Table of Contents

Table of Contents

Basic tutorial

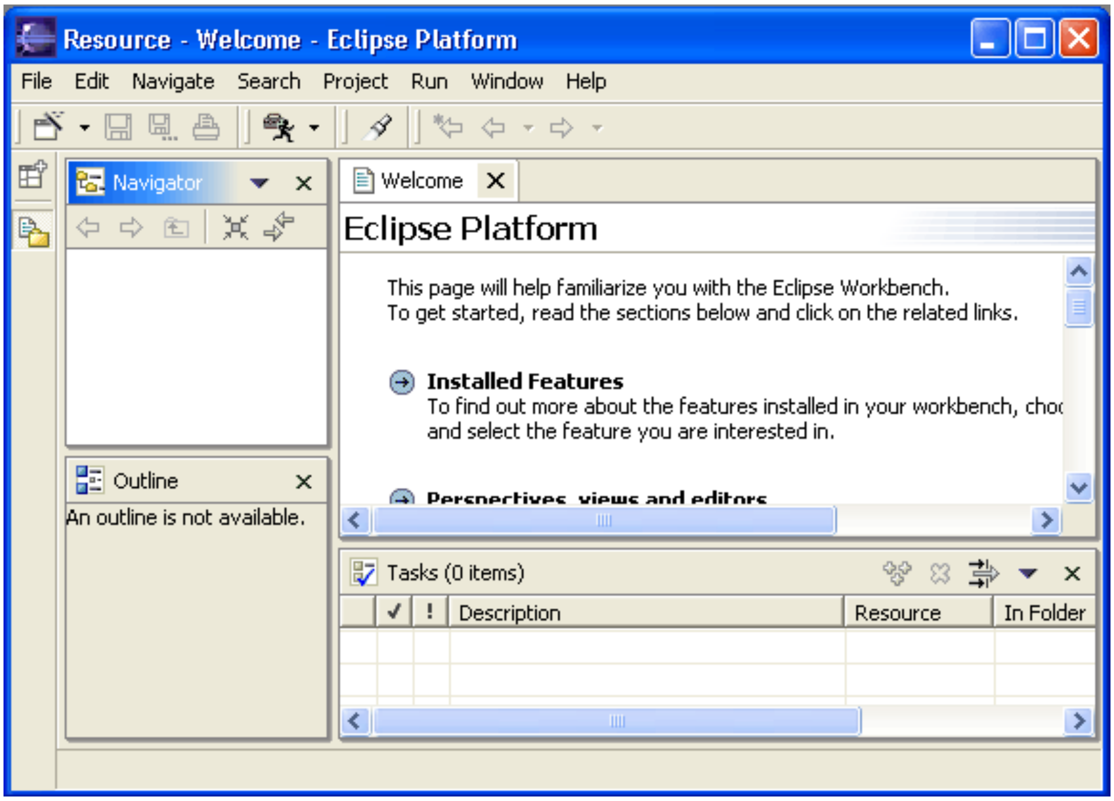
This tutorial provides a step by step walk−through of the Workbench.

The Workbench

Once you have started the Workbench you will see a single window. The Workbench window displays one or more perspectives. Initially the Resource perspective is displayed. A perspective contains views (like the Navigator) and editors. More than one Workbench window can be open at a time.

At the far left of the window is a shortcut bar that allows you to open new perspectives and move between ones already open. The name of the active perspective is shown in the title of the window and its icon in the shortcut bar is a pushed−in icon.

Looking at the title bar of the Workbench window we can see that we are working with the Resource perspective. The Navigator, Tasks, and Outline views are open, along with an editor on the welcome page. You might want to take a moment to peruse the welcome page. It provides a quick overview if you are eager to customize your Workbench or learn about other features.


Resource - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Welcome
Outline
Tasks (0 items)

Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

Installed Features
To find out more about the features installed in your workbench, choo
and select the feature you are interested in.

Perspectives, views and editors

An outline is not available.

	✓	!	Description	Resource	In Folder

Editors and views

Before we dive into using the Workbench, let's start by familiarizing ourselves with its various elements. A Workbench consists of:

- perspectives
- views
- editors

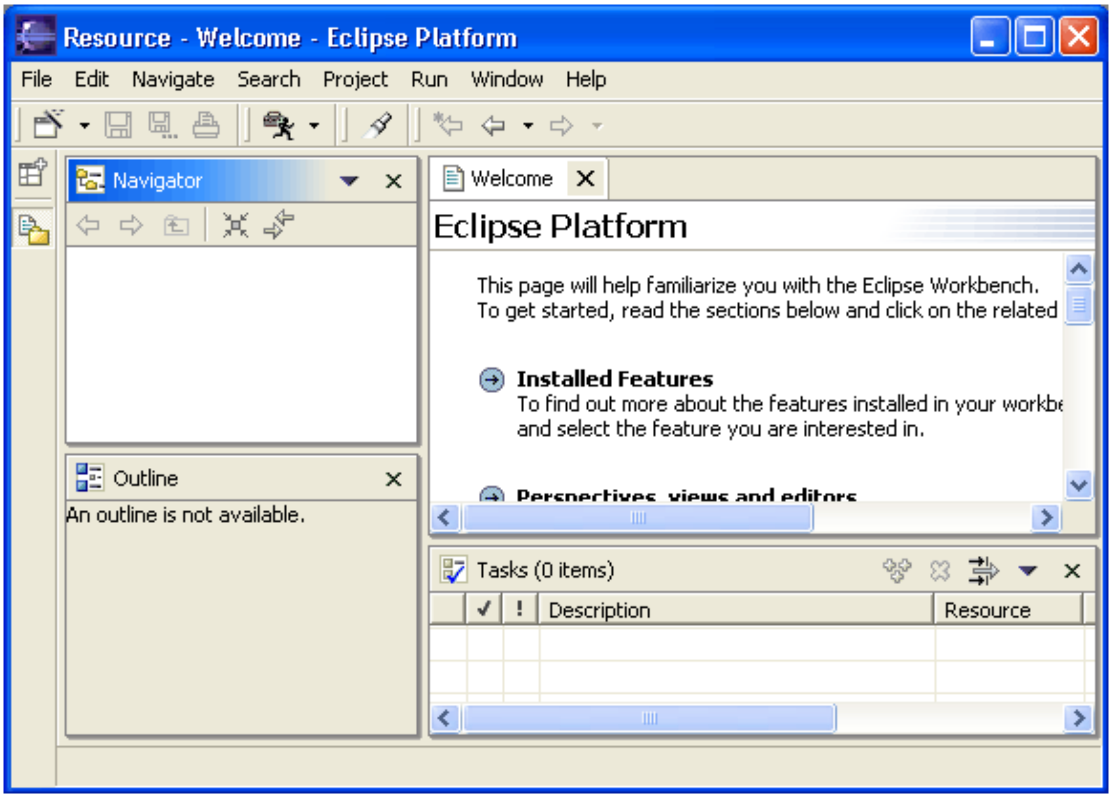
A perspective is a group of views and editors in the Workbench window. One or more perspectives can exist in a single Workbench window. Each perspective contains one or more views and editors. Within a window, each perspective may have a different set of views but all perspectives share the same set of editors.

A view is a visual component within the Workbench. It is typically used to navigate a hierarchy of information (like the resources in your Workbench), open an editor, or display properties for the active editor. Modifications made in a view are saved immediately. Only one instance of a particular view type may exist within a Workbench window.

An editor is also a visual component within the Workbench. It is typically used to edit or browse a resource. Modifications made in an editor follow an open−save−close lifecycle model. Multiple instances of an editor type may exist within a Workbench window.

Editors and views can be active or inactive, but only one view or editor can be active at any one time. The active editor or view is the one whose title bar is highlighted. The active part is the target for common operations like cut, copy and paste. The active part also determines the contents of the status line. If an editor tab is white it indicates the editor is not active, however views may show information based on the last active editor.

If you click on the Navigator view you will notice it becomes active.


Resource - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
Welcome
Tasks (0 items)
Description
Resource

Eclipse Platform

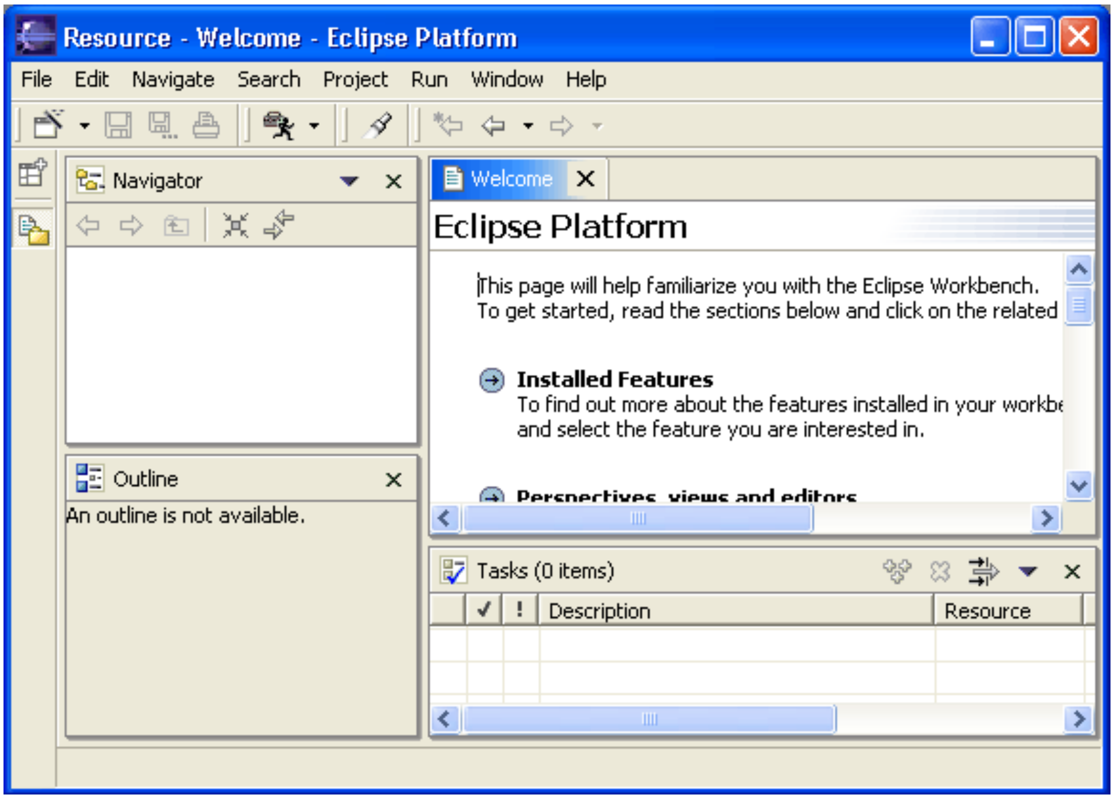
This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related

Installed Features
To find out more about the features installed in your workbe
and select the feature you are interested in.

Perspectives, views and editors

An outline is not available.

Clicking on the welcome page (editor) you can see that the editor's tab turns blue, and the Navigator's title bar is no longer blue. The welcome page is now our active editor.


Resource - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
Welcome
Tasks (0 items)
Description
Resource

Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related

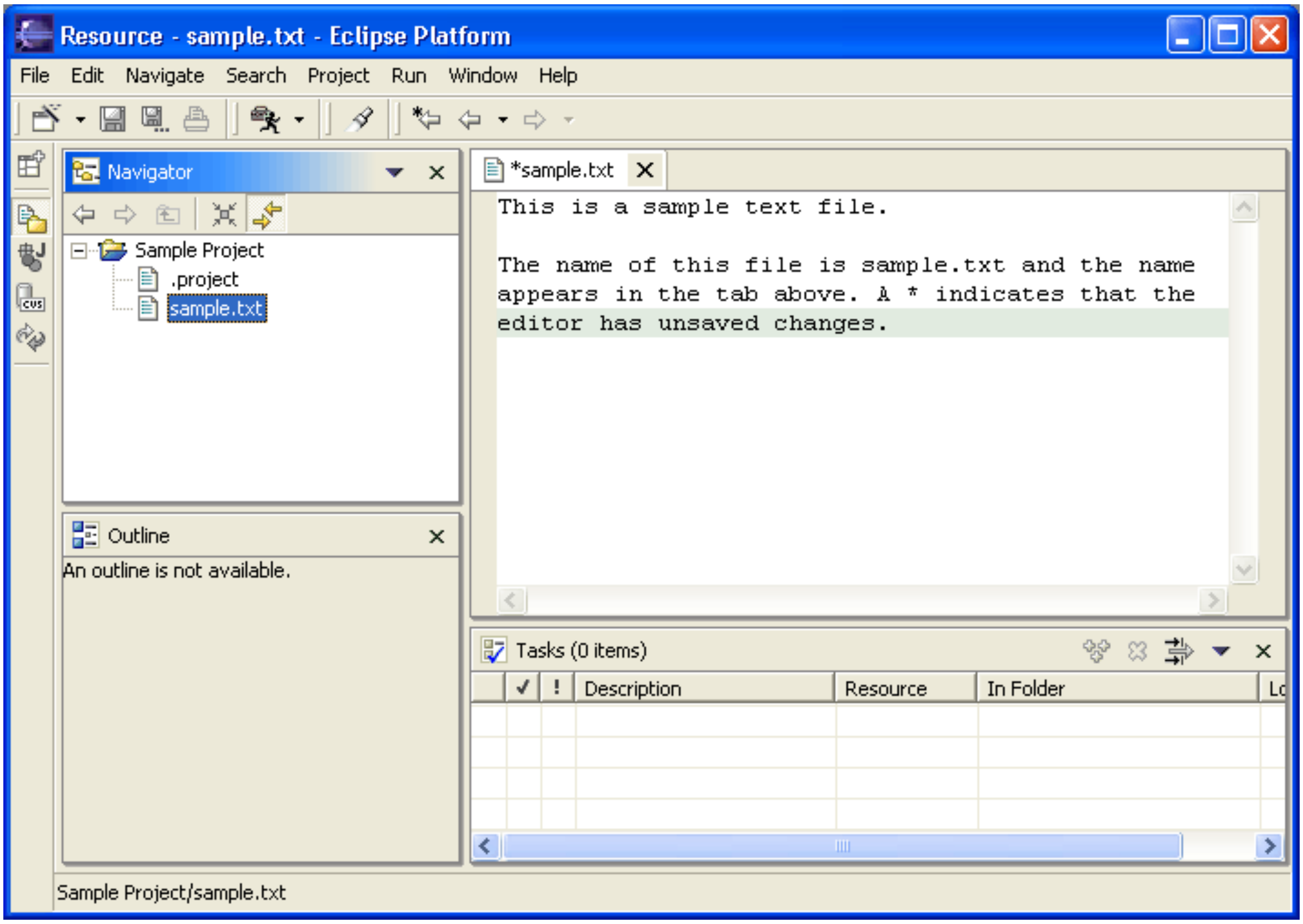
Installed Features
To find out more about the features installed in your workbe
and select the feature you are interested in.

Perspectives, views and editors

An outline is not available.

Editors

Depending on the type of file you are editing, the appropriate editor is displayed in the editor area. For example, if you are editing a .TXT file, a text editor is displayed in the editor area. The figure below shows an editor open on the file *sample.txt*. The name of the file appears in the tab of the editor. If an asterisk (*) appears at the left side of the tab this indicates that the editor has unsaved changes. If you attempt to close the editor or exit the Workbench with unsaved changes you will be prompted to save your editor's changes.

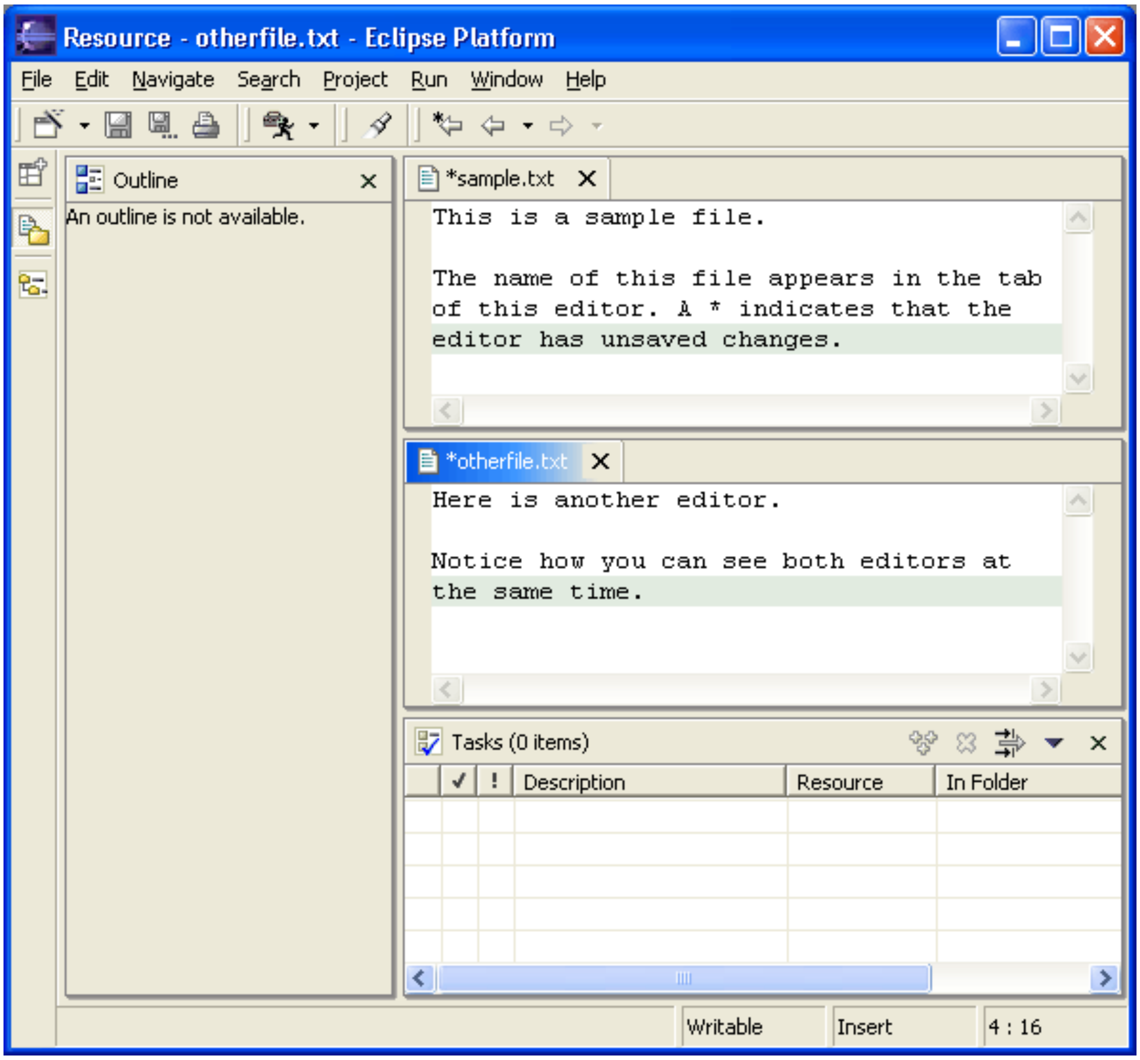

Resource - sample.txt - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Sample Project
.project
sample.txt
*sample.txt
Outline
Tasks (0 items)
Description
Resource
In Folder
Sample Project/sample.txt

```
This is a sample text file.

The name of this file is sample.txt and the name
appears in the tab above. A * indicates that the
editor has unsaved changes.
```

An outline is not available.

When an editor is active the Workbench menu bar and toolbar contain operations applicable to the editor. When a view becomes active, the editor operations are disabled. However, certain operations may be appropriate in the context of a view and will remain enabled.

You can stack editors in the editor area and activate individual editors by clicking the tab for the editor. You can also tile editors side−by−side in the editor area so their content can be viewed simultaneously. In the figure below editors for sample.txt and otherFile.txt have been placed one above the other. We will discuss how you can rearrange views and editors later in this tutorial.



An outline is not available.

```
This is a sample file.

The name of this file appears in the tab
of this editor. A * indicates that the
editor has unsaved changes.
```

```
Here is another editor.

Notice how you can see both editors at
the same time.
```

If you open a resource that does not have an associated editor, the Workbench will attempt to launch an external editor registered with your platform. These external editors are not tightly integrated with the Workbench and are not embedded into the Workbench's editor area.

You can also cycle through the editors using the back and forward arrow buttons in the toolbar and by using the Ctrl+F6 accelerator. The arrow buttons move through the last mouse selection points and allow you to potentially move through several points in a file before moving to another one. Ctrl+F6 pops up a list of currently selected editors with the editor used before the current one selected by default.

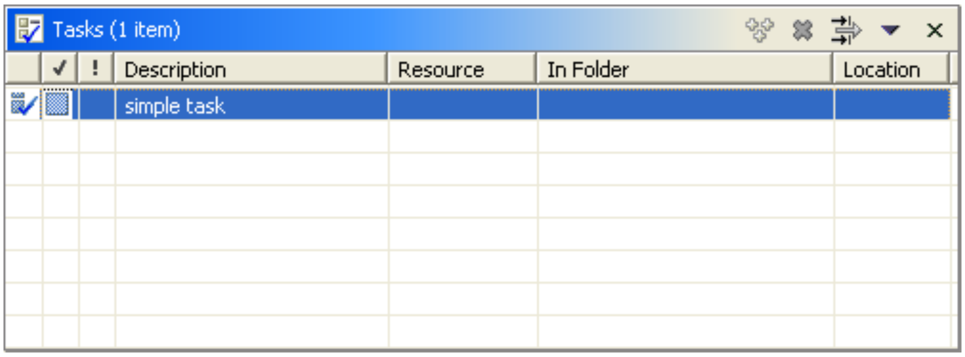
WIN On Windows the Workbench will also attempt to launch the editor in−place as an OLE document editor. For example editing a DOC file will cause Microsoft Word to be opened in−place within the Workbench if you have Microsoft Word installed on your machine. If you have not installed Microsoft Word, Word Pad will open instead.

Views

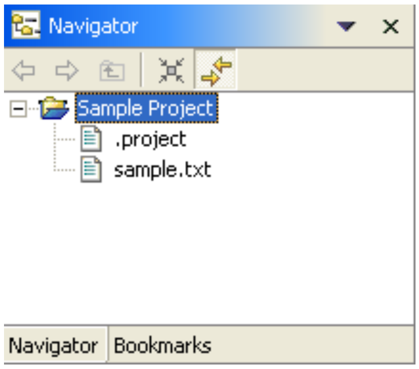
Views support editors and provide alternative presentations or navigations of the information in your Workbench. For example:

- The Bookmarks view displays all bookmarks in the Workbench along with the names of the files with which the bookmarks are associated.
- The Navigator view displays the projects and other resources that you are working with.

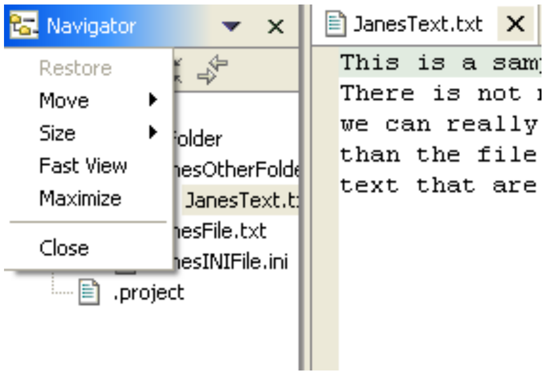
A view might appear by itself or stacked with other views in a tabbed notebook.



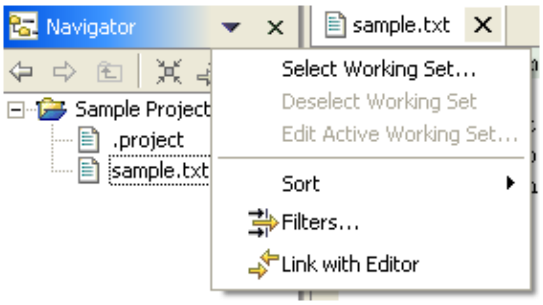
To activate a view that is part of a tabbed notebook simply click on its tab. As we will soon discover, the Workbench provides a number of quick and easy ways to configure your environment, including whether the tabs are at the bottom or top of your notebooks.



Views have two menus, the first is a menu associated with the top left area of the view which allows you to manipulate the view in much the same manner as the menu associated with the Workbench window.

