easy to set up. At a command prompt enter:

```
svnserve -d -r [repository location]
```

Leave the command window open. When you create a connection to your repository, use:

```
svn://[computer name]/
```

as the repository location.

For the examples used in this guide, at a command prompt enter `svnserve -d -r c:/svn-repos/` and leave the command window open. In the Subversion Navigator, double-click the connection name `file:///c:/svn-repos`. In the Edit Subversion Connection dialog, overwrite the Repository URL with `svn://localhost/`. Test the connection, and click **OK** to close the dialog.