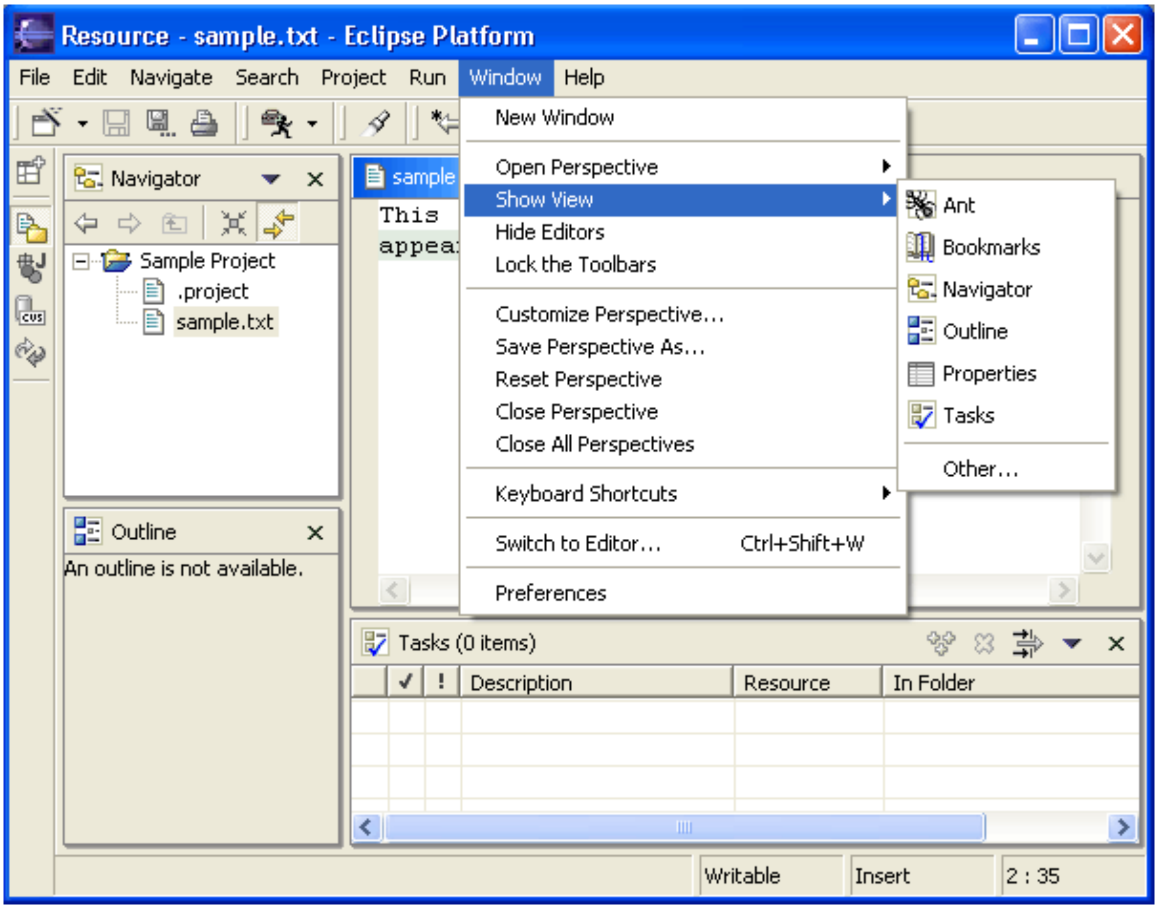


The second menu is accessed by clicking on the down arrow ▼. The view pull−down menu typically contains operations that apply to the entire contents of the view, but not to a specific item shown in the view. Operations for sorting and filtering are commonly found on the view pull−down.



If you decided to experiment by rearranging your Workbench, this is probably a good opportunity to use the **Window** > **Reset Perspective** menu operation. The reset operation restores the layout to its original state.

You can display a view by selecting it from the **Window** > **Show View** menu. A perspective determines the views you are likely to need and shows these on the **Show View** submenu. Additional views are available by choosing **Other...** at the bottom of the **Show View** submenu. As we will see later this is just one of the many features that allows you to build your own custom work environment.



Our first project

Now that we have familiarized ourselves with the basic elements of the Workbench, let's create our first project. New projects, folders, and files can be created using several different approaches. In this section we will create resources using three different approaches:

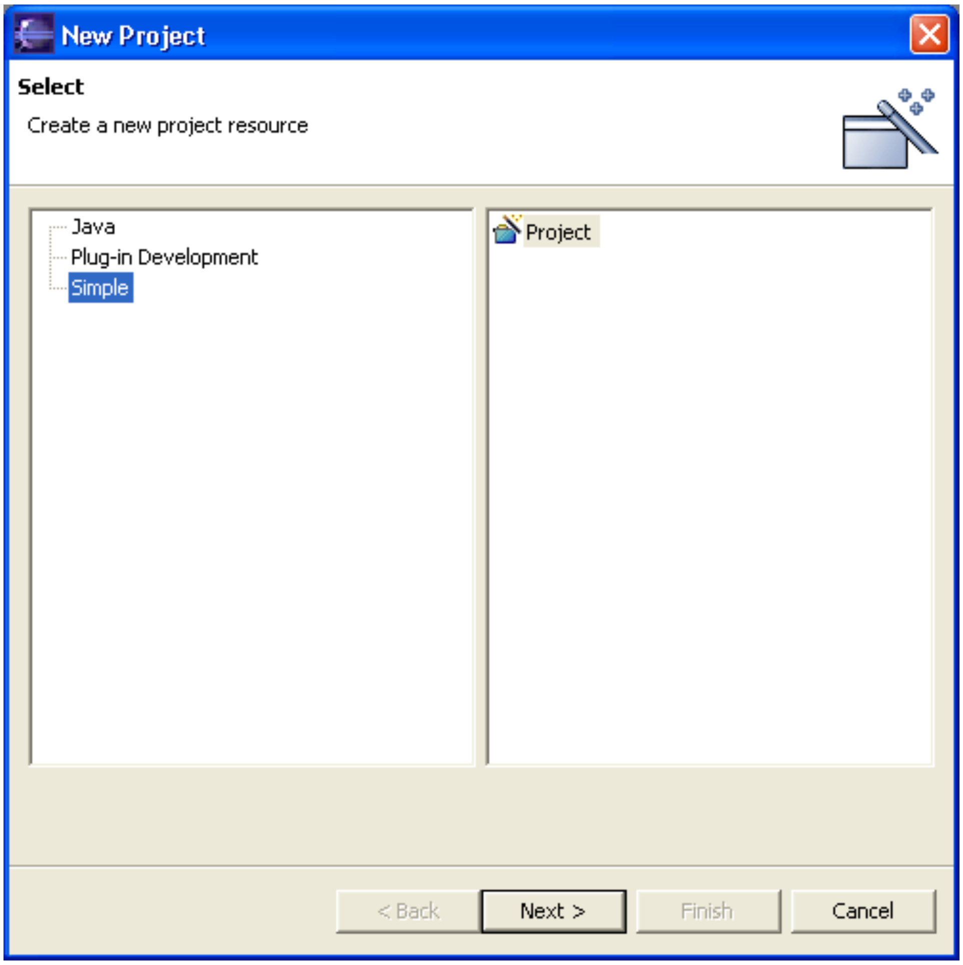
1. **File** menu
2. Navigator context menu
3. New Wizard button

We will start by creating a project using the **File** menu. Once we have our project we will create a folder and file.

Using the File menu

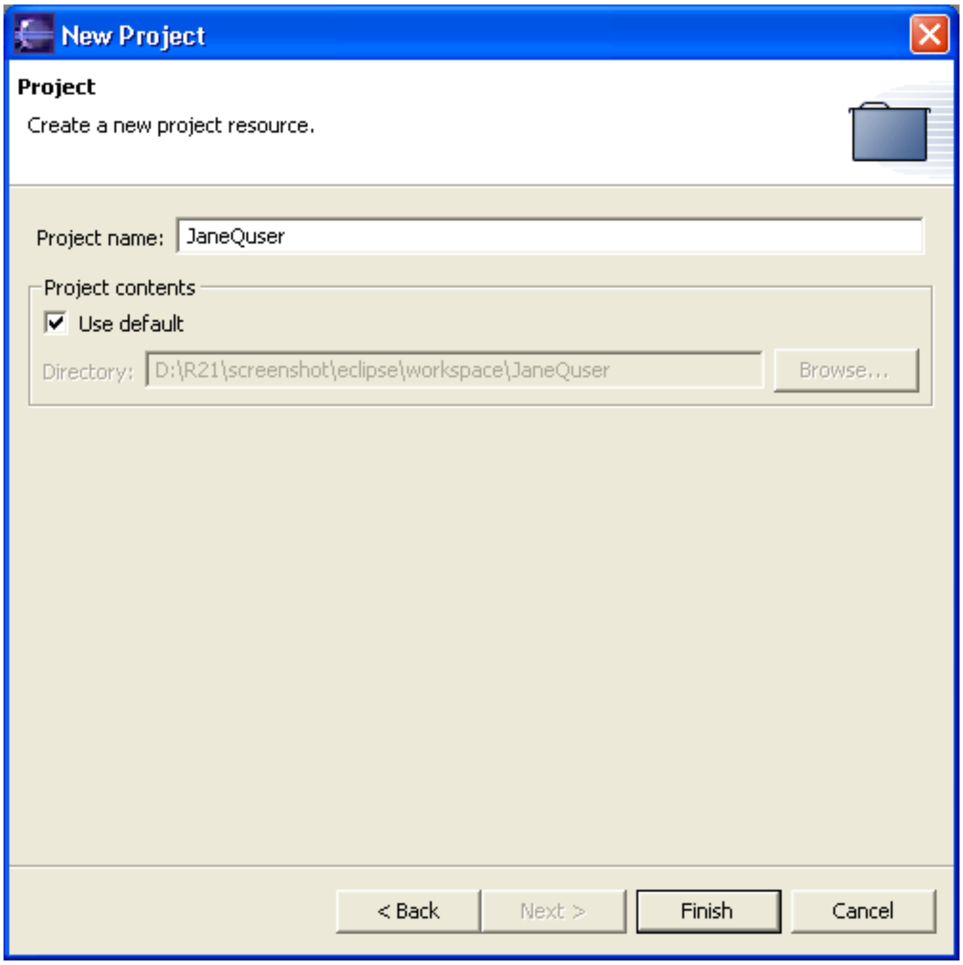
You can create new resources by using the **File** > **New** menu on the Workbench menubar. Start by creating a simple project as follows:

1. From the menubar, select **File** > **New** > **Project...**



2. In the left pane, select **Simple**, and on the right, select **Project**; then click **Next**.

3. In the **Project name** field, type your name as the name of your new project. Do not use spaces or special characters in the project name (e.g., "JaneQuser").
4. Leave the box checked to use the default location for your new project. Click **Finish** when you are done.



New Project

Project

Create a new project resource.

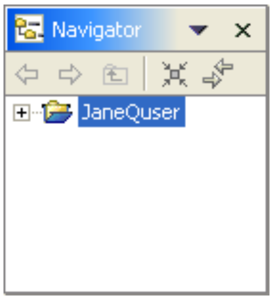
Project name: JaneQuser

Project contents

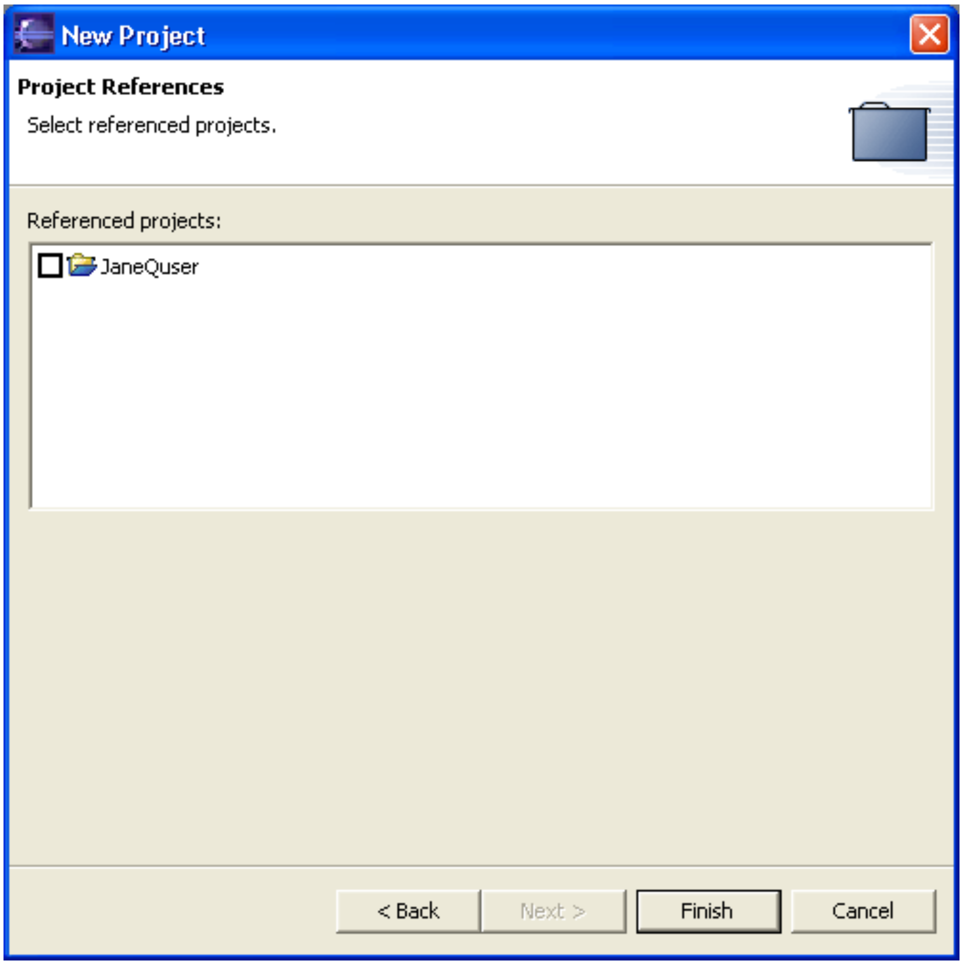
[✓] Use default

Directory: D:\R21\screenshot\eclipse\workspace\JaneQuser

If you sneak a peak at the Navigator view, you will see that it now contains the simple project we just created.



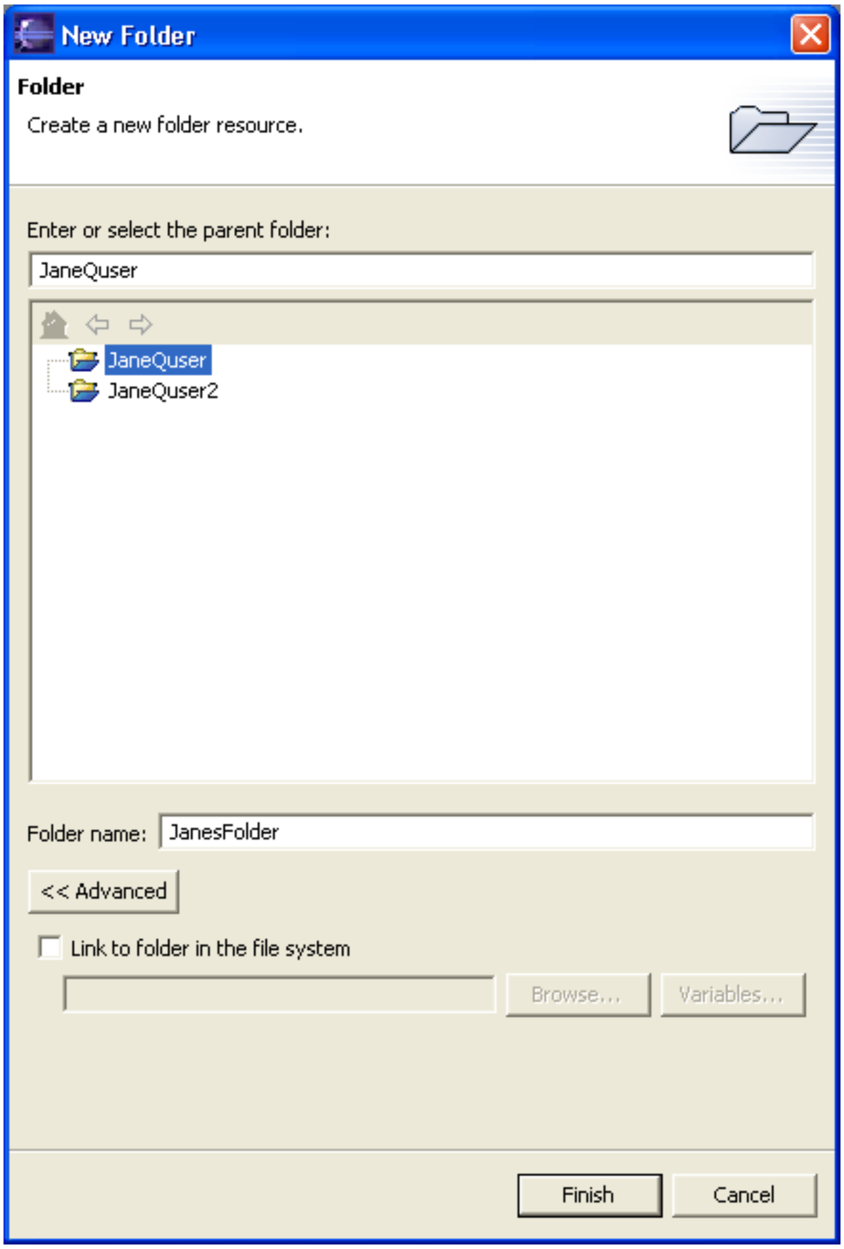
Create a second project called JaneQuser2 using the same steps, but instead of clicking **Finish**, click **Next**. At this point you can specify other projects that project JaneQuser2 depends on. Since we want to create two independent projects we will not select any of the projects in the **Referenced projects** table. Click **Finish** to create your second simple project.



Using the popup

Now that we have our project we will create a folder. We will create our folder using the Navigator view's popup menu.

1. Activate the Navigator view and select the project JaneQuser (the first project we created in the Navigator view). From the view's popup menu choose **New > Folder**.
2. In the New Folder wizard, your project name appears by default in the **Enter or select the parent folder** field. This is because we chose to create the new folder from your project's context menu.
3. In the **Folder name** field, type a unique name for your new folder (e.g., using your first and last names in the title). Do not use spaces or special characters in the folder name (e.g., "JanesFolder").



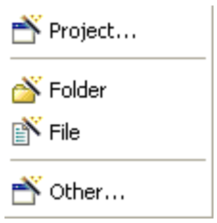
4. Click **Finish** when you are done. The Navigator view will update to show your newly created folder.

Note that there is now also an **Advanced** button. This button when selected allows you to enter a location outside of a projects hierarchy as the location for one of its folders. This is called a linked folder.

Using the New button

We have seen how to create resources using **File** > **New** and **New** from the Navigator's context menu. We will now create a file for our project using the third alternative, the toolbar **New** button.

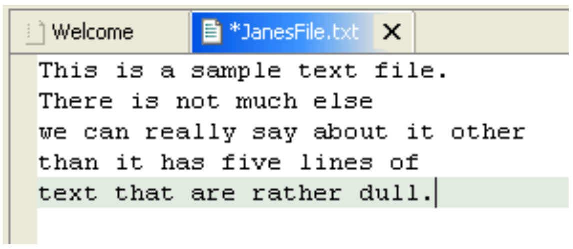
1. Select the folder JanesFolder in the Navigator view.
2. In the Workbench window's toolbar, activate the drop−down menu on the New Wizard button and select **File**. To activate the drop−down menu simply click on the down arrow.



3. In the New File wizard, your folder's name and path already appear by default in the **Enter or select the parent folder** field. This is because you chose to create the new file while this folder was selected in the Navigator view and the view was active.
4. In the **File name** field, type a unique name for a new text file, including the .txt file extension. Do not use spaces or special characters in this file name (e.g., "JanesFile.txt").
5. Click **Finish** when you are done.
6. The Workbench has an editor capable of editing text files. A text editor is automatically opened on the newly created file.
7. In the text editor, type in the following five lines:

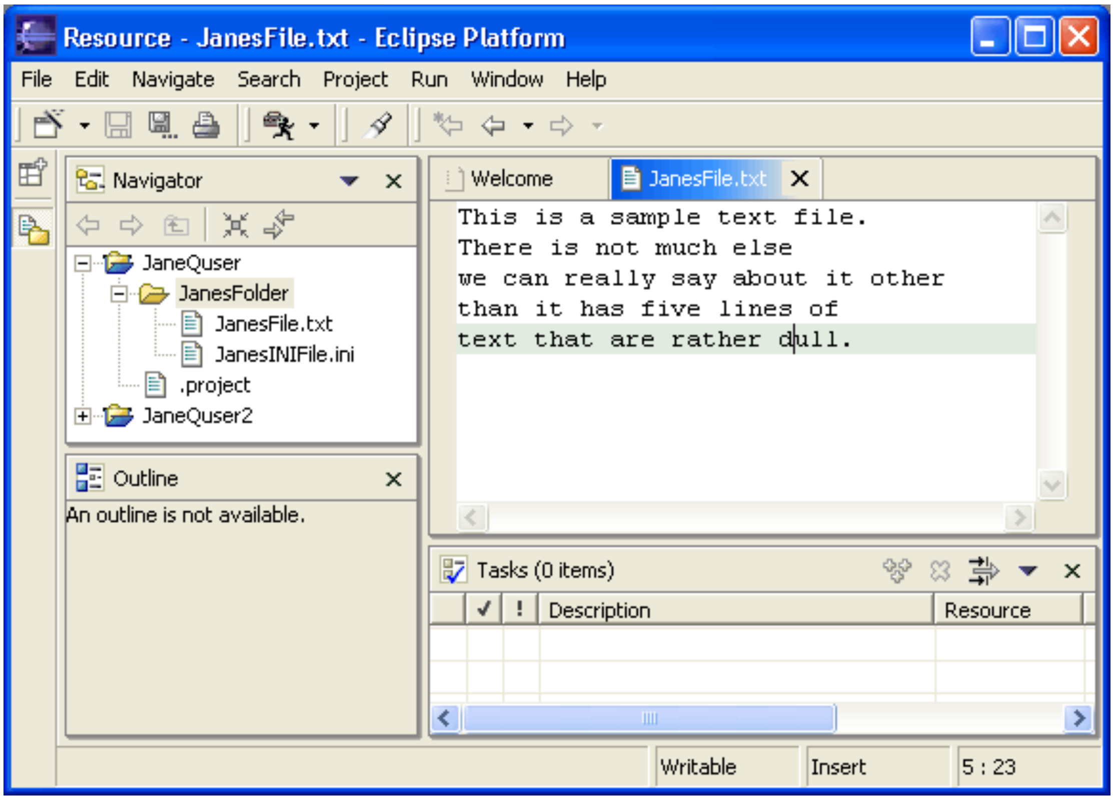
   This is a sample text file.
   There is not much else
   we can really say about it other
   than it has five lines of
   text that are rather dull.

   Notice that the editor tab has an asterisk (*) at the left of the filename. The asterisk indicates that the editor has unsaved changes.

   ```
   This is a sample text file.
   There is not much else
   we can really say about it other
   than it has five lines of
   text that are rather dull.
   ```
8. In the Workbench window's toolbar, click the Save button to save your work.
9. In the Navigator view ensure your folder is still selected and the Navigator view is active.
10. Click the New Wizard button in the Workbench toolbar. Previously we clicked on the drop−down arrow of the New button. Here we clicked on the button itself which has the same effect as choosing **File > New > Other...**
11. In the New wizard, click **Simple** in the left pane, and in the right pane, click **File**. Then click **Next**.
12. Once again, your folder's name and path appears by default in the **Enter or select the parent folder** field.
13. In the **File name** field, type a unique name for an .ini file. Do not use any spaces or special characters in the file name (e.g., "JanesINIFile.ini"). Click Finish when you are done.
14. Since the Workbench does not have any editors registered for .ini files, it may launch an external editor on the file if one is registered with the operating system. For the moment, close the editor.

Now that we have created our resources, the Navigator view shows our two projects, the folder and its two files. To the right of the Navigator view is the text editor open on the first file we created (JanesFile.txt). If you look closely you can also see a second, inactive editor tab, "Welcome".



```
This is a sample text file.
There is not much else
we can really say about it other
than it has five lines of
text that are rather dull.
```

Click on the Welcome editor tab. Now select the file JanesINIFile.ini in the Navigator view. Then select the **Link with Editor** button on the Navigator view. Lastly, click on the editor tab for JanesFile.txt. Notice how the Navigator updated itself to select the file you are currently editing (JanesFile.txt). If you don't like this automatic update you can easily turn it off by deselecting the **Link with Editor** button.

Closing an editor

Now that we have a couple of editors open, we should learn how to close them.

1. Select the JanesFile.txt editor tab.
2. In the text area add a 6th line of text:

   This is a 6th line
3. To close the editor, do one of the following options:
   - Click the close button ("X") JanesFile.txt ✕ in the tab of any open editor.
   - Choose **File** > **Close** from the menubar.
4. Note that you are prompted to save the file before the editor is closed.
5. Click OK to save your changes and close the editor.

If you closed the editor using **File** > **Close** you may have noticed the option **File** > **Close All**. This is a quick way to close all of your open editors. If you choose **File** > **Close All**, you will be prompted to choose which editors with unsaved changes you want to save.

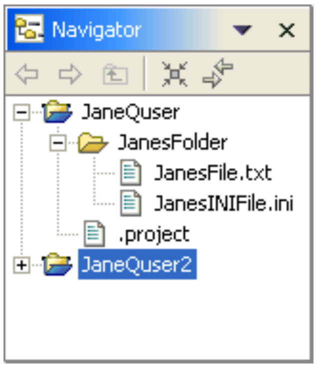
If you choose **Window** > **Preferences** then **Workbench** > **Editors**, you will find the preference to close editors automatically. There you can configure how many editors are opened before editors are reused and what should happen when all editors have unsaved changes.

Navigating resources

In this section we will work with the Navigator and Tasks views. These views are initially part of the resource perspective. If you want to experiment with other views, you can see them by using the **Window** > **Show View** menu.

One important view to become familiar with is the Navigator view, which displays information about the contents of the Workbench and how the resources relate to each other in a hierarchy.

In the Workbench, all resources reside in projects. Projects can contain folders and/or individual files.



Opening resources in the Navigator

Using the Navigator there are several ways to open an editor.

1. In the Navigator select the file JanesFile.txt
2. To open an editor on the file choose one of the following approaches:

   - To edit a resource using the default editor for that resource you can either double click on the resource (in the Navigator), or select the resource and choose **Open** from its popup menu.
   - If you want to use a specific editor to edit the resource, start by selecting the resource in the Navigator, and choose the **Open With** option from the popup menu.

   The Workbench remembers the last editor that you used for editing a specific file. This makes it easier to use the same editor down the road.

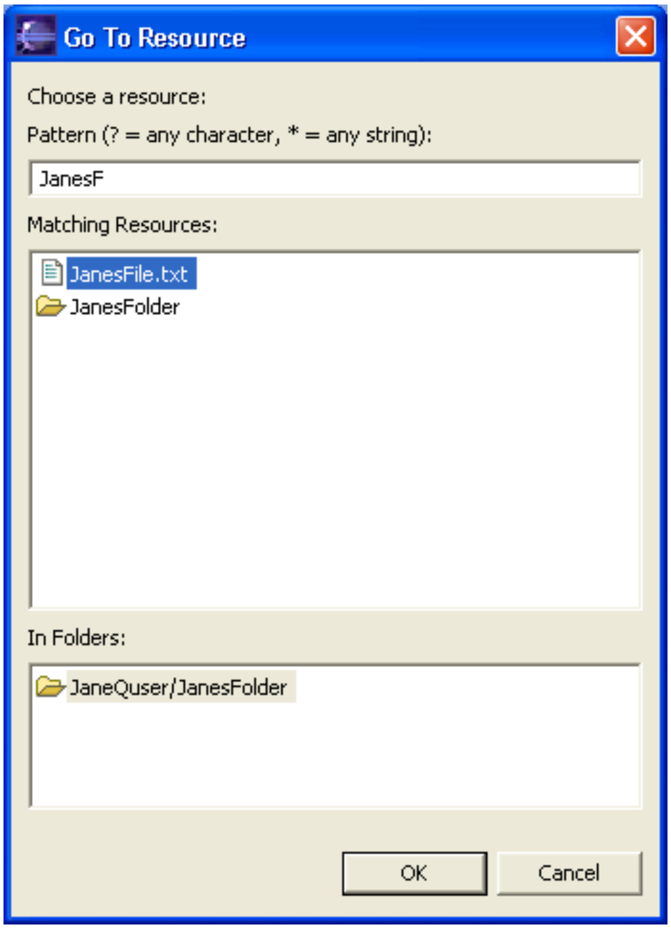
You can configure the default editors using **Window** > **Preferences** > **Workbench** > **File Associations**.

Go To

The Navigator's Go To operation makes it easy to jump to a specific resource in the Navigator.

1. Select the Navigator view. Its title bar will be highlighted (according to your operating system's color scheme).
2. From the pull−down menu choose **Navigate > Go To > Resource...**.
3. In the Go To Resource dialog type "JanesF" into the **Pattern** field at the top of the dialog.

   As you type the filename the dialog filters the set of possible matches based on what you have entered so far.
4. Select JanesFile.txt from the **Matching Resources** field and click OK or simply press Enter.
5. The Navigator will select the file JanesFile.txt.

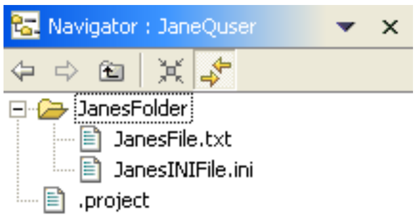


Go Into

The Navigator shows all of the resources in your Workbench. You can pick a project or folder, "Go Into it" and hide all other resources. Let's try this with our JaneQuser project.

1. Select the Navigator view. Its title bar will be highlighted (according to your operating system's color scheme).

2. Select the JaneQuser project.
3. Choose **Go Into** from the popup menu.
4. The Navigator now shows only the contents of the JaneQuser project. The title of the Navigator shows the name of the resource you are currently looking at.



5. You can use the back, forward and up buttons to move between showing all resources and showing only the JaneQuser project.

   Click the back button to show all of the resources.

Show me the files

Right about now you're probably asking yourself "Where are my files stored?". Perhaps you are even a little concerned they might be stored in some mystical repository. Answering this question is easy.

1. Start by opening your operating system's file system explorer.
2. Navigate to the location of where you installed the Workbench. You should see a sub−directory called "workspace".
3. If you look in the workspace sub−directory you will see the projects, folders and files that we created.

All of your resources are stored as normal files and folders on your machine. This allows you to use other tools to work with your files. Those tools can be completely oblivious to the Workbench. In a later section we will look at how to work with external editors that are not integrated into the Workbench.

Exporting files

You can export files from the Workbench either by:

- WIN dragging and dropping to the file system, or
- WIN copying and pasting to the file system, or
- using the Export wizard.

Drag and drop or copy and paste

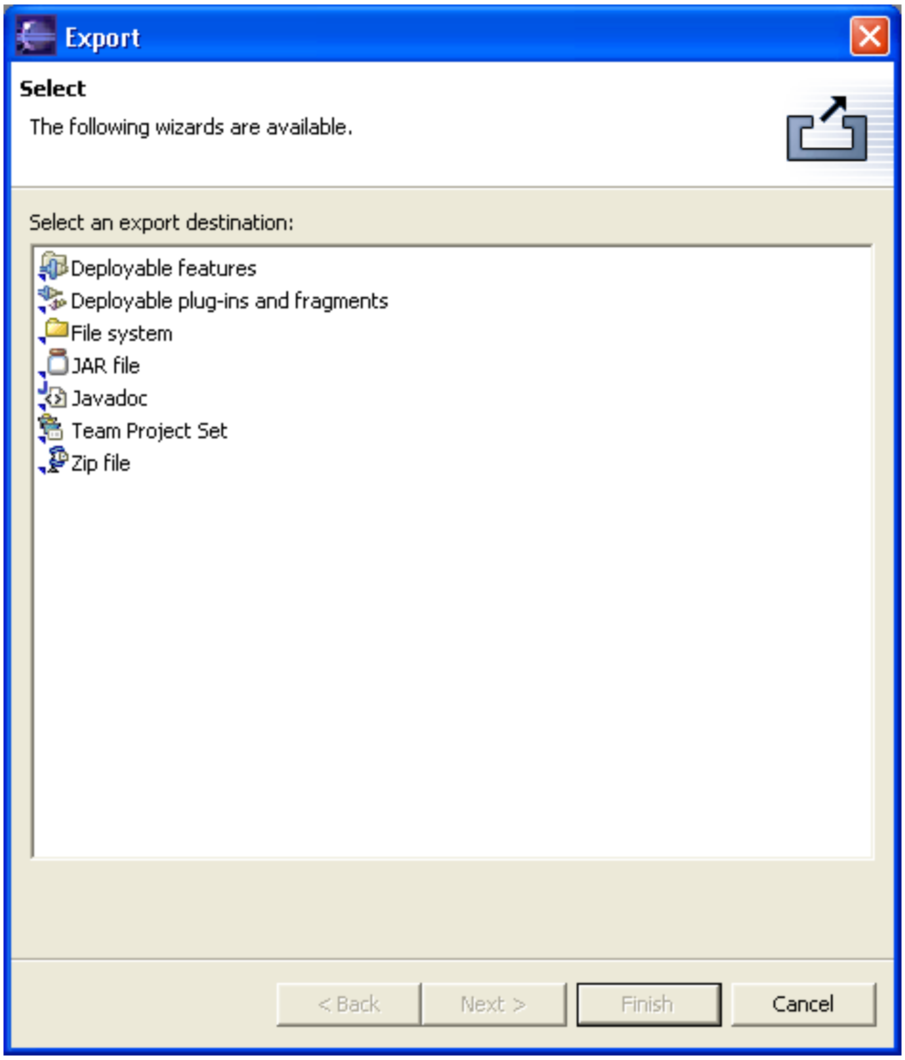
WIN You can use your operating system's file system explorer to export a copy of a folder or file from the Workbench to the file system.

1. Open your operating system's file system explorer.
2. Drag the file JanesFile.txt from the Navigator to the file system explorer.
3. Hold down the Ctrl key while dragging to ensure the file is copied. Otherwise, depending on your operating system, the file may be removed from the workspace.
4. The export can also be achieved by selecting the file in the Navigator and choosing **Edit > Copy**, then pasting it in the file system explorer.

Export wizard

You can use the Export wizard to export from the Workbench to the file system.

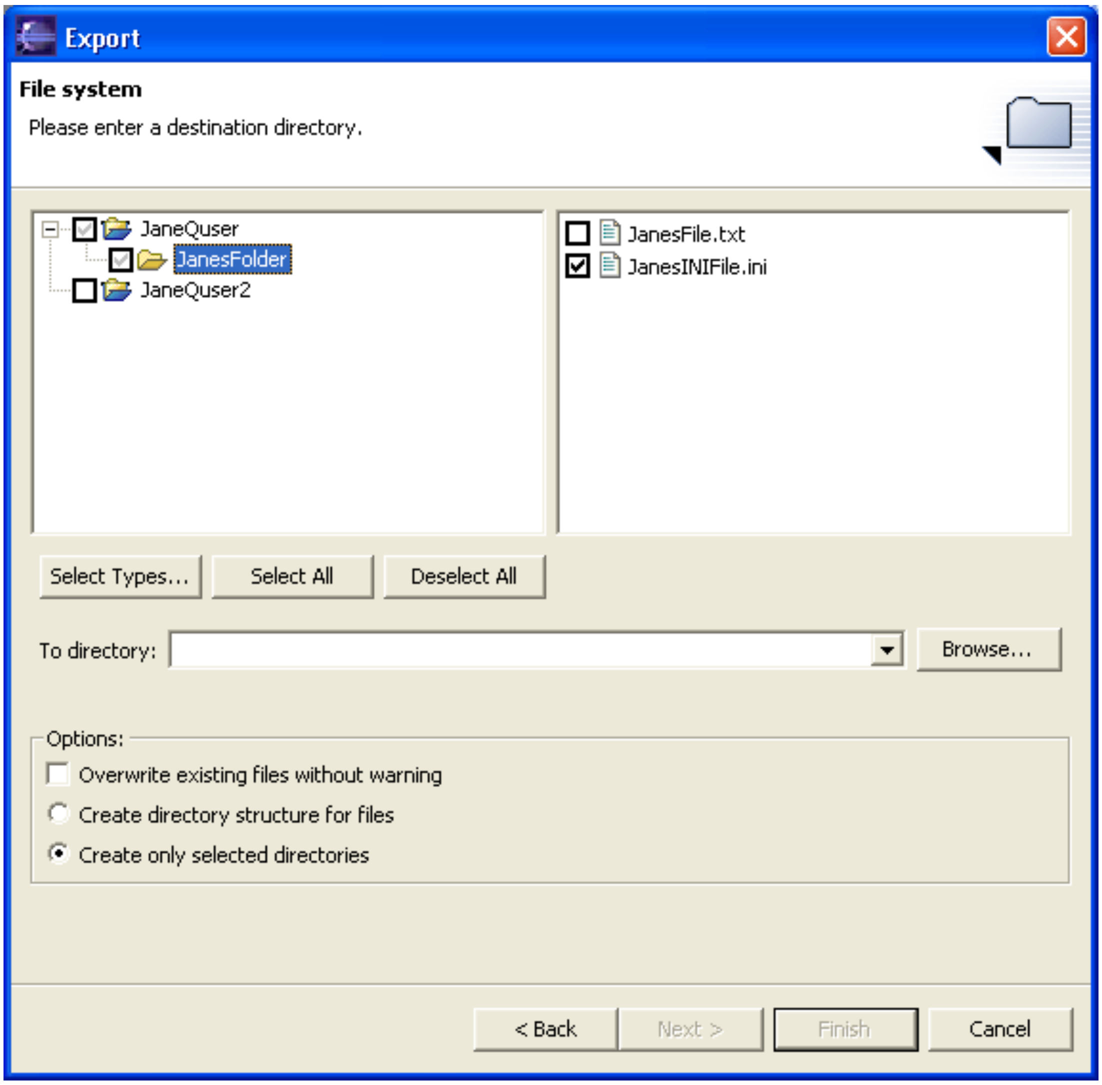
1. Select the project JaneQuser in the Navigator.
2. From the popup menu, select **Export**.
3. In the Export wizard, select **File system**, then click **Next**.



4. Expand JaneQuser project, and click on JanesFolder. In the right pane ensure that only JanesINIFile.ini is checked. Notice the folder and project in the left pane now have a grayed checkbox indicating that some, but not all, of their contents will be exported.

   You can use the **Select Types** button to filter the types of resources you want to export.

   If we had known ahead of time that we only wanted to export JanesINIFile.ini we could have simply selected it in the Navigator and chosen **File** > **Export**. The Export wizard would have automatically ensured it was the only file checked for export.



5. In the **To directory** field, type or browse to select a location in the file system where you want the exported resources to reside.

   If you enter the name of a directory that does not exist the Export wizard will offer to create it for you once you press **Finish**.
6. In the **Options** area, you can choose to:
   - Overwrite existing resources without warning
   - Create directory structure for files or Create only selected directories
7. Click **Finish** when you are done.

Importing files

You can import files into the Workbench either by :

- WIN dragging and dropping from the file system, or
- WIN copy and pasting from the file system, or
- Using the Import wizard.

In this section we will import the two files we just exported and place them into our second project JaneQuser2.

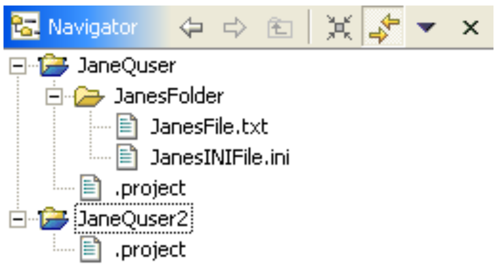
Drag and drop or copy and paste

WIN You can use your operating system's file system browser to copy folders and files from the file system into the Workbench.

Note: You must drag the resource(s) to the exact location in the Navigator view hierarchy where you want the resources to reside; you cannot simply drag and drop them onto a blank area in the Navigator view.

1. Open your operating system's file system explorer.
2. Locate the file JanesFile.txt that we recently exported and drag it to a specific location in the Navigator view in the Workbench.

   When dragging resources into the Navigator, the project/folder that you are trying to drop the resource into will be selected.

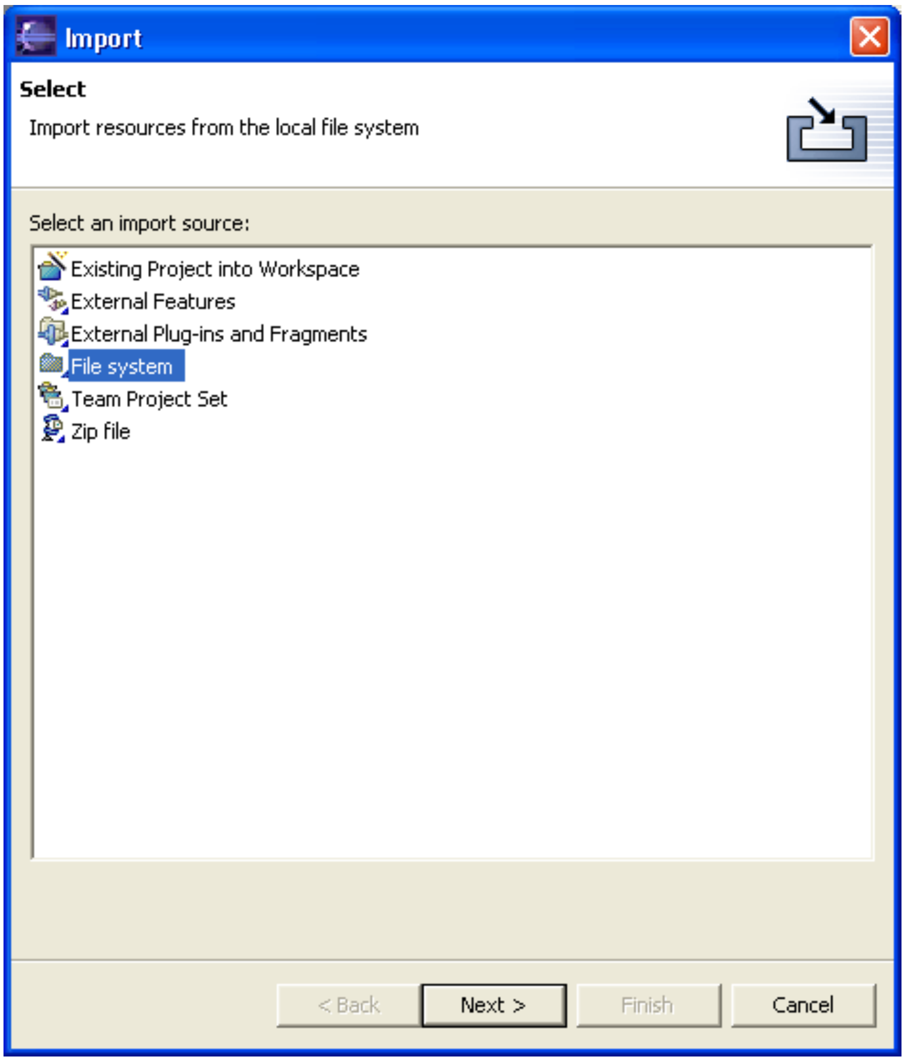
   

   Drag the file over JaneQuser2 and release the mouse button.
3. Notice that the file is copied into the Workbench and placed into JaneQuser2.
4. This can also be achieved by copying the file in the file system explorer, then selecting the destination in the Navigator and choosing **Edit** > **Paste**.

Import wizard

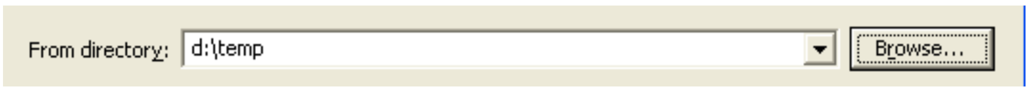
You can use the Import wizard to copy resources into the Workbench.

1. Select the project JaneQuser.
2. Select **Import** from the popup.
3. In the Import wizard, select **File system**, then click **Next**.



4. In the **From directory** field, type or browse to select the directory containing the file JanesINIFile.ini that we recently exported.

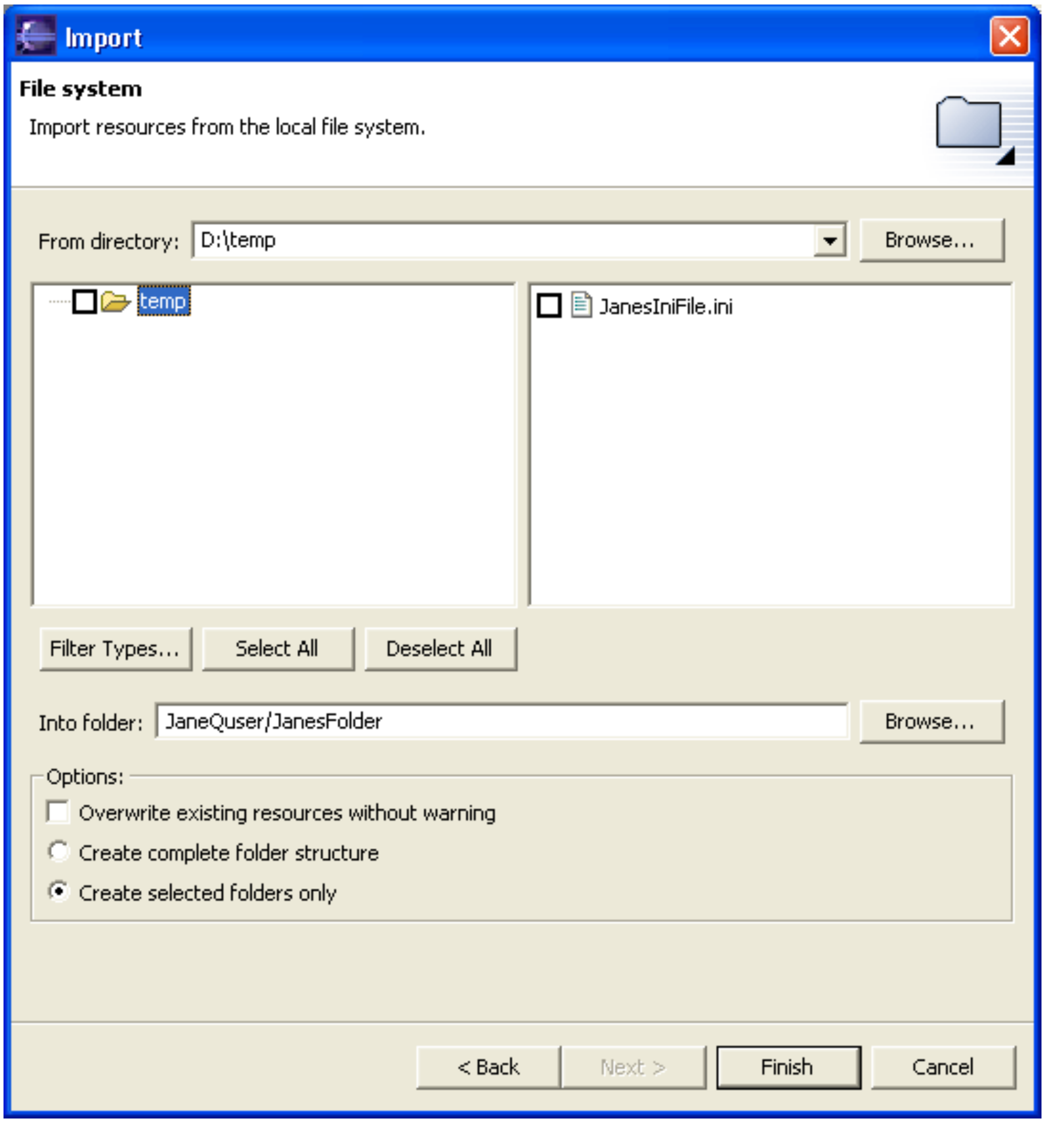
   Recent directories that you have imported from are shown on the **From directory** field's combo box.



5. In the right pane check the file JanesINIFile.ini

   Checking a folder in the left pane will import its entire contents into the Workbench. A grayed checkbox (as shown below) indicates that only some of the files in the folder will be imported into the Workbench.

   You can use the **Filter Types** button to filter the types of files you want to import.


Import

File system

Import resources from the local file system.

From directory: D:\temp

☐ temp

☐ JanesIniFile.ini

Into folder: JaneQuser/JanesFolder

Options:

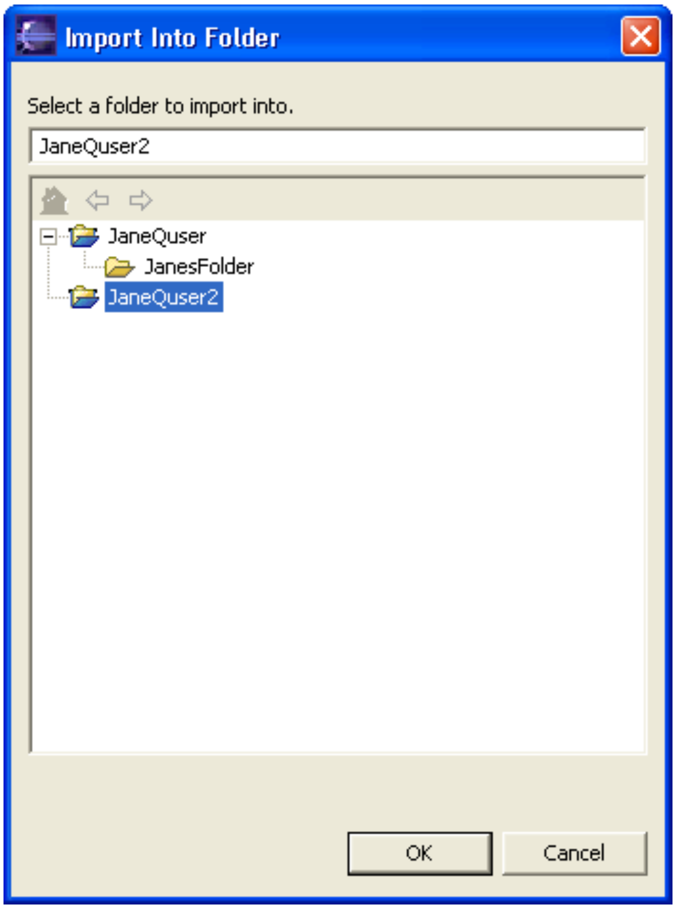
☐ Overwrite existing resources without warning

○ Create complete folder structure

● Create selected folders only

6. The **Into folder** field should already be filled in with the name of the project (JaneQuser).
7. You can change the destination project or folder by clicking **Browse**;

   Click the **Browse** button and choose our second project JaneQuser2.



8. In the **Options** area, you can choose to:
   - Overwrite existing resources without warning
   - Create complete folder structure or Create selected folders only
9. Click **Finish** when you are done. The file JaneINIFile.ini is now shown in the Navigator in the project JaneQuser2.

Deleting resources

Now that we have imported a few files into our second project (JaneQuser2) we will see how easy it is to delete the project.

1. Select project JaneQuser2 in the Navigator view.
2. To delete the project do one of the following:
   - From the project's pop−up menu choose **Delete**
   - Press the DEL key
   - Choose **Edit** > **Delete** from the pull−down menu
3. You will be asked to confirm the deletion and also choose the type of deletion you want to perform.

   You can:

   - Delete the contents of the project from the file system

♦ Delete the project from the workspace but keep its contents in the file system

Just accept the default (do not delete contents) and click **Yes**. Note that this is only an option for projects. Files and folders are always deleted from the file system when deleted.

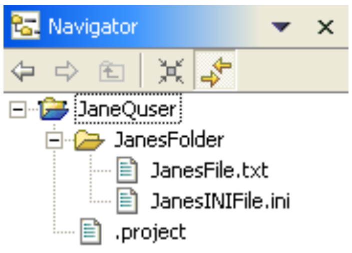
The same steps work for any resource shown in the Navigator.

Working with other editors

We have seen how to import and export resources from the Workbench. In this section we will look at how to edit Workbench resources using the following three approaches:

- External editors launched by the Workbench
- Embedded OLE documents WIN
- External editors launched without the Workbench's knowledge

Before we continue, take a moment and confirm that the Navigator contains the following resources:



External editors

When you open a resource the Workbench first consults its list of registered editors. If no registered editors are found for the resource the Workbench checks with the underlying operating system to determine if it has any editors registered for the particular file type. If an editor is located, the Workbench will automatically launch that editor. This type of editor is referred to as an external editor because it does not show up as an editor tab in the Workbench.

1. Select the file JanesINIFile.ini.
2. Double−click the file in the Navigator view to launch the external editor.

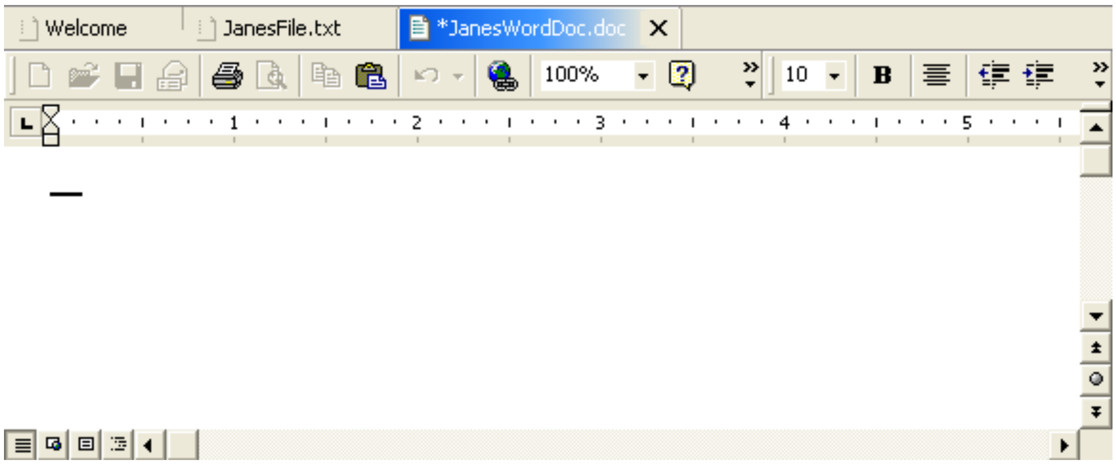
   If an editor for INI files is not registered with your underlying operating system the Workbench will attempt to use its own default text editor. If this happens and you are still determined to see an external editor you can import another file (see previous sections) that is associated with a third party editor. Double click again on this new file and you should see your favorite editor open in its own window.

   WIN

If you discovered that your favorite editor magically opens up inside the Workbench don't panic. The Workbench supports OLE document editors, and your editor happens to provide OLE document support allowing it to be opened in its own window, or embedded inside another window like the Workbench. We'll discuss this in more detail in the next section.

Embedded editors

WIN The Workbench supports OLE document editors.



1. Select JanesFolder in the navigator.
2. Create the file JanesWordDoc.doc.
3. Notice how the Workbench automatically opens the OLE document editor in place and integrates its pull−down menu options into the menu bar. There should also be an editor tab for JanesWordDoc.doc.
4. Make a change to the file.
5. Close the editor by clicking the X on the editor tab; when prompted, save its contents.
6. Reopen the file by double−clicking it in the Navigator view.

Editing files outside the Workbench

In a previous section we launched an external editor on the file JanesINIFile.ini by double clicking on it in the Navigator. We can also use the same external editor by launching it outside of the Workbench.

1. Close any editors that are open on JanesINIFile.ini.
2. In your file system explorer, navigate to the directory where you installed the Workbench and go into the workspace sub−directory.
3. Edit the file JanesINIFile.ini and save it . Do not use the Workbench's Navigator to open the editor.
4. Return to the Workbench and in the Navigator view, select the project JaneQuser.
5. Choose **Refresh** from the project's context menu. This instructs the Workbench to look for any changes to the project that have been made in the local file system by external tools.
6. Select the file JanesINIFile.ini.
7. For a little variety choose **Open With** > **Text Editor** from the file's popup menu.

8. Observe that the Workbench's own default text editor is opened.

   In the default text editor verify the changes you made externally are reflected in the Workbench.

As we've seen the Workbench stores all of its resources in the local file system. This means you can use your system file explorer to copy files from the Workbench's workspace area even when the Workbench is not running. You can also copy resources into the workspace directory. You should use Refresh to update the Workbench with any changes you have made outside the Workbench.

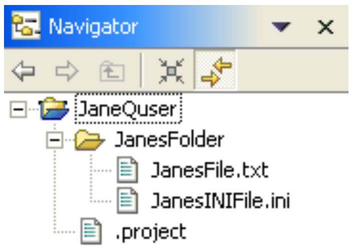
Copying, renaming and moving

You can copy, move and rename Workbench resources using popup menu operations in the Navigator. In this section we will copy and rename several of the files that we have created.

Before we start copying our files we need to do some setup:

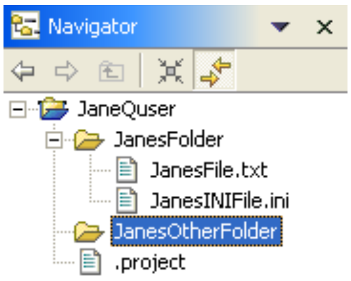
Setup

1. In the Navigator view, delete the file JanesWordDoc.doc. Your Navigator should look like this:



2. Double click on JanesFile.txt and ensure that it contains the following text.

   This is a sample text file
   There is not much else
   we can really say about it other
   than it has five lines of
   text that are rather dull.
3. Close the editor on JanesFile.txt.
4. Select the project JaneQuser. Using the project's pop−up menu create a folder named JanesOtherFolder.



Copying

Let's start by copying JanesFile.txt to our new folder (JanesOtherFolder). Once we have copied our file we will then rename it.

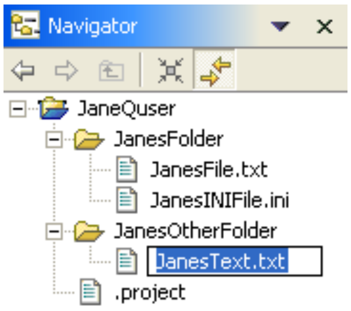
1. Ensure that you have performed the setup described in the introduction to this section.
2. In the Navigator view, select JanesFile.txt.
3. From the file's context menu, select **Copy** (or Ctrl+C)
4. In the Navigator view, select JanesOtherFolder as the destination.
5. From the folder's context menu, select **Paste** (or Ctrl+V).

We have seen how to copy files using the copy operation. It is also possible to copy files by holding down the Ctrl key while dragging a file from one folder to another folder.

Renaming

Now that we have copied JanesFile.txt from JanesFolder to JanesOtherFolder we are ready to rename something.

1. In the Navigator view, select JanesFile.txt in JanesOtherFolder.
2. From the file's context menu, select **Rename**.
3. The Navigator overlays the file's name with a text field. Type in JanesText.txt and press Enter.



If you decide you do not want to rename a resource, you can press Escape to dismiss the text field.

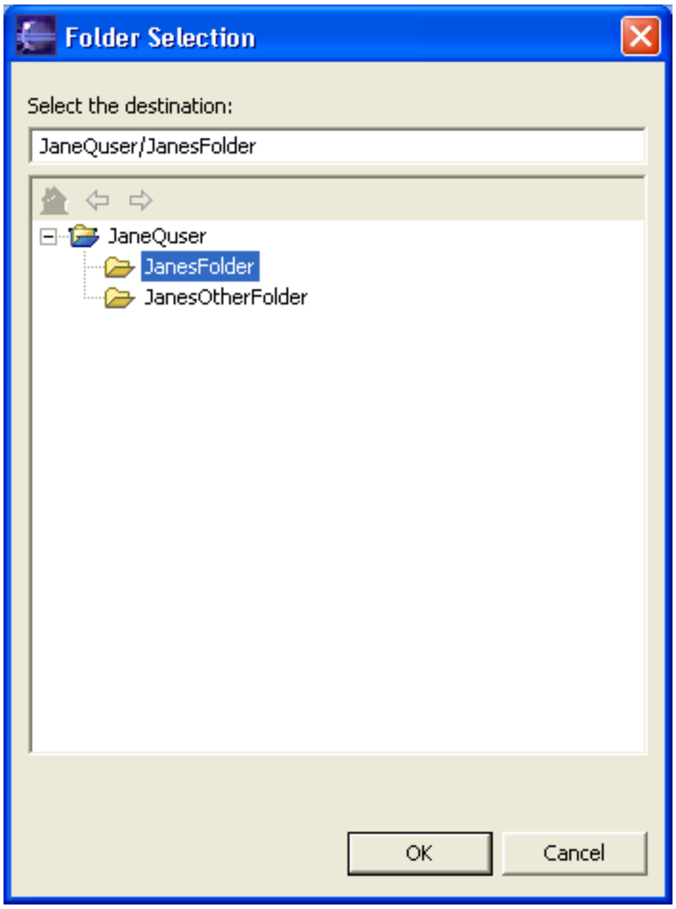
You are probably wondering whether or not copy and rename work on anything besides files. To answer this question we can try renaming a folder.

1. In the Navigator view, select the folder JanesOtherFolder.
2. From the folder's context menu choose **Rename**.
3. Once again the Navigator overlays the folder name with an entry field to allow you to type in the new name. Change the folder name to be JanesSecondFolder.
4. Rename the folder back to its original name (JanesOtherFolder).

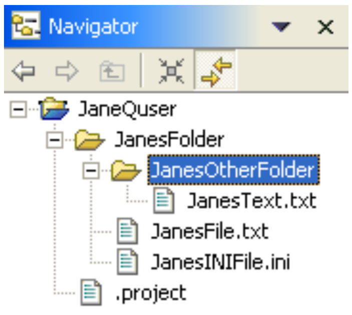
Moving

We have copied and renamed several of our resources. Now it's time to move some resources around. We will move our JanesOtherFolder and its file to be a sub−folder of our original folder JanesFolder.

1. In the Navigator view, select JanesOtherFolder.
2. From the file's context menu, select **Move**.
3. In the Folder Selection dialog choose JanesFolder and click **OK**.



4. In the Navigator JanesFolder now contains JanesOtherFolder. Expand JanesOtherFolder and confirm that it contains JanesText.txt.



We have seen how to move files using the move operation. It is also possible to move files by dragging a file from one folder to another folder. You may recall that to copy files from one folder to the other you need to hold down Ctrl while performing the drag and drop operation.

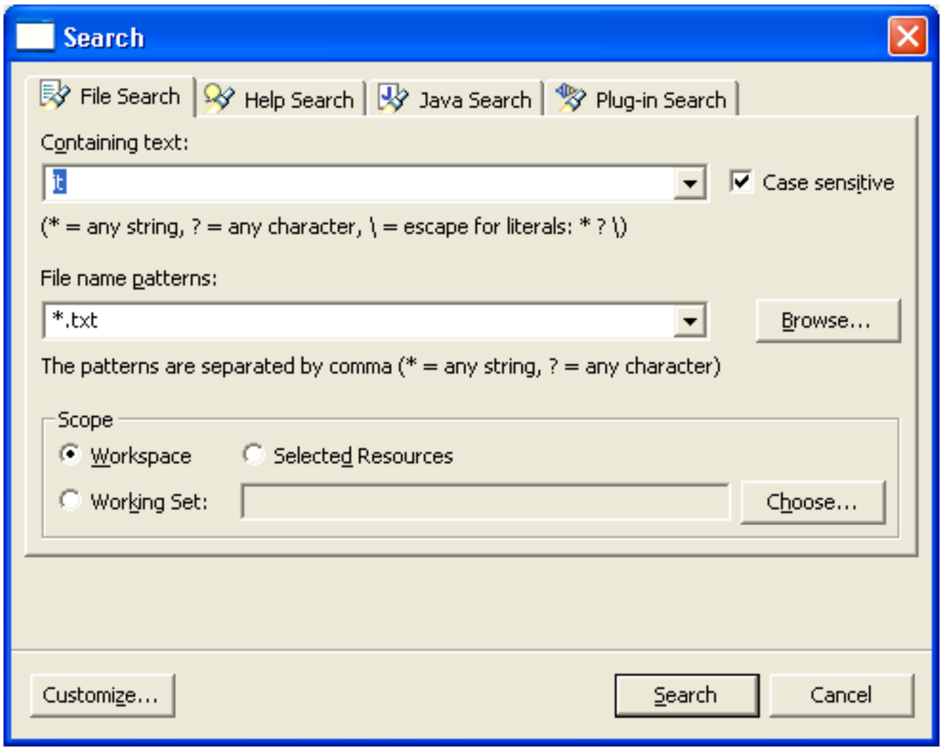
Searching

You can search for text strings and files in the Workbench. In this section we will use the Search button to perform a text search across the resources that are shown in our Navigator. We will then learn how to use the Search view to work with the results.

Starting a search

You can search for text strings in the Workbench as follows:

1. In the Workbench toolbar, click the search button .
2. In the **Containing text** field, type in: *it*
   The combo box for the **Containing text** field also lets you select from a list of recently performed searches.



Containing text: it

[✓] Case sensitive

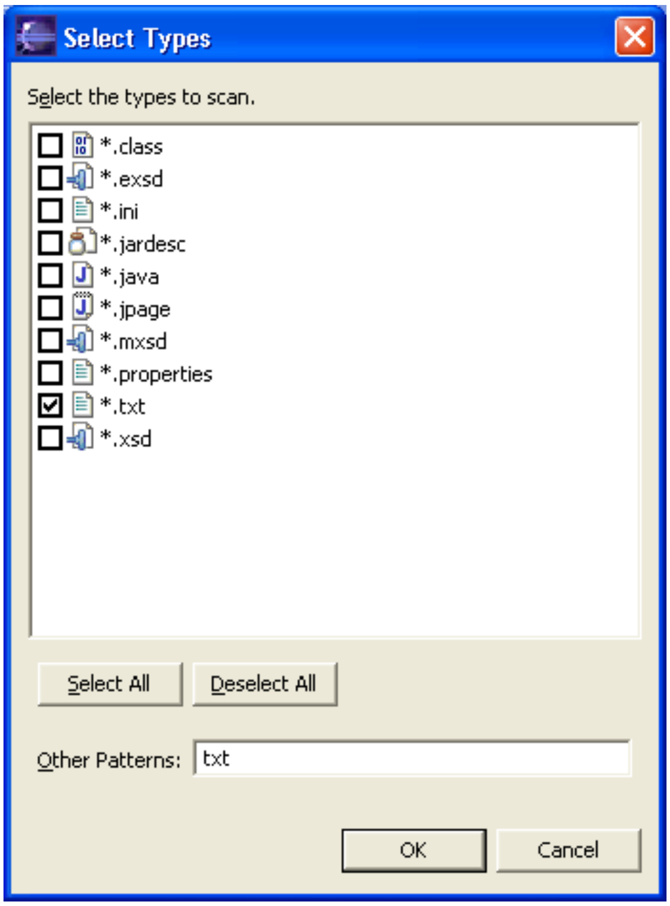
(* = any string, ? = any character, \ = escape for literals: * ? \)

File name patterns: *.txt

The patterns are separated by comma (* = any string, ? = any character)

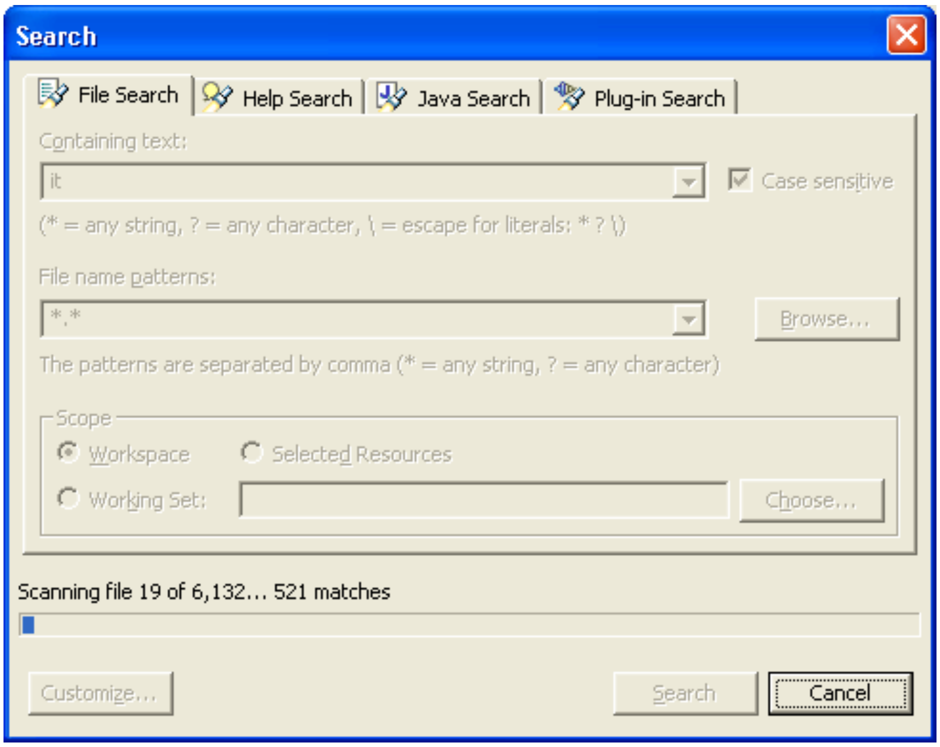
Scope: [● Workspace] ○ Selected Resources ○ Working Set:

3. You can select or deselect the **Case sensitive** check box depending on whether or not you want to perform a case sensitive or insensitive search.
   We want to search for lowercase "*it*" so check the **Case sensitive** box.
4. In the **File name patterns** field you can specify the kinds of files to include in your search. Click **Browse...** to open the Select Types dialog. This dialog provides you with a quick way to select from a list of registered extensions.



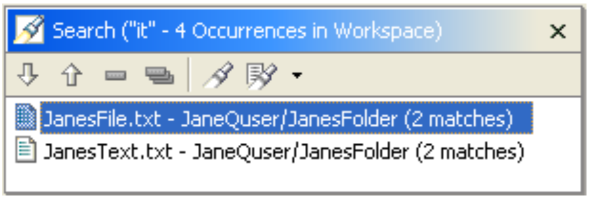
For the moment we will confine our search to .txt files. Ensure *.txt is checked and click **OK**.

5. In the **Scope** field you can specify the files and folders to include in the search. The choices are: the entire workspace, the currently selected resources in the Workbench, or a working set which is a named, customized group of files and folders. Leave the scope as workspace.
6. The **Customize** button allows you to choose what kinds of searches are available in the dialog. This setting may be left unchanged.
7. Click **Search**.
   Note that you can click **Cancel** to cancel the search while it is in progress and that the Search dialog shows the progress of the search.



Scanning file 19 of 6,132... 521 matches

8. Observe that the Search view displays:



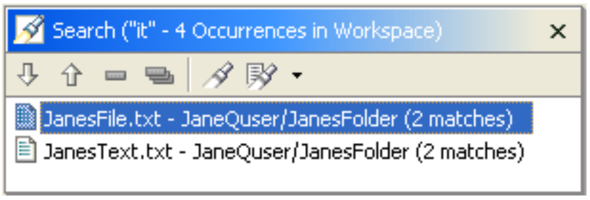
Search ("it" - 4 Occurrences in Workspace)

JanesFile.txt - JaneQuser/JanesFolder (2 matches)
JanesText.txt - JaneQuser/JanesFolder (2 matches)

In the next section we will learn how to work with the Search view.

The Search view

Now that we have completed our search for "it" the Search view is visible. The title of the Search view shows that four matches were found.

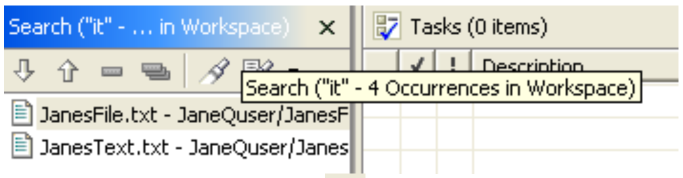


Search ("it" - 4 Occurrences in Workspace)

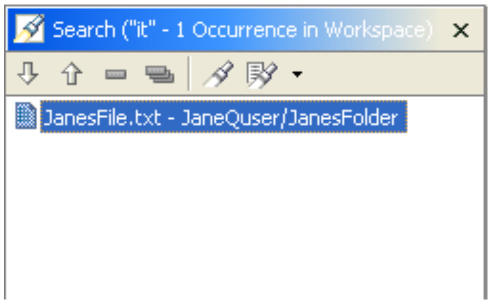
JanesFile.txt - JaneQuser/JanesFolder (2 matches)
JanesText.txt - JaneQuser/JanesFolder (2 matches)

Within the Search view two files are shown and we can further see that within each file there were 2 matches found. Let's take a more detailed look at the Search view.

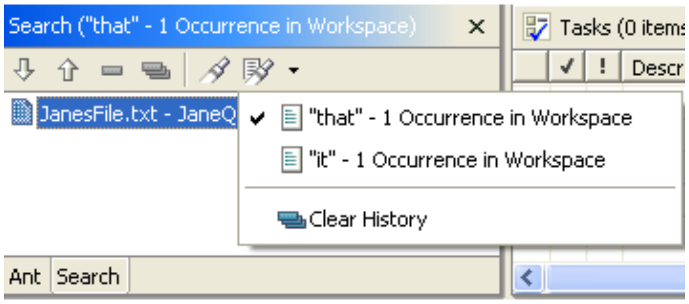
1. The title of the Search view provides us with a description of the search we just performed. If the view is too narrow to display the full description, hover over the title of the Search view. Observe that hover help appears and provides the full description.


Search ("it" - ... in Workspace)
Tasks (0 items)
Description
Search ("it" - 4 Occurrences in Workspace)
JanesFile.txt - JaneQuser/JanesF
JanesText.txt - JaneQuser/Janes

2. Click the Show Next Match button ⇩ to navigate to the first match of the search expression ("it"). Notice that the file JanesFile.txt is automatically selected and opened in the editor area.
   Click Show Next Match button two more times. Once again the Search view automatically opens the file (JanesText.txt).
3. It is sometimes useful to remove uninteresting matches from the search results. The Search view's popup menu provides two ways to remove matches:
   - **Remove Current Match** − Removes the actual match that you are currently looking at. If a file contains multiple matches, as in our case, only the current match is removed.
   - **Remove Selected Matches** − Removes the file entry and all matches in it.
4. Choose **Remove Current Match** from the popup. Notice how JanesText.txt only has one match left.
5. Select JanesFile.txt in the Search view.
6. Choose **Remove Selected Matches** from the popup menu.
   The Search view now shows only one match for JanesText.txt


Search ("it" - 1 Occurrence in Workspace)
JanesFile.txt - JaneQuser/JanesFolder

7. Perform a second search for "that" by clicking on the Search button in the Workbench's toolbar.
8. The Search view updates to show the results of the new search.
   You can move back and forth between your two search results using the drop down button on the Search view's toolbar.


Search ("that" - 1 Occurrence in Workspace)
Tasks (0 items
Descr
JanesFile.txt - JaneQ
"that" - 1 Occurrence in Workspace
"it" - 1 Occurrence in Workspace
Clear History
Ant
Search

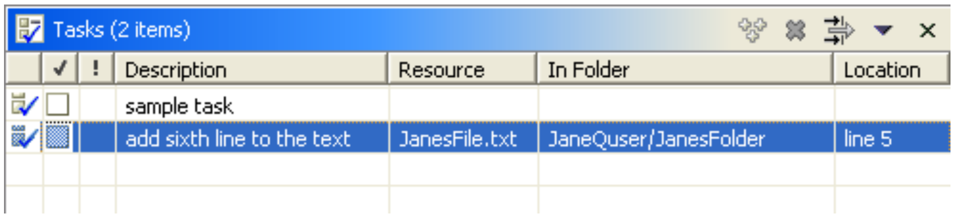
9. In the drop down button choose "it" − 1 Occurrence. The Search view switches back to show you the original search. On the context menu choose **Search Again** to repeat the initial search. Notice that once again we have four matches.

Tasks and markers

There are various types of markers including bookmarks, task markers, debugging breakpoints and problems. In this section we will focus on tasks and the Tasks view.

The Tasks view displays all the tasks and problems in the Workbench. It allows us to view those associated with specific files, specific lines in specific files, as well as generic tasks that are not associated with any specific file.

In the figure below "sample task" is a generic task not associated with any specific resource. The second task ("add sixth line to the text") is associated with the file JanesFile.txt.



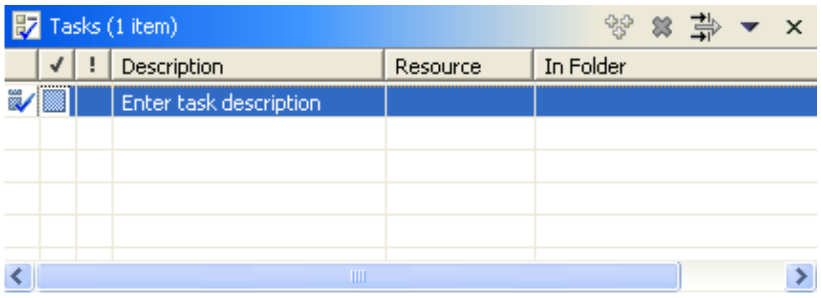
Tasks (2 items)

	✓	!	Description	Resource	In Folder	Location
	☐		sample task			
	▨		add sixth line to the text	JanesFile.txt	JaneQuser/JanesFolder	line 5

Unassociated tasks

Unassociated tasks are not associated with any specific resource. To create an unassociated task:

1. In the Tasks view, click the New Task button . A new task dialog appears.
2. Type a brief description for the task and press Enter. If you change your mind while entering the description you can press Escape to cancel the dialog. The new task appears in the Tasks view.



Tasks (1 item)

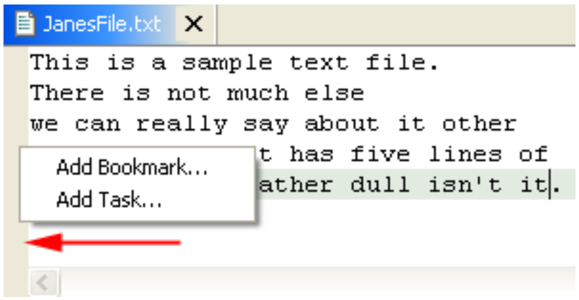
	✓	!	Description	Resource	In Folder
	▨		Enter task description		

Associated tasks

Associated tasks are associated with a specific location in a resource. To associate a task with our JanesFile.txt:

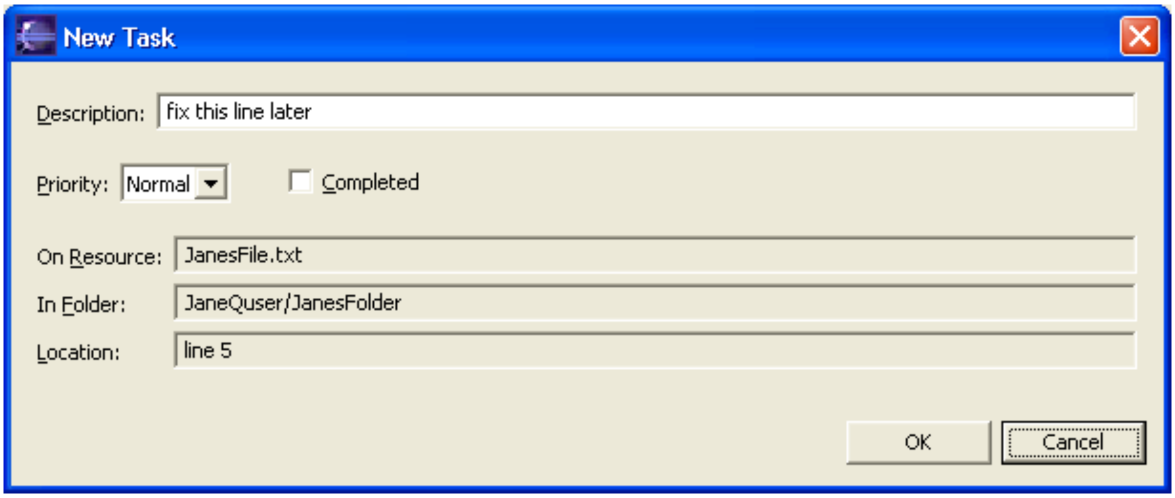
1. Open a text file (JanesFile.txt) by double clicking on it in the Navigator view.
2. In the editor area directly to the left of any line in the text editor, access the context menu from the marker bar. The marker bar is the vertical bar to the left of the main text area.
3. From the marker bar's context menu, select **Add Task**.

   The marker bar displays any marker including bookmarks, task markers (for associated tasks), and/or debugging breakpoints. You can associate various markers with specific lines in a file by accessing the context menu from the marker bar directly to the left of that line.



```
This is a sample text file.
There is not much else
we can really say about it other
t has five lines of
ather dull isn't it.
```

4. In the **Description** field, type a brief description for the task you want to associate with that line in the text file.



Description: fix this line later

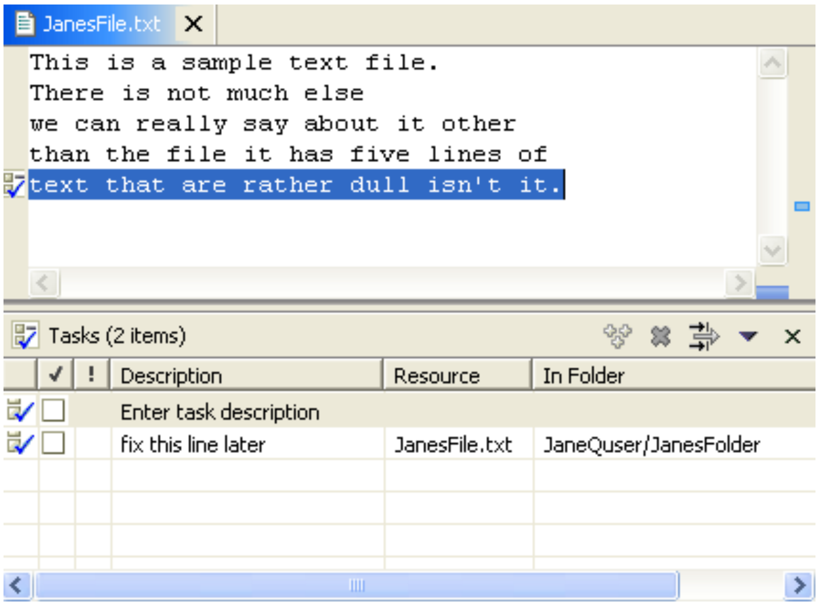
Priority: Normal ☐ Completed

On Resource: JanesFile.txt

In Folder: JaneQuser/JanesFolder

Location: line 5

5. Click **OK** when you are done.
6. Notice that a new task marker appears in the marker bar, directly to the left of the line where you added the task. Also, notice that the new task appears in the Tasks view.



```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it.
```

Tasks (2 items)

	✓	!	Description	Resource	In Folder
✓	☐		Enter task description		
✓	☐		fix this line later	JanesFile.txt	JaneQuser/JanesFolder

7. After you have added the new task, click in the editor on the first line or any other line above the line with which the new task is associated.
8. Add several lines of text to the file at this point.
9. Notice that as you add lines of text above it, the task marker moves down in the marker bar in order to remain with the associated line in the file. The line number in the Tasks view is updated when the file

is saved.
10. In the Tasks view, access the context menu for the task you just created.
11. Select **Mark Completed**.
12. Now select **Delete Completed Tasks** from the marker's context menu.
13. Notice that the task marker disappears from the marker bar and the task is removed from the Tasks view.

Opening files

The Tasks view provides two approaches for opening the file associated with a task:

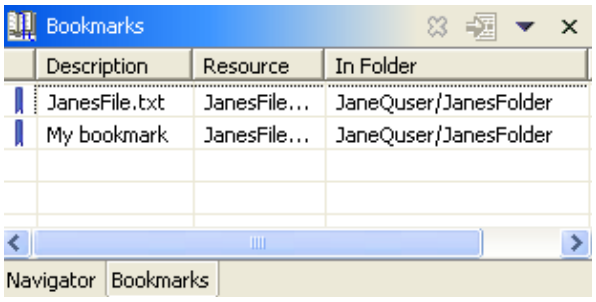
- Select the task, and from the context menu choose **Go To**
- Double click on the task

In both cases the file's editor is opened and the line with which the selected task is associated is highlighted.

Bookmarks

Bookmarks are a simple way to navigate to resources that you frequently use. In this section we will look at setting and removing bookmarks and viewing them in the Bookmarks view.

The Bookmarks view displays all bookmarks in the Workbench. To show the Bookmarks view choose **Window** > **Show View** > **Bookmarks** while in the Resource perspective.

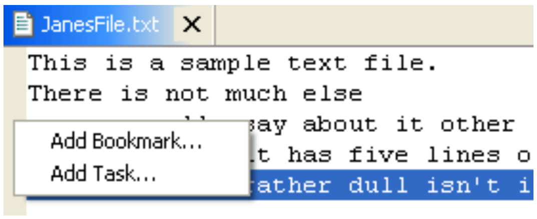


Description	Resource	In Folder
JanesFile.txt	JanesFile...	JaneQuser/JanesFolder
My bookmark	JanesFile...	JaneQuser/JanesFolder

Adding and viewing bookmarks

The Workbench allows you to bookmark individual files or locations within a file. In this section we will set several bookmarks and view them using the Bookmarks view.

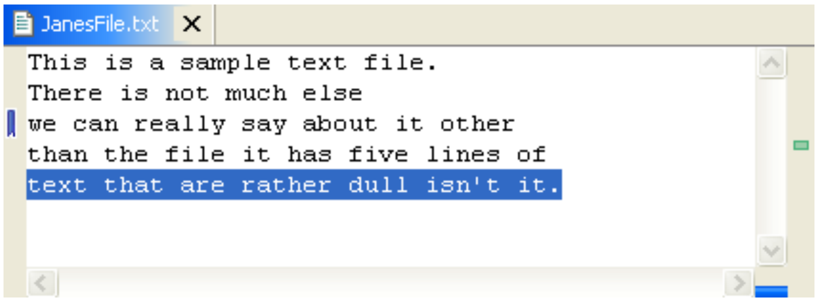
1. From the menubar, select **Window** > **Show View** > **Bookmarks**. The Bookmarks view appears in the Workbench.
2. Edit the file JanesFile.txt
3. Position your cursor over the editor's marker bar next to any line in the file. Then, from the context menu on the marker bar, select **Add Bookmark**.



```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it.
```

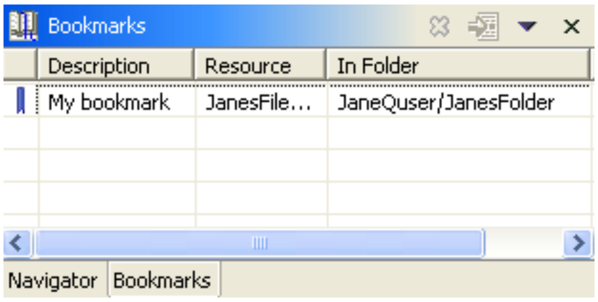
When the Add Bookmark dialog opens type in a description for this bookmark. Type in "My Bookmark".

4. Notice that a new bookmark appears in the marker bar.



```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it.
```

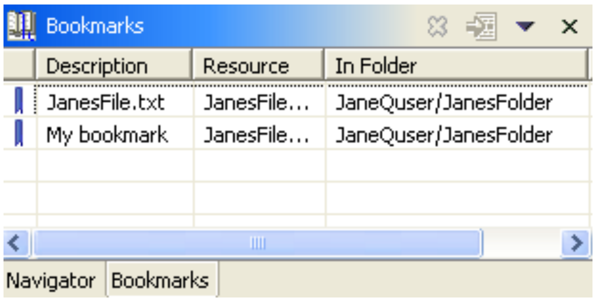
The new bookmark also appears in the Bookmarks view.



Description	Resource	In Folder
My bookmark	JanesFile...	JaneQuser/JanesFolder

5. In the Navigator view select the file JanesText.txt. From the file's popup menu choose **Add Bookmark**

   This will bookmark the file using the filename to describe the bookmark. Observe the Bookmarks view now contains two bookmarks.



Description	Resource	In Folder
JanesFile.txt	JanesFile...	JaneQuser/JanesFolder
My bookmark	JanesFile...	JaneQuser/JanesFolder

Using bookmarks

Now that we have our bookmarks we will see how to get to the files associated with the bookmarks.

1. Close all of the files in the editor area.
2. In the Bookmarks view, double−click on the first bookmark we created (My Bookmark).
3. Notice that a file editor opens displaying the file with which the bookmark is associated and that the line associated with the bookmark is highlighted.

The Bookmarks view supports two additional ways to open the file associated with a selected bookmark:

1. From the bookmark's context menu choose **Go To**
2. From the view's toolbar click on the Go To toolbar button

The Bookmarks view allows you to select the associated file in the Navigator.

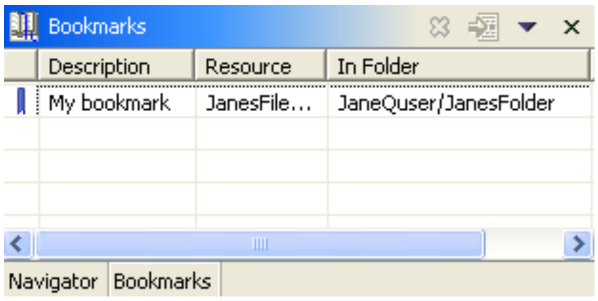
1. In the Bookmarks view, select My Bookmark.
2. From the bookmark's context menu choose **Show in Navigator**.
3. Notice that the Navigator view is now visible and the file JanesFile.txt is automatically selected for us. JanesFile.txt is the file My Bookmark was associated with.

Removing bookmarks

In this section we will learn how to remove the bookmarks we have created.

1. In the Bookmarks view, select JanesFile.txt (the second bookmark we created) and do one of the following:
   - Click the Delete button on the view's toolbar.
   - From the bookmark's context menu choose delete.
   - Hit the Delete key on your keyboard.

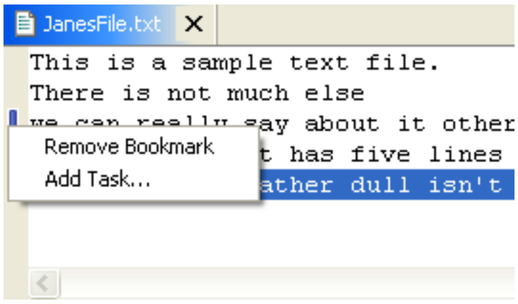
   Notice that the bookmark is removed from the Bookmarks view.


Bookmarks

Description	Resource	In Folder
My bookmark	JanesFile...	JaneQuser/JanesFolder

Navigator Bookmarks

2. We should have one remaining bookmark. This bookmark is associated with a line in the file JanesFile.txt. There are two other approaches to removing this bookmark.
   - Use **Remove Bookmark** in the marker bar of the JanesFile.txt editor. You may recall that we used **Add Bookmark** in the marker bar when we first created the bookmark.



```
This is a sample text file.
There is not much else
we can really say about it other
t has five lines
ather dull isn't
```

- Delete the bookmark (as we did above) by using **Delete** from the bookmark's popup menu in the Bookmarks view.

Let's go with the second approach.

3. Ensure there is an editor open on JanesFile.txt.

   Although we don't actually need to open the editor, doing so will allow us to watch the editor update as we delete the bookmark.
4. In the Bookmarks view, select JanesFile.txt (the remaining bookmark). Click the Delete button on the view's toolbar. Notice that the bookmark is removed from the Bookmarks view and the JanesFile.txt editor.

Rearranging views and editors

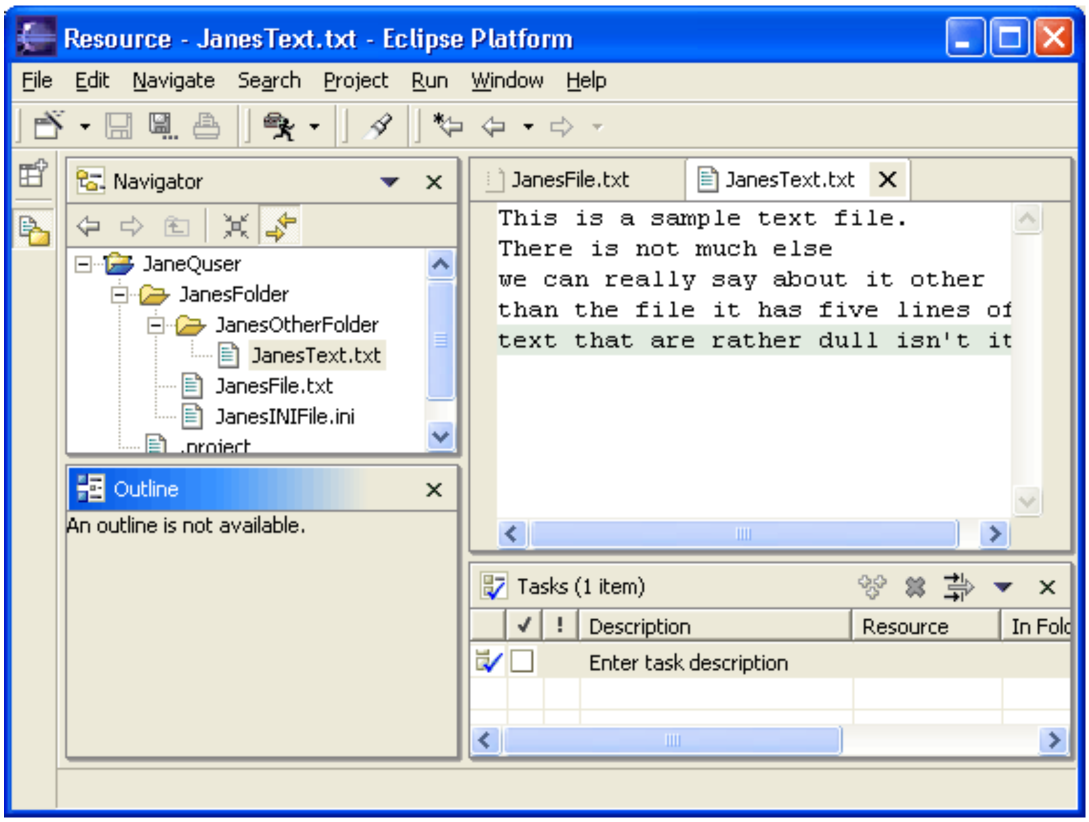
We have seen how to work with editors and views. In this section we will learn how to rearrange these to customize the layout of the Workbench.

Setup

Before we start rearranging our Workbench, we need to do a little housekeeping.

1. Start by choosing **Window** > **Reset Perspective** and selecting **OK**. This will reset the current perspective to its original views and layout.
2. Ensure there are editors open for JanesFile.txt and JanesText.txt. Close any other editors including the Welcome Page. If you need to, you can always get back to the welcome page by choosing **Help** > **Welcome...** and selecting **Eclipse Platform**.

Your Workbench should now look like this:



```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it
```

An outline is not available.

✓	!	Description	Resource	In Fold
☐		Enter task description		

Drop cursors

Drop cursors indicate where you can dock views in the Workbench window. Several different drop cursors may be displayed when you are rearranging views.

⬆	Dock above: if you release the mouse button when a dock above cursor is displayed, the view will appear above the view underneath the cursor.
⬇	Dock below: if you release the mouse button when a dock below cursor is displayed, the view will appear below the view underneath the cursor.
➡	Dock to the right: If you release the mouse button when a dock to the right cursor is displayed, the view will appear to the right of the view underneath the cursor.
⬅	Dock to the left: if you release the mouse button when a dock to the left cursor is displayed, the view will appear to the left of the view underneath the cursor.
▭	Stack: if you release the mouse button when a stack cursor is displayed, the view will appear as a tab in the same pane as the view underneath the cursor.
⊘	Restricted: if you release the mouse button when a restricted cursor is displayed, the view will not dock there. For example, you cannot dock a view in the editor area.

Rearranging views

We can change the position of our Navigator view in the Workbench window.

1. Click in the title bar of the Navigator view and drag the view across the Workbench window. Do not release the mouse button yet.
2. While still dragging the view around on top of the Workbench window, note that various drop cursors appear. These drop cursors (see previous section) indicate where the view will dock in relation to the view or editor area underneath the cursor when you release your mouse button.
3. Dock the view in any position in the Workbench window, and view the results of this action.
4. Click and drag the view's title bar to re−dock the view in another position in the Workbench window. Observe the results of this action.
5. Finally, drag the Navigator view over the Outline view. You will see a stack cursor. If you release the mouse button the Navigator will be stacked with the Outline view into a tabbed notebook.

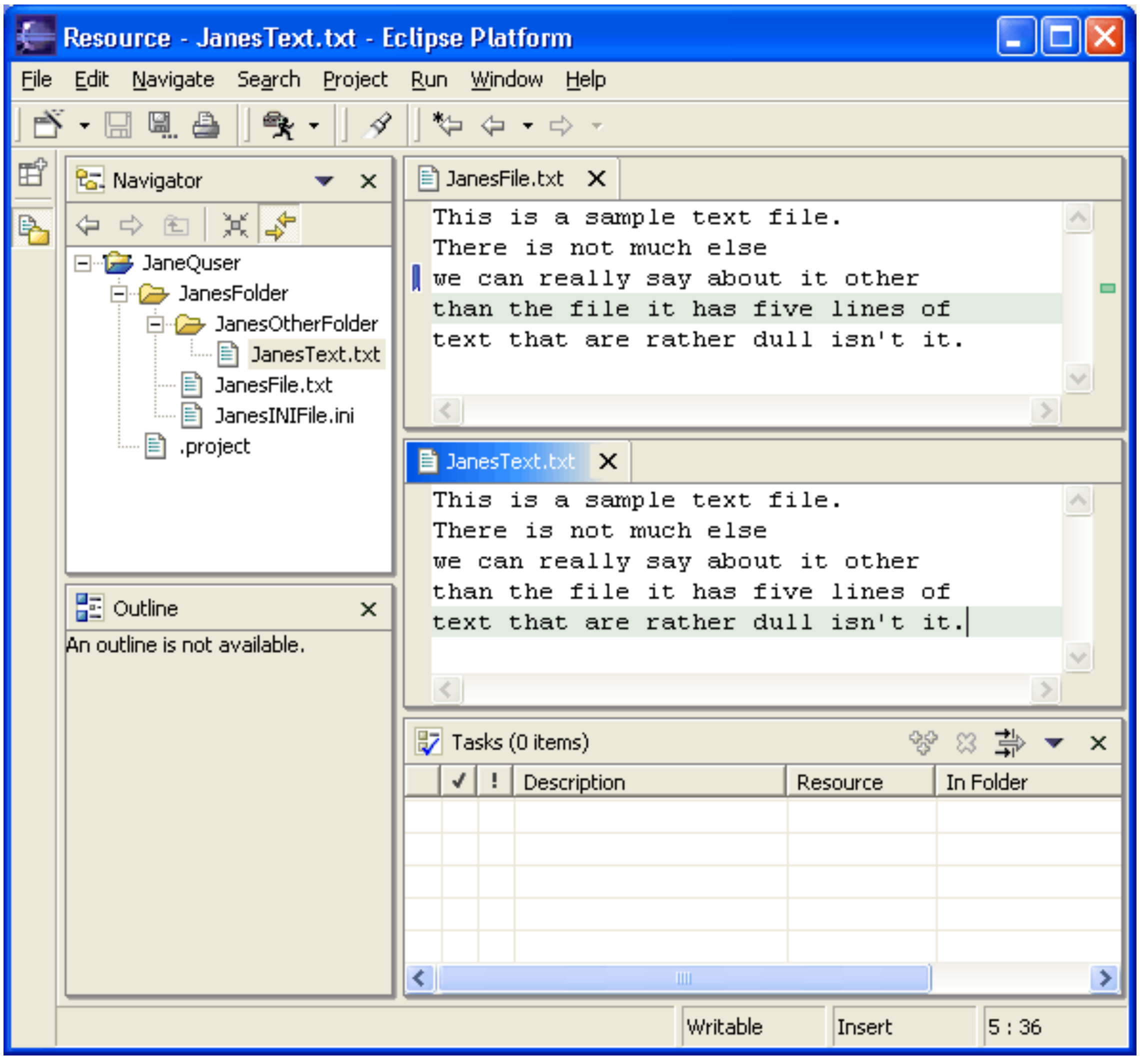
Tiling editors

The Workbench allows you to create two or more sets of editors in the editor area. You can also resize the editor area but you cannot drag views into the editor area.

1. Open at least two editors in the editor area by double−clicking editable files in the Navigator view.
2. Click and drag one of the editor's tabs out of the editor area. Do not release the mouse button.
3. Notice that the restricted cursor displays if you attempt to drop the editor either on top of any view or outside the Workbench window.
4. Still holding down the mouse button, drag the editor over the editor area and move the cursor along all four edges as well as in the middle of the editor area, on top of another open editor. Notice that along the edges of the editor area the directional arrow drop cursors appear, and in the middle of the editor area the stack drop cursor appears.
5. Dock the editor on a directional arrow drop cursor so that two editors appear in the editor area.
6. Notice that you can also resize each editor as well as the entire editor area to accommodate the editors and views as necessary.
7. It is important to observe the color of an editor tab (in the figure below there are two groups, one above the other)

   blue − indicates that the editor is currently active

   default (gray on Windows XP) − indicates that the editor was the last active editor. If there is an active view, it will be the editor that the active view is currently working with. This is important when working with views like the Outline and Properties that work closely with the editor.
8. Drag and dock the editor somewhere else in the editor area, noting the behavior that results from docking on each kind of drop cursor. Continue to experiment with docking and resizing editors and views until you have arranged the Workbench to your satisfaction. The figure below illustrates the layout if you drag and drop one editor below another.



JanesFile.txt

```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it.
```

JanesText.txt

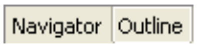
```
This is a sample text file.
There is not much else
we can really say about it other
than the file it has five lines of
text that are rather dull isn't it.
```

An outline is not available.

Rearranging tabbed views

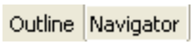
In addition to dragging and dropping views on the Workbench you can also rearrange the order of views within a tabbed notebook.

1. Choose **Window > Reset Perspective** to reset the Resource perspective back to its original layout.
2. Click on the Outline title bar and drag it on top of the Navigator view. The Outline will now be stacked on top of the Navigator.
3. Click on the Navigator tab and drag it to the right of the Outline tab.



4. Once your cursor is to the right of the Outline tab and the cursor is a stack cursor release the mouse button.

   Observe the Navigator tab is now to the right of the Outline tab.



Maximizing

Sometimes it is useful to be able to maximize a view or editor. Maximizing both views and editors is easy.

- To maximize a view you can either double click on its title bar or choose **Maximize** from the popup menu available from the top left corner of the view.
- To maximize an editor double click on the editor tab or choose **Maximize** from the popup menu available from the editor tab.

Restoring a view to its original size is done in a similar manner (double click or choose **Restore** from the menu).

Fast views

Fast views are hidden views, which can be quickly made visible. They work the same as normal views, only when they are hidden they do not take up screen space on your Workbench window.

In this section we will learn how to convert the Navigator view into a fast view.

Creating fast views

Fast views are hidden views, which can be quickly made visible. We will start by creating a fast view from the Navigator view and then dig into how to use the view once it is a fast view.

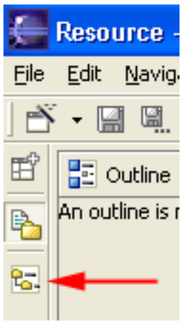
There are two ways to create a fast view

- Using drag and drop
- Using a menu operation available from the view System menu.

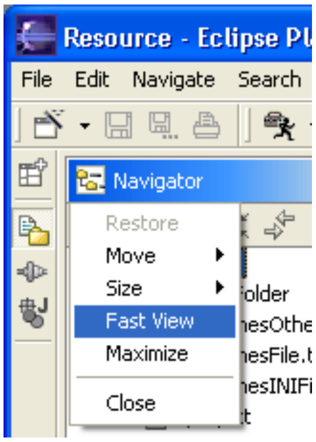
Let's start by creating a fast view using drag and drop.

1. In the Navigator view click on the title bar and drag it to the shortcut bar at the far left of the window.
2. Once you are over the shortcut bar the cursor will change to a stack cursor. Release the mouse button to drop the Navigator onto the shortcut bar.

   The shortcut bar now includes a button for the Navigator fast view



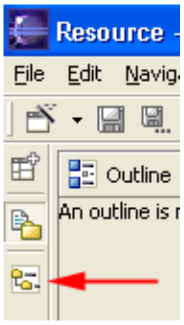
To create a fast view using the second approach we would started by popping up the context menu over icon in the top left corner of the Navigator view. From this menu we would have chosen **Fast View**.



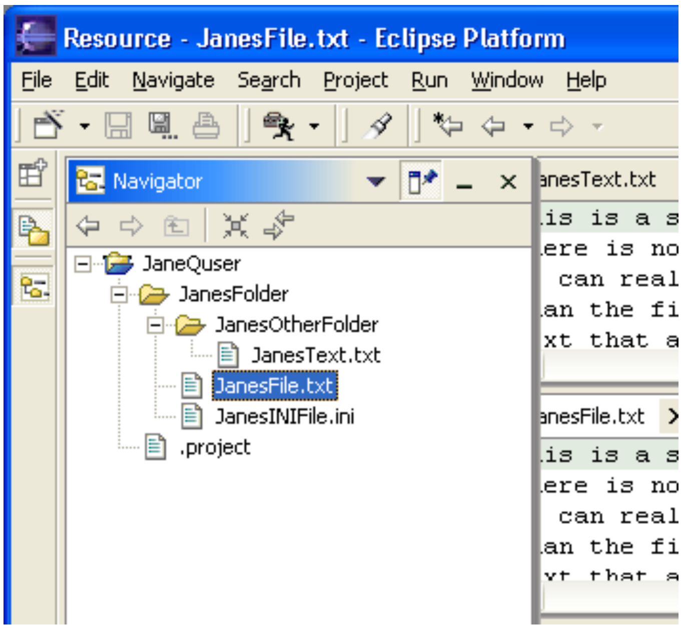
Working with fast views

Our navigator has been converted into a fast view. The next question is, "What can we do with it?". The answer to this question and others are about to be revealed.

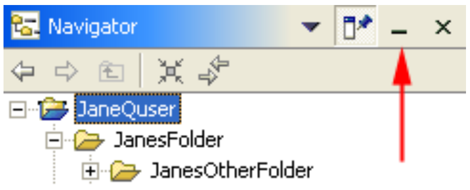
Confirm that your shortcut bar at the far left of the window still has the Navigator view and looks like this:



1. In the shortcut bar click on the Navigator fast view button.
2. Observe the Navigator view slides out from the shortcut bar.



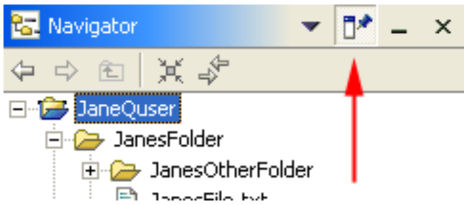
3. You can use the Navigator fast view as you would normally. To resize a fast view, move the mouse to the right edge of the fast view where the cursor will change to a double−headed arrow. Then hold the left mouse button down as you move the mouse.
4. To hide the fast view simply click on another view or editor or click on the **Minimize** button on the fast view's toolbar



Note: If you open a file from the Navigator fast view, the fast view will automatically hide itself to allow you to work with the file.

To convert a fast view back to a regular view you can either:

- Click the Fast View button on the fast view's toolbar



- Choose **Fast View** from the context menu of the icon in the top left corner of the view.
- Drag the fast view icon from the tool bar, and drop it somewhere in the Workbench window.

Perspectives

A perspective defines the initial set and layout of views in the Workbench window. One or more perspectives can exist in a single Workbench window.

Perspectives can be opened in one of two ways:

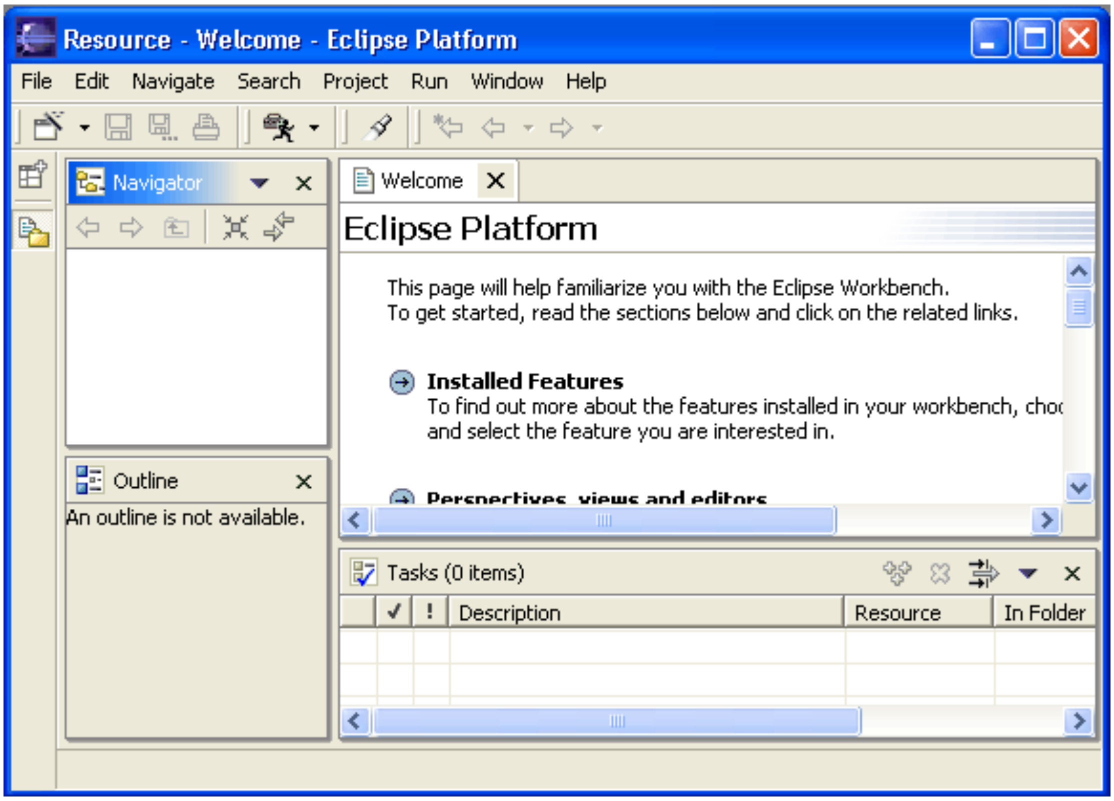
- In the same (existing) Workbench window
- In a new Workbench window

Perspectives define visible action sets, which you can change to customize a perspective. You can save a perspective that you build in this manner, making your own custom perspective that you can open again later.

The Workbench window displays one or more perspectives. Initially one perspective, the Resource perspective, is displayed. A perspective consists of views like the Navigator as well as editors for working with resources. More than one Workbench window can be open at any given time.

So far we have only used the Resource perspective (shown below). In this section we will explore how to open and work with other perspectives.

A perspective provides us with a set of capabilities aimed at accomplishing a specific type of task, or working with a specific type of resource.


Resource - Welcome - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
Welcome
Tasks (0 items)
Description
Resource
In Folder

Eclipse Platform

This page will help familiarize you with the Eclipse Workbench.
To get started, read the sections below and click on the related links.

Installed Features
To find out more about the features installed in your workbench, choo
and select the feature you are interested in.

Perspectives, views and editors

An outline is not available.

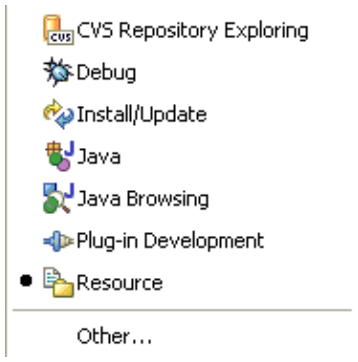
New perspectives

There are several ways to open a new perspective within this Workbench window:

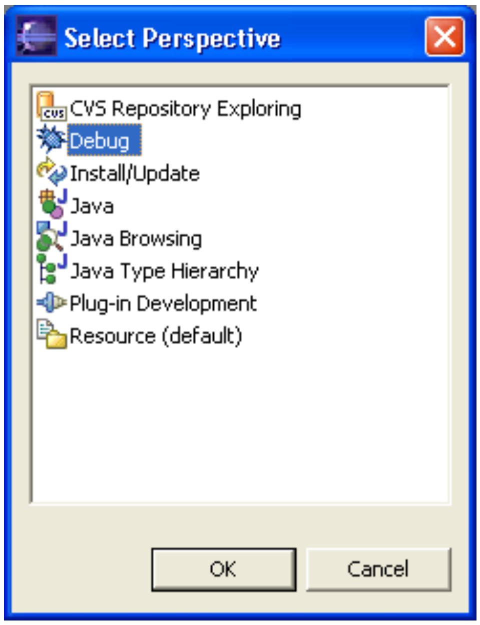
- Using the Open Perspective button on the shortcut bar.
- Choosing a perspective from the **Window** > **Open Perspective** menu.

We will open one by using the shortcut bar button.

1. Click on the Open Perspective button .
2. A menu appears showing the same choices as shown on the **Window** > **Open Perspective** menu. Choose **Other** from the menu.



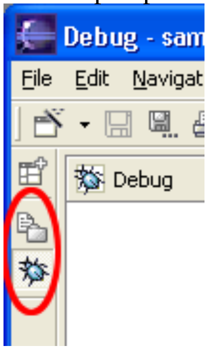
3. In the Select Perspective dialog choose **Debug** and click **OK**.



   The Debug perspective is displayed.
4. There are several other interesting things to take note of.

   - ♦ The title of the window now indicates that we are using the Debug perspective.
   - ♦ The shortcut bar now contains two perspectives, the original Resource perspective and the new Debug perspective. The Debug perspective button is pressed in, indicating that it is the

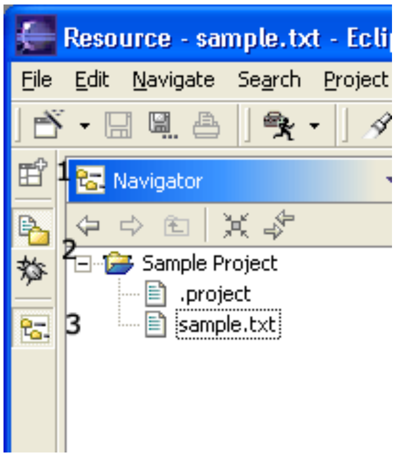
current perspective.



5. In the shortcut bar, click on the Resource perspective button. The Resource perspective is once again our current perspective. Notice that the set of views is different for each of the perspectives.

Earlier in this tutorial we created a Navigator fast view by dragging the Navigator view onto the shortcut bar. If we do this again we will observe that our shortcut bar now has three areas:

1. The Open Perspective button.
2. The list of open perspectives. In the figure below the Resource and Debug perspectives are open and the Resource perspective is our current perspective.
3. The list of fast views. In the figure below there is one fast view, the Navigator.

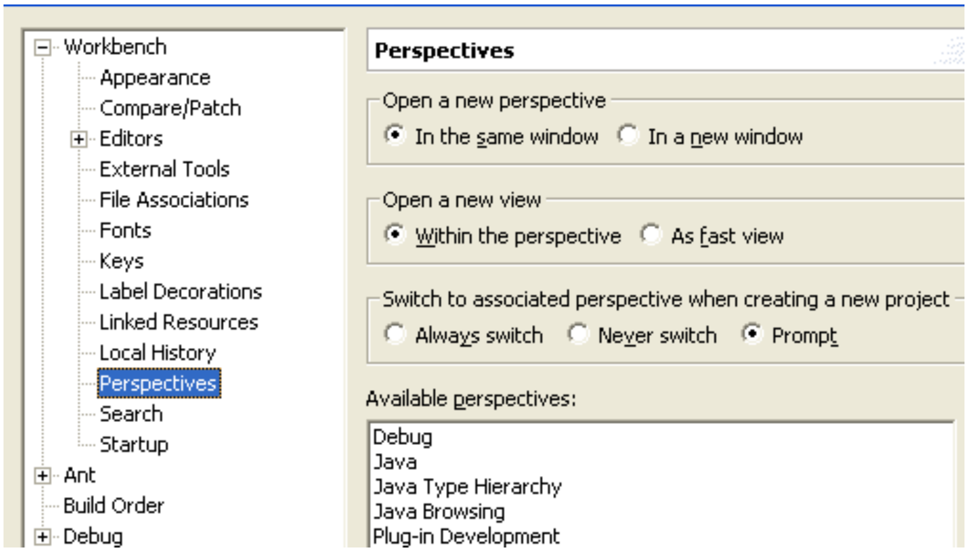

Resource - sample.txt - Ecli
File Edit Navigate Search Project
Navigator
1
2
3
Sample Project
.project
sample.txt

In step 5 above we observed that the set of views are different between the open perspectives. Similarly each perspective has its own list of fast views.

New windows

We have just seen how to open a perspective inside the current Workbench window. We can also open new perspectives in their own window.

By default, new perspectives are opened in the current window. You can configure this default behavior using **Window** > **Preferences** > **Workbench** > **Perspectives**.



Perspectives

Open a new perspective
[✓ In the same window] In a new window

Open a new view
[✓ Within the perspective] As fast view

Switch to associated perspective when creating a new project
Always switch Never switch [✓ Prompt]

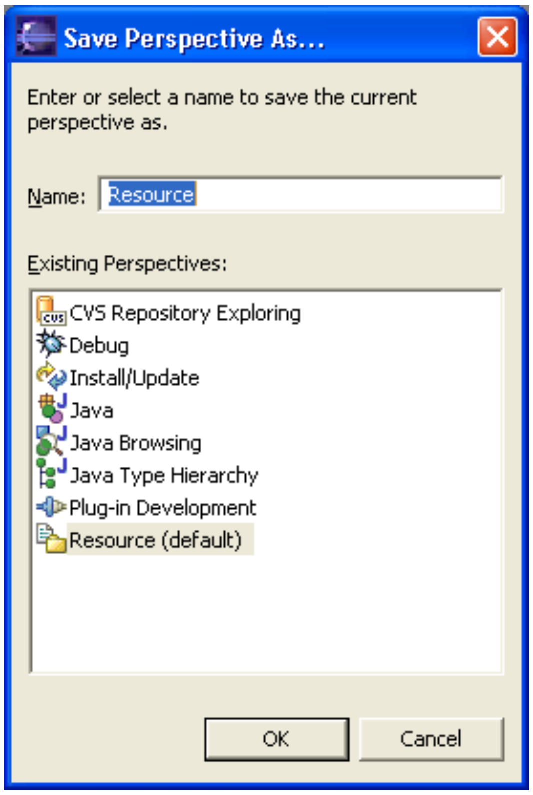
Available perspectives:
Debug
Java
Java Type Hierarchy
Java Browsing
Plug-in Development

Saving perspectives

In this tutorial we have seen how to add new views to our perspective, rearrange the views and convert views into fast views. The Workbench also allows you to save this layout for future use.

1. In the shortcut bar click on the Resource perspective. The Resource perspective is now active.
2. Drag the Outline view and stack it with the Navigator view.
3. Choose **Window > Save Perspective As...**
4. The Save Perspective As dialog allows you to redefine and existing perspective or create a new perspective.

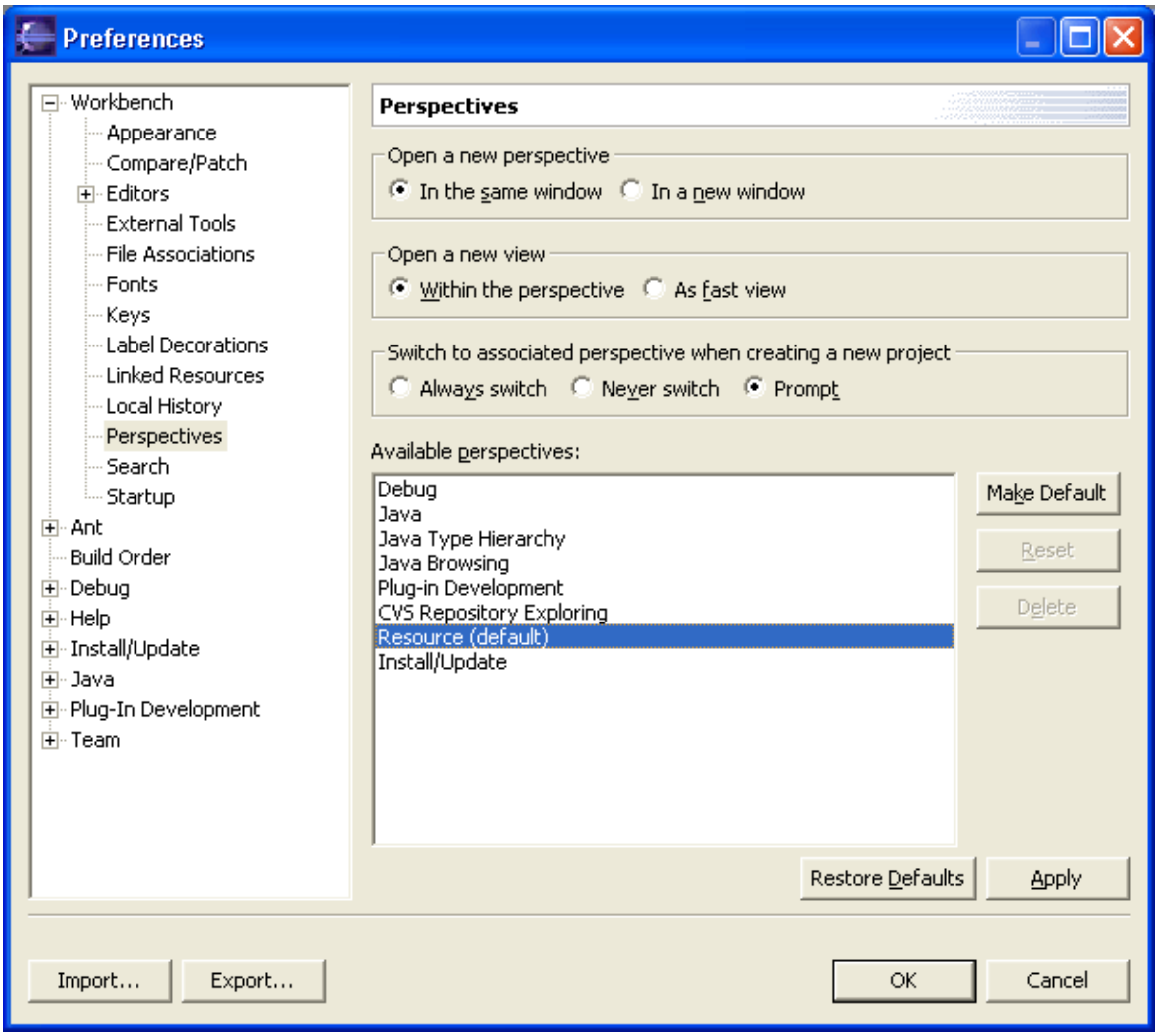
   Click **OK** to update the Resource perspective and **Yes** to the subsequent confirmation dialog. The new perspective layout will be used if you reset the perspective or open a new one.



5. In the Resource perspective move the Outline view so that it is now stacked with the Tasks view.
6. Choose **Window** > **Reset Perspective**. Notice the Outline view is stacked with the Navigator. Originally when we first started the Workbench it was below the Navigator, but because we saved the perspective with Repositories and Outline stacked, it now considers this its initial layout.
7. Choose **Window** > **New Window** to open a second window showing the resource perspective. Observe that it uses the newly saved layout.
8. Close the second window.

Now that we have forever changed our Resource perspective you're probably hoping there is some way to get back the original layout. To reset the Resource perspective to its original layout:

1. Choose **Window** > **Preferences**.
2. Expand **Workbench** and choose **Perspectives**.
3. Select the **Resource** perspective and click the **Reset** button and then Click **OK**.


Preferences

- Workbench
  - Appearance
  - Compare/Patch
  - Editors
  - External Tools
  - File Associations
  - Fonts
  - Keys
  - Label Decorations
  - Linked Resources
  - Local History
  - Perspectives
  - Search
  - Startup
- Ant
- Build Order
- Debug
- Help
- Install/Update
- Java
- Plug-In Development
- Team

Perspectives

Open a new perspective: [✓ In the same window] In a new window

Open a new view: [✓ Within the perspective] As fast view

Switch to associated perspective when creating a new project: Always switch Never switch [✓ Prompt]

Available perspectives:

- Debug
- Java
- Java Type Hierarchy
- Java Browsing
- Plug-in Development
- CVS Repository Exploring
- Resource (default)
- Install/Update

4. We have undone any changes to the saved state of the perspective. If we also want to update the current copy of the Resource perspective that we are working with we must also choose **Window** > **Reset Perspective** from the Workbench's menubar.

Configuring perspectives

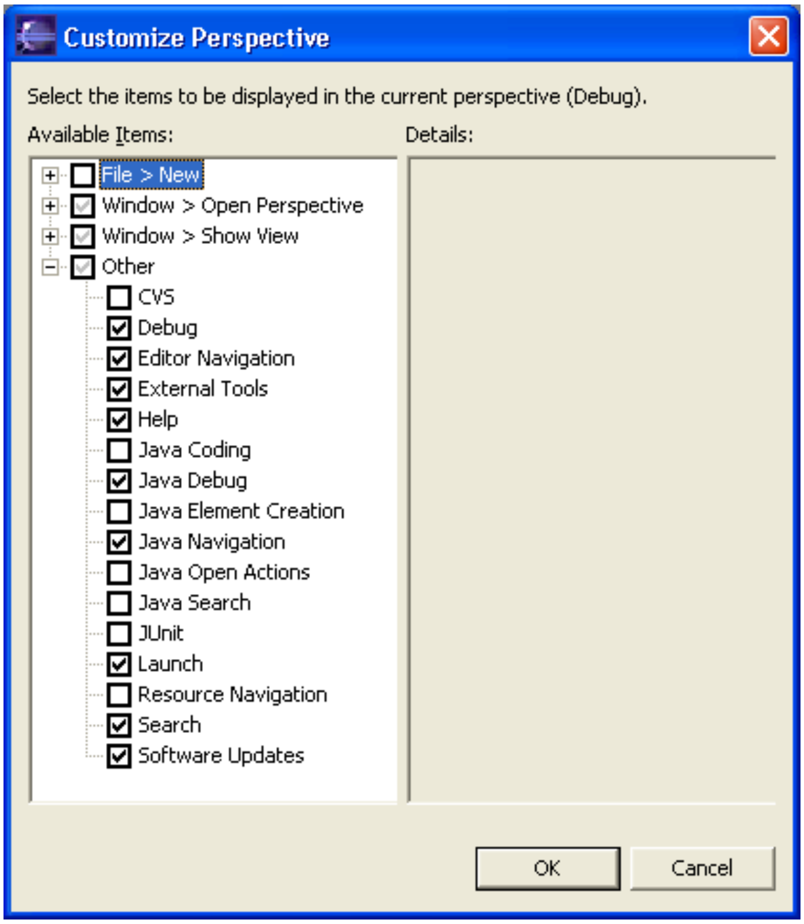
In addition to configuring the layout of your perspective you can also control several other key aspects of a perspective. These include:

- the **New** menu
- the **Window** > **Open Perspective** menu
- the **Window** > **Show View** menu
- action sets that show up on the toolbar

Let's try customizing one of these items.

1. In the shortcut bar click on the Resource perspective.
2. Select **Window** > **Customize Perspective...**
3. Expand **Other**.

4. Check **Launch** and click **OK**.



Customize Perspective

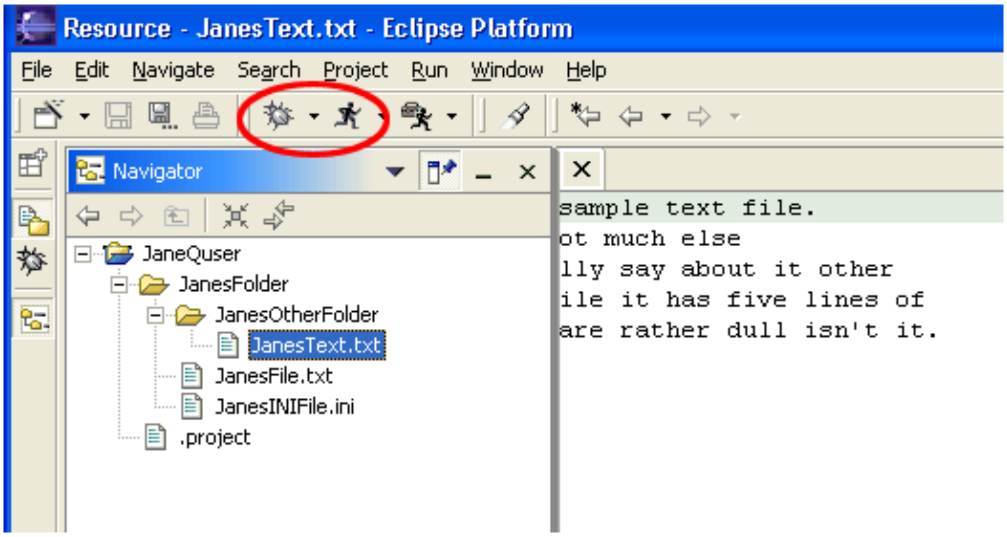
Select the items to be displayed in the current perspective (Debug).

Available Items:

- [ ] File > New
- [✓] Window > Open Perspective
- [✓] Window > Show View
- [✓] Other
  - [ ] CVS
  - [✓] Debug
  - [✓] Editor Navigation
  - [✓] External Tools
  - [✓] Help
  - [ ] Java Coding
  - [✓] Java Debug
  - [ ] Java Element Creation
  - [✓] Java Navigation
  - [ ] Java Open Actions
  - [ ] Java Search
  - [ ] JUnit
  - [✓] Launch
  - [ ] Resource Navigation
  - [✓] Search
  - [✓] Software Updates

Details:

5. Observe that the toolbar now includes buttons for debug/run launching.



Resource - JanesText.txt - Eclipse Platform

- JaneQuser
  - JanesFolder
    - JanesOtherFolder
      - JanesText.txt
    - JanesFile.txt
    - JanesINIFile.ini
  - .project

```
sample text file.
ot much else
lly say about it other
ile it has five lines of
are rather dull isn't it.
```

6. When you have finished experimenting with the other options on the Customize Perspective dialog, choose **Window** > **Reset Perspective** to return the perspective to its original state.

Comparing

The Workbench allows us to compare multiple resources and present the results in a special compare editor.

Setup

Before we get started with compare we need to create a few files. This will also be a good time to recap some of the basic features we have already learned about.

1. Start by selecting all of the projects in the Navigator and deleting them by using **Delete** on the pop−up menu.
2. Create a new simple project using **File** > **New** > **Project**. Be sure to give the project a distinct name by typing your name as the name of your new project (e.g., "JaneQUserCompare"). Do not use spaces or special characters in the project name.
3. Use the project's pop−up menu to create a file called file1.txt.

   In the editor for file1.txt type the following lines of text and save the file:

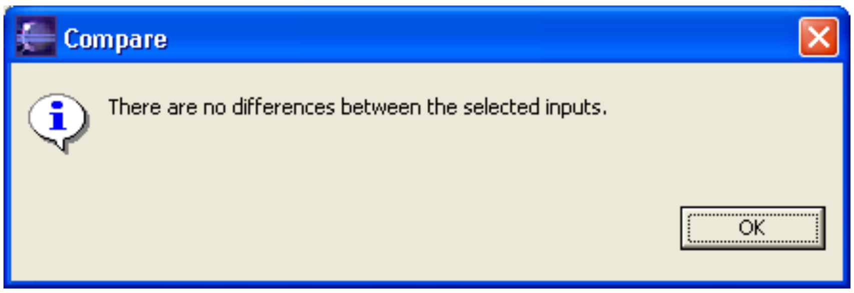
   This is line 1.
   This is line 2.
   This is line 3.
   This is line 4.
   This is line 5.
4. In the Navigator select file1.txt and use Ctrl+C to copy the file.
5. Use Ctrl+V (Paste) to create the copy. In the name conflict dialog which appears, rename the file to file2.txt.

We now have two files, file1.txt and file2.txt that are identical. Let's have some fun with the compare feature.

Simple compare

Comparing files is easy. In the Navigator select file1.txt and file2.txt choose **Compare With** > **Each Other** from the context menu.

Well that wasn't very interesting. Right about now you are probably staring at a dialog like the one below, which tells us the two files are the same.


Compare

There are no differences between the selected inputs.

To make things a little more interesting edit file1.txt as follows:

- delete line 1 "*This is line 1.*"
- change line 3 to be "This is a much better line 3."
- insert a line 4a (before line 5) that reads "This is line 4a and it is new"

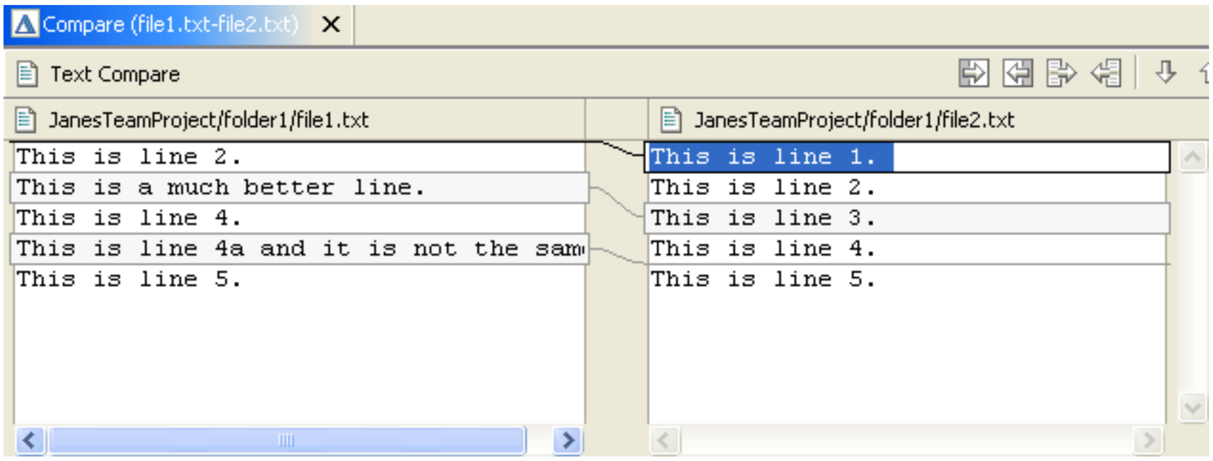
Your file (file1.txt) should now look like this:

This is line 2.
This is a much better line 3.
This is line 4.
This is line 4a and it is new
This is line 5.

Save the contents of the file by choosing **File** > **Save** (or pressing Ctrl+S).

Now we're really (honest we are) ready to do a compare. Once again select file1.txt and file2.txt and choose **Compare With** > **Each Other** in the Navigator's context menu.

A special compare editor opens. In the next section we will see how to use this compare editor.


Compare (file1.txt-file2.txt)
Text Compare
JanesTeamProject/folder1/file1.txt
JanesTeamProject/folder1/file2.txt

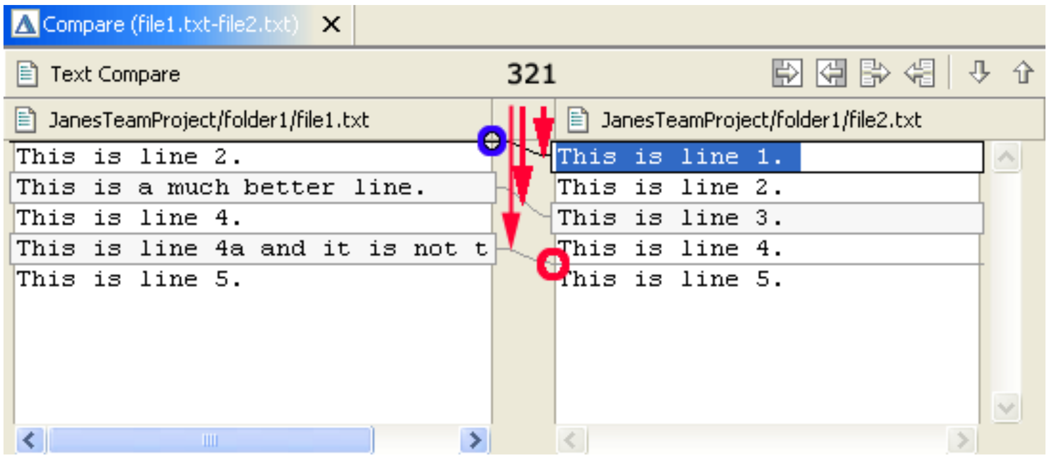
```
This is line 2.
This is a much better line.
This is line 4.
This is line 4a and it is not the sam
This is line 5.
```

```
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
```

Understanding the comparison

Comparing file1.txt and file2.txt resulted in the following compare editor. The left side shows the contents of file1.txt and the right side shows the contents of file2.txt. The lines connecting the left and right panes indicate the differences between the files.

If you need more room to look at the comparison you can double click on the editor tab to maximize the editor.


Compare (file1.txt-file2.txt)
Text Compare
321
JanesTeamProject/folder1/file1.txt
JanesTeamProject/folder1/file2.txt

```
This is line 2.
This is a much better line.
This is line 4.
This is line 4a and it is not t
This is line 5.
```

```
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
```

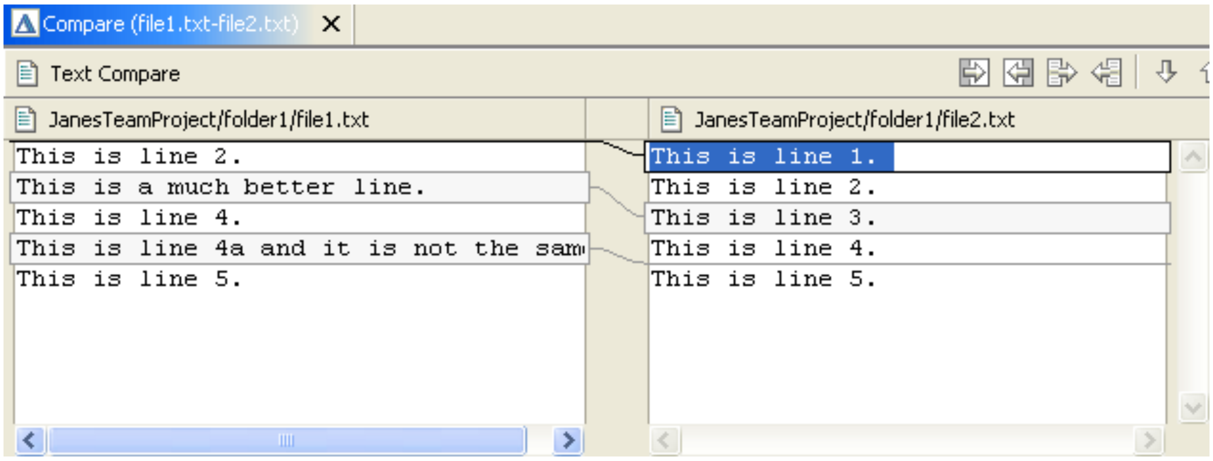
Looking at the numbered changes let's work our way down the left side of the difference editor.

- Starting with the top line (in the left pane) we can see that the difference bar (in the area of the blue circle) indicates something is missing from the very top of the left file. If we follow the difference band (see #1) to the right file we can see that the it contains "This is line 1" .
- The next line "This is line 2." is white indicating it matches the right file.
- Moving onto the next line (colored in the background color) we can see that the left file and right file have different contents for this line (see #2).
- The next line (This is line 4) is once again in white, so we can skip it.
- The next line exists in the left file but since it is in the background color we follow its difference bar to the right (see #3) and notice that the right file does not contain the line (see red circle).

Initially the compare editor might seem a bit daunting but if you work down the left side and focus on the items marked as gray, and those items missing from the left side, it turns out not to be as tricky as it first seems.

Working with the comparison

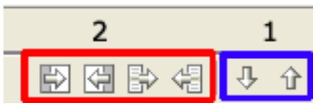
Comparing file1.txt and file2.txt resulted in the following compare editor. In this section we will learn how to use the compare editor to resolve the differences between the two files.


Compare (file1.txt-file2.txt)
Text Compare
JanesTeamProject/folder1/file1.txt
JanesTeamProject/folder1/file2.txt

```
This is line 2.
This is a much better line.
This is line 4.
This is line 4a and it is not the sam
This is line 5.
```

```
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
```

There are two parts to the compare editor's local toolbar. The right group of local toolbar buttons (#1) allows you to move to the next or previous change.



1. Click the Select Next Change button ⇩. Observe how it selects the next difference.
2. Click Select Next Change button a second time to go to the next change.
3. Click the Select Previous Change button.

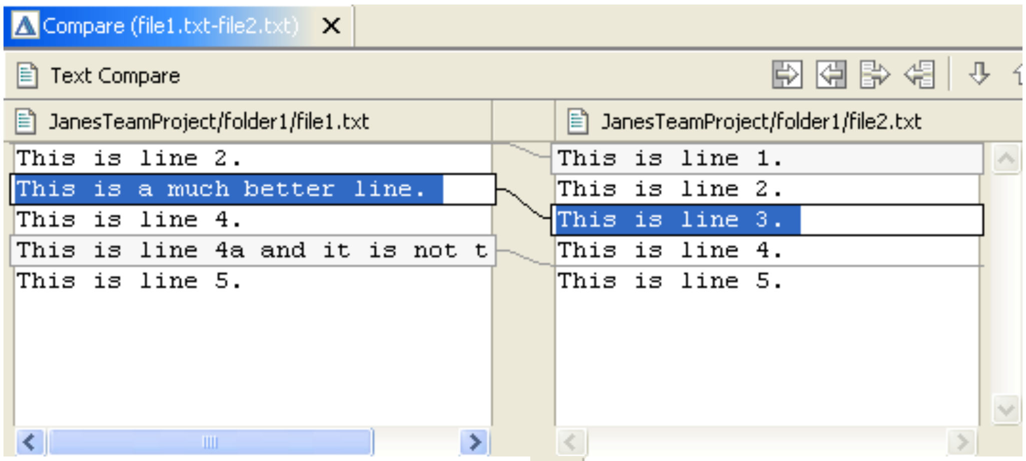
The left group of local toolbar buttons (see #2) allows you to merge changes from the left file to the right file and vice versa. There are four types of merges you can perform:

- Copy whole document from left to right.
- Copy whole document from right to left.
- Copy current change from left to right.
- Copy current change from right to left.

Typically the copy whole document actions are used when you know the entire file on either the left or right can just be replaced by the contents of the other file.

The Copy current change buttons allow you to merge a single change.

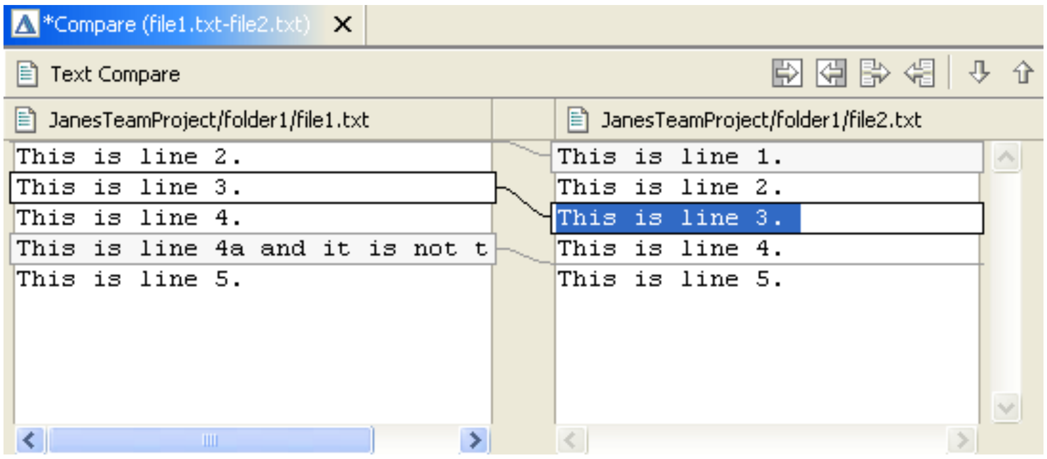
1. Ensure that the second difference is selected (as shown below):


Compare (file1.txt-file2.txt)
Text Compare
JanesTeamProject/folder1/file1.txt
JanesTeamProject/folder1/file2.txt

```
This is line 2.
This is a much better line.
This is line 4.
This is line 4a and it is not t
This is line 5.
```

```
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
```

2. Click Copy current change from right to left button. Observe that the selected text from the right file is copied to the left file.


*Compare (file1.txt-file2.txt)
Text Compare
JanesTeamProject/folder1/file1.txt
JanesTeamProject/folder1/file2.txt

```
This is line 2.
This is line 3.
This is line 4.
This is line 4a and it is not t
This is line 5.
```

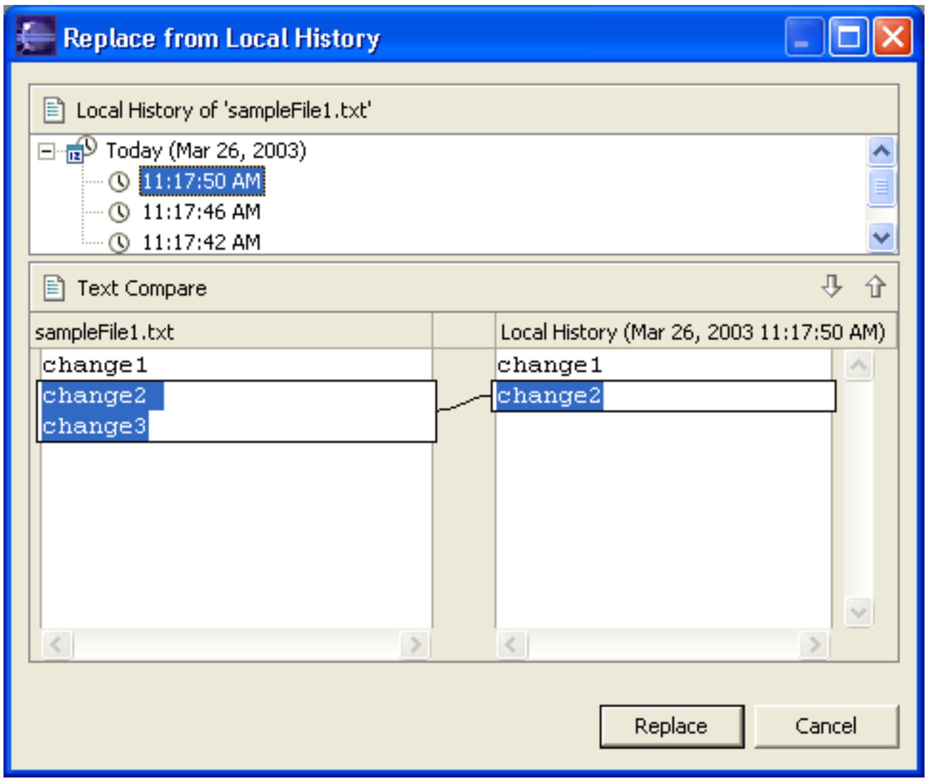
```
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
```

3. Close the compare editor and choose **Yes** to save the changes. Alternatively you can save the changes by choosing **File** > **Save** (Ctrl+S).

Local history

Every time you save an editable file in the Workbench, the Workbench updates the local history of that file and logs the changes that you have made. You can then access the local history of a file and revert to a previously saved copy of the file, as long as the desired state is recent enough in the save history.

1. Create a new file named sampleFile.txt.
2. In the editor for sampleFile.txt modify the resource by adding the line "change1" and saving the file.
3. Repeat this by entering a new line "change2" and saving it again.
4. Add a third line "change3" and save it again.
5. From the resource's context menu in the Navigator view, select **Replace With** > **Local History**.
6. The Replace from Local History dialog opens and shows the previous local history of the file.


Replace from Local History
Local History of 'sampleFile1.txt'
Today (Mar 26, 2003)
11:17:50 AM
11:17:46 AM
11:17:42 AM
Text Compare
sampleFile1.txt
Local History (Mar 26, 2003 11:17:50 AM)
Replace
Cancel

```
change1
change2
change3
```

```
change1
change2
```

The left pane of the dialog contains the Workbench's copy of the file. In the figure above we can see the Workbench contains the copy with all 3 lines − the same copy that is currently shown in the editor area of the Workbench.

The first item in the local history (see above) contains the last saved copy of the file. This is the one with only two lines of text. The final entry in the tree is the first copy of the file.

The bottom area of the dialog allows us to see the differences between the Workbench file and the specific copy of the file selected in the local history.

7. Select the first item (shown above) in the local history. The right pane should show one line of text.
8. Click **Replace**. This replaces the Workbench's copy of sampleFile.txt with the chosen local history item.
9. Observe that the sampleFile.txt editor now contains two lines.

Exiting the Workbench

Each time you exit the Workbench, the Workbench is automatically saved, including all open perspectives and windows. The next time you reopen the Workbench, it will appear exactly as it was when you closed it.

To exit the Workbench, select **File** > **Exit** from the menu bar. You will be prompted if you really wish to exit the workbench. Note that his dialog also has an option to turn off the prompting if you wish.

Team CVS tutorial

In this chapter, we will learn to use the CVS team capabilities built into the Workbench. The steps in this chapter are team oriented; it will be difficult to complete these tutorials if others are not simultaneously working through this chapter.

We will learn how to work on our project, then commit changes to the repository for others on the team to use. As we continue to work on the project alongside other users we will see how to commit resources that other users may be simultaneously working on, and how to update our workspace with changes others make in the project.

Remember, before we get started you need to entice at least one coworker into working through these steps with you.

Setting up a CVS repository

A repository is a persistent store that coordinates multi−user access to the resources being developed by a team. The Workbench comes with CVS support built−in. The CVS server is available at http://www.cvshome.org.

Please refer to this site for information on how to install and configure your CVS repository including user access and passwords.

In order to continue with this tutorial you must have access to a CVS repository.

Starting offline

We will start our team CVS tutorial by working offline and creating a simple project. Once the project is made we will see how to commit it to the repository.

1. Create a new "Simple" project using **File > New > Project** Use your name as the project name (e.g. JanesTeamProject).
2. Create a folder named folder1.
3. Create two text files (.txt) in folder1 called file1.txt and file2.txt. Their contents should be as follows:

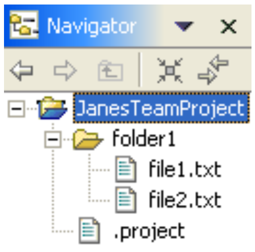
   file1.txt

   > *This is the contents*
   > *of file 1.*

   file2.txt

   > *File2 is a small file*
   > *with simple text.*

The Navigator should now appear as follows:



We can continue to work with our project in this mode but unless we commit the project into the repository others on our team will be unable to work on the project with us. In the next few sections we will see how to commit our project to the repository.

Sharing our project

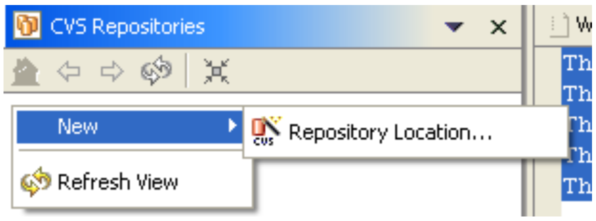
We have created our project in offline mode. To make this project available to other team members we need to do two things:

1. Specify the location of our team's CVS repository.
2. Commit the work into that repository.

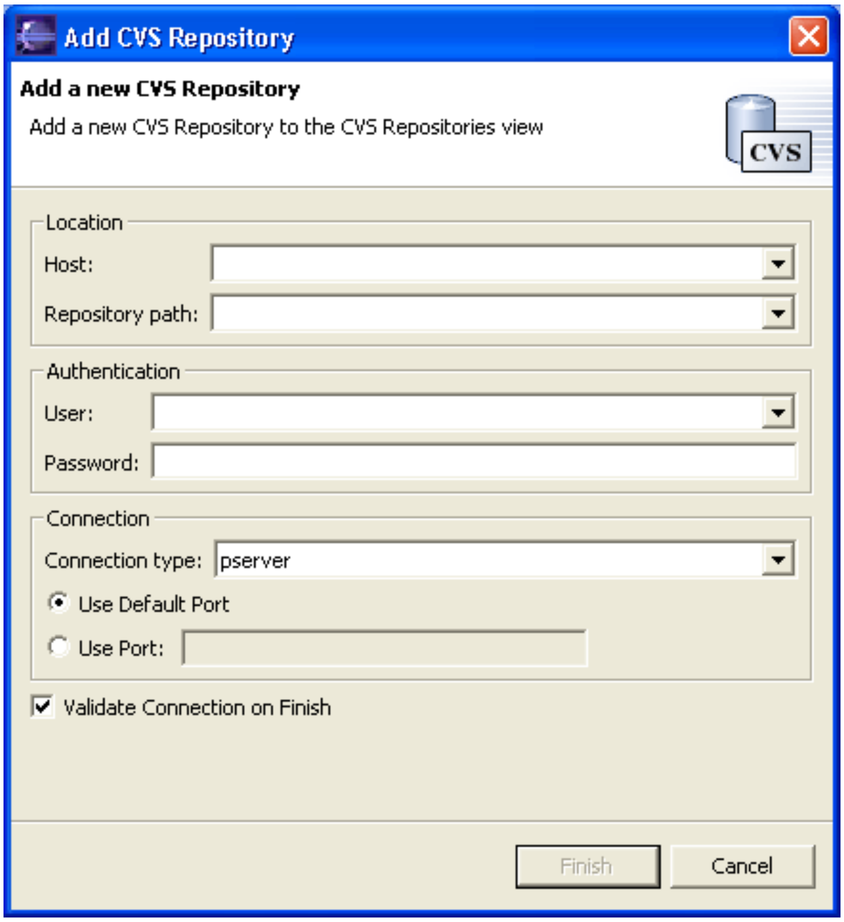
Specifying a repository location

Before we can share our project with other users we must first specify an available repository.

1. Open the CVS Repository Exploring perspective. The top left view shows all of the CVS repositories that we are currently working with. As you can see, the view is empty, meaning we still need to specify a repository.
2. In the context menu of the CVS Repositories view choose **New** > **Repository Location**

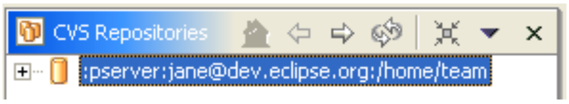


3. In the CVS Repository Location wizard we need to fill in the location of our repository and our login information. You may require assistance from your repository administrator in order to fill in the necessary information.



4. In the **Host** field, type the address of the host (e.g., "teamsamples.com").
5. In the **Repository path** field, type the path for the repository at the host address (e.g., "/home/cvsroot/repositoryName").
6. In the **User** field, type the user name under which you want to connect.
7. In the **Password** field, type your password.
8. In the **Connection type** field, select the type of CVS connection for the repository (The default is pserver).
9. Leave **Use Default Port** enabled.
10. By default the **Validate Connection on Finish option** is checked.
11. Click **Finish** when you are done.

    Since we checked **Validate location on finish,** the wizard will now attempt to validate the information by connecting to the repository. In doing so it may prompt you for your password. Note that the repository connection is only used to validate the information.
12. Observe that the CVS Repositories view now shows the new repository location.



What is a repository location?

We have just added a new repository location to the CVS Repositories view. This is probably a good time to ask ourselves what a repository location is or, more importantly, what it isn't.

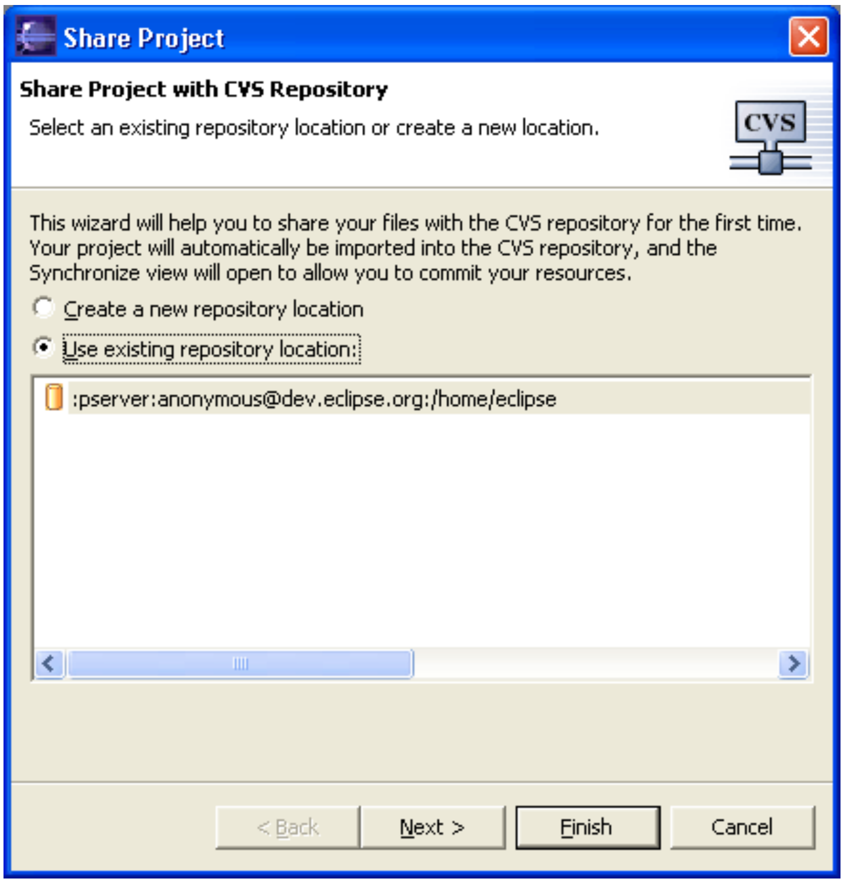
A repository location is not an actual live connection. Instead, it is a description of where the repository is located. At a later time, when we are committing work to the repository or updating with work done by others, the Workbench will create a connection based on this location information. Connections will be opened and closed as required when performing CVS team operations, and those connections will be based on the information you provided in the repository location.

Suppose we disconnected from the network and went home. The CVS Repositories view would continue to show us the list of known repository locations. In addition, the projects themselves will still know the repository location they are associated with.

Synchronizing our project

We have created our project and specified our repository location. Now we need to make our project available to other team members.

1. In the Navigator view select the project JanesTeamProject.
2. From the project's context menu choose **Team** > **Share Project**. If you have more than one repository provider installed, pick CVS and press Next.
3. In the sharing wizard page, select the location we created previously.



Share Project with CVS Repository

Select an existing repository location or create a new location.

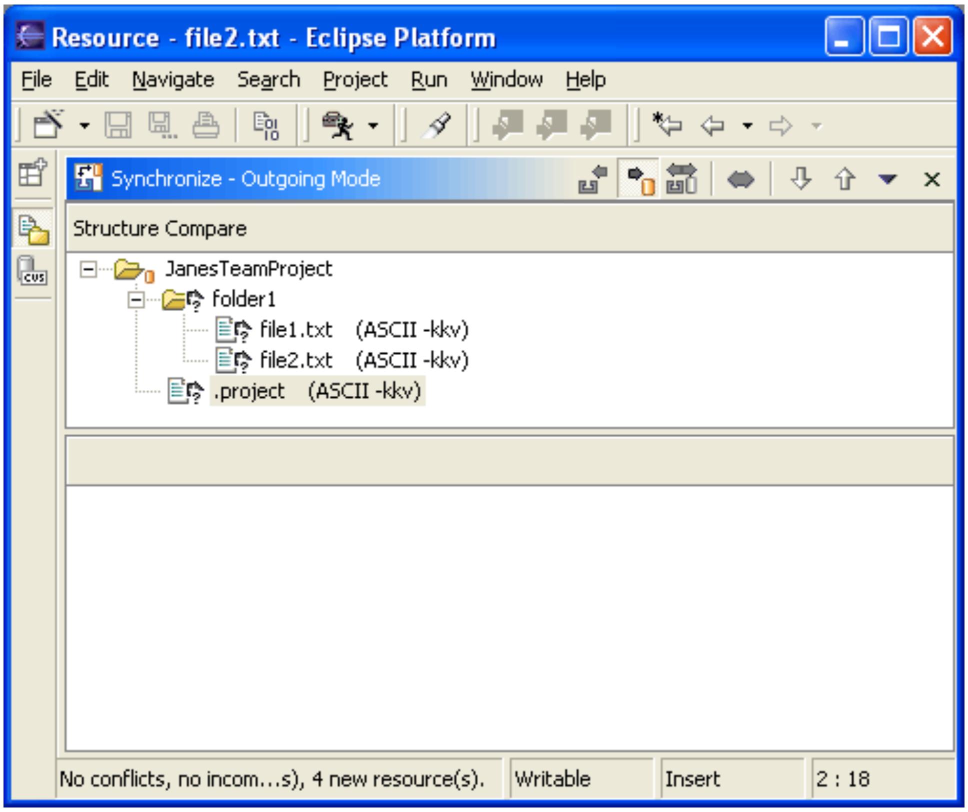
This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Synchronize view will open to allow you to commit your resources.

○ Create a new repository location

● Use existing repository location:

:pserver:anonymous@dev.eclipse.org:/home/eclipse

4. Click **Finish** when you are done.
5. Observe that the Synchronize view automatically opens. In the next section we will see how to use this view to commit our work to the repository.


Resource - file2.txt - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Synchronize - Outgoing Mode
Structure Compare

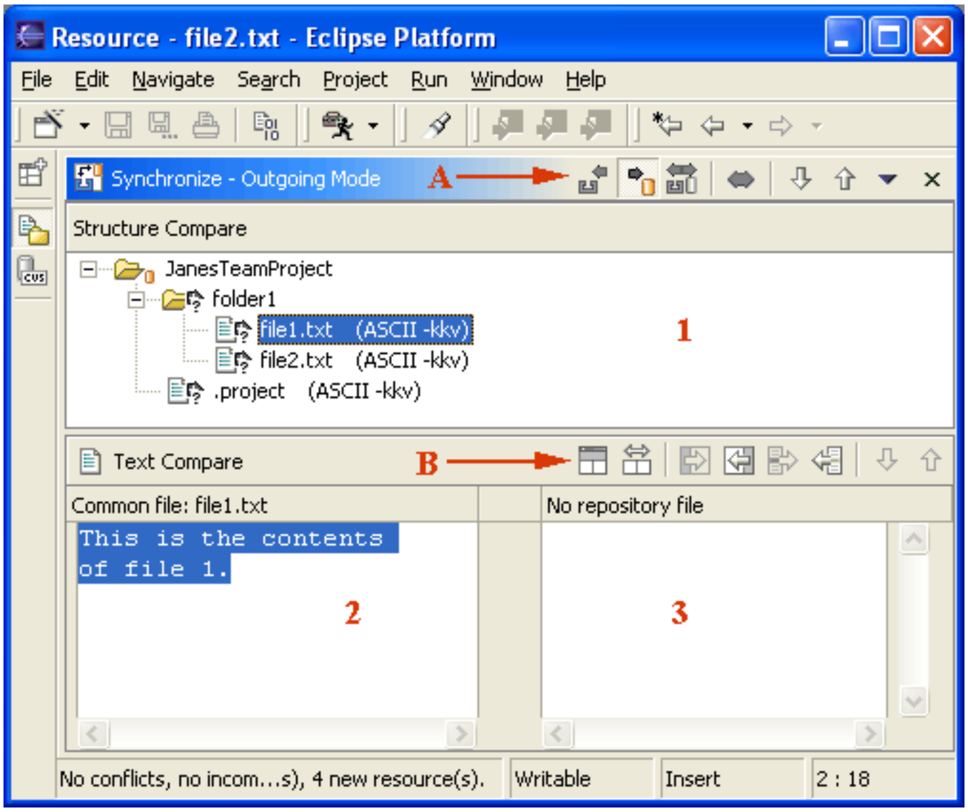
JanesTeamProject
folder1
file1.txt (ASCII -kkv)
file2.txt (ASCII -kkv)
.project (ASCII -kkv)

No conflicts, no incom...s), 4 new resource(s). Writable Insert 2 : 18

Synchronize view

The Synchronize view provides you with a single place where you can see both the changes you have made, and changes others have made and committed to the repository. In addition to providing a combined *local* + *remote* resource tree, it allows you to put your changes in the repository (commit), and get the changes others have made (update).

Let's take a look at the Synchronize view in more detail.



```
This is the contents
of file 1.
```

No conflicts, no incom...s), 4 new resource(s). Writable Insert 2 : 18

The first thing to notice is that the title of the view indicates we are in "Outgoing Mode" and the Outgoing Mode button (see A) is pressed in. Outgoing mode allows us to commit changes we have made locally to the repository. By default if you Synchronize with Repository and the repository has no changes for you to take, you are automatically placed into outgoing mode to let you quickly commit your changes. We'll come back to "Incoming Mode"a little later.

In "Outgoing Mode" the top area (#1) shows all of the resources that need to be committed.

The bottom area (#2 and #3) is the text compare area we worked with in the basic tutorial. By double−clicking on a resource in the **Structured Compare** list the lower area displays the differences between what we have in the Workbench (#2) versus what is currently in the repository (#3).

Toolbar buttons (A) allow you to switch between outgoing mode and other synchronization modes that will be examined in a later section.

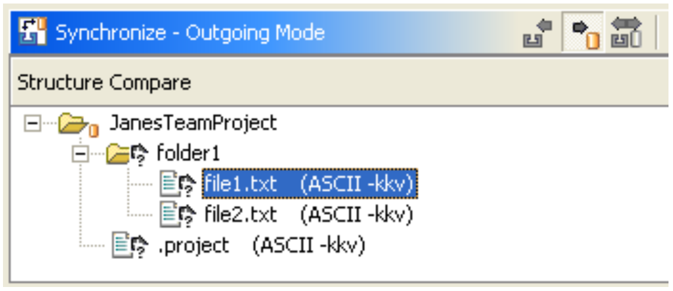
The lower toolbar (B) allows you to merge changes between your file and the repository copy of the file.

Committing

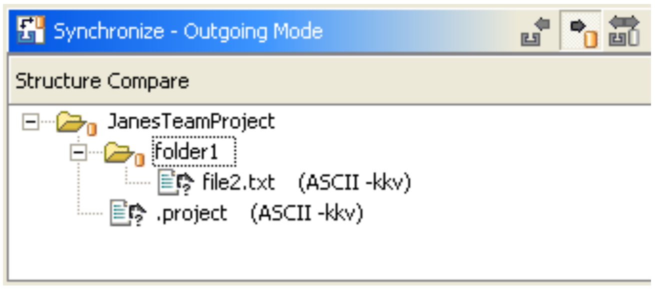
Now that we have a better understanding of the Synchronize view let's go ahead and commit the work we have done.

1. In the **Structure Compare** list select file1.txt

.



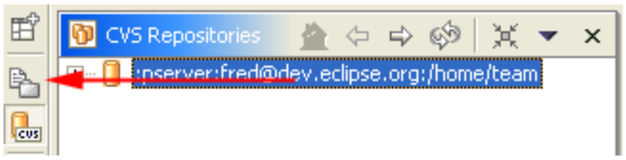
2. Pop−up the menu and choose **Commit**. Click **OK** when asked to add and to supply a commit comment.
3. Observe that file1.txt is no longer in the list.



In addition to committing individual resources we can also select the project and commit all of the changes in the project.

4. Select JanesTeamProject.
5. Choose **Commit** from the project's context menu and type in "my first project" when prompted for a commit comment.

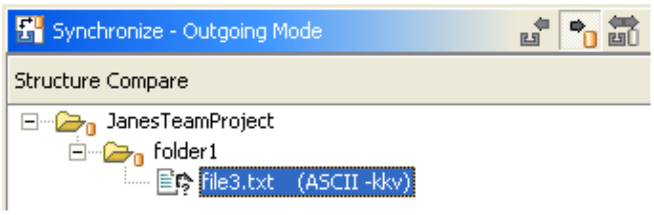
   Observe that the remaining resources in the project are committed and the structure compare list is now empty.
6. Click on the Resource perspective button in the shortcut bar (at the far left of the window).



7. In the navigator create a third file called file3.txt with the following contents:

   This is the brief contents
   of file 3.
8. Select the project (JanesTeamProject) and choose **Team > Synchronize with Repository** again from the context menu.

   When the Synchronize view opens, there is only one change, the new file3.txt.



9. Select file3.txt, and choose **Commit** from the context menu.

Working with another user

We have seen how to create a project and commit it to the repository. We first specified our repository location. Next we shared our project with that location and the HEAD branch. Then we synchronized our project to open the Synchronize view. Finally we committed the resources using the Synchronize view.

The repository currently contains our project and the three files that we committed (file1.txt, file2.txt and file3.txt).

It's time to find that coworker (let's call him Fred) to work through some steps with you. In this section we will learn how two people can work on the same project and simultaneously commit changes to the repository. Specifically we will do the following:

- Fred will add the project to his Workbench
- Fred will make changes to file1.txt and file2.txt
- You will simultaneously make changes to file2.txt and file3.txt
- Fred will synchronize and commit his (outgoing) changes to the two files.
- You will synchronize to commit your changes at which time you will discover there are outgoing (file3.txt), incoming (file1.txt) and conflicts (file2.txt).

incoming	Changes that are in the repository, which you have not yet updated to.
outgoing	Changes that you are committing.
conflict	Both you and the repository have modifications to the same resource.

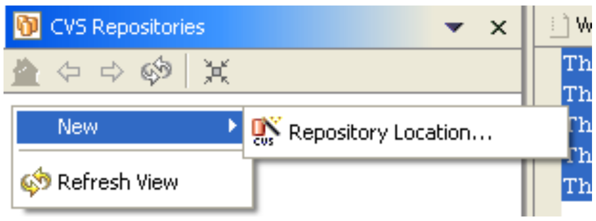
Checking out a project

Coworker Fred has several tasks in front of him:

- Fred will add the project to his Workbench
- Fred will make changes to file1.txt and file2.txt
- Fred will synchronize and commit his outgoing changes to the two files.

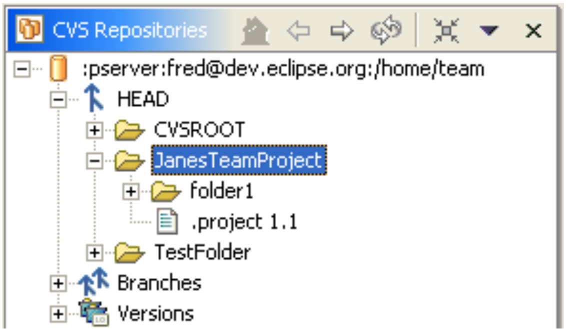
Fred's first step is to add the project to his workspace as follows:

1. In the CVS Repositories view context menu choose **New > Repository Location**

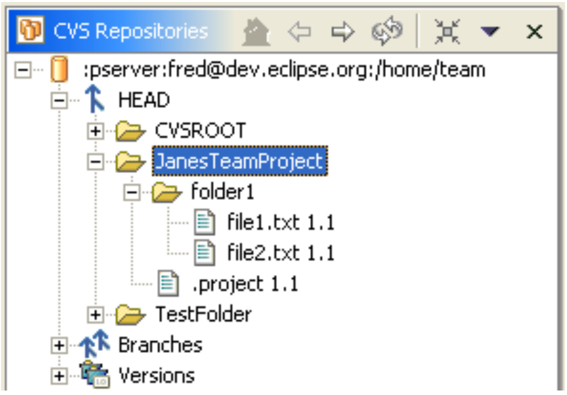


2. In the CVS Repository Location wizard fill in the same repository location that Jane filled in.
3. In the CVS Repositories view expand the repository location, then expand HEAD.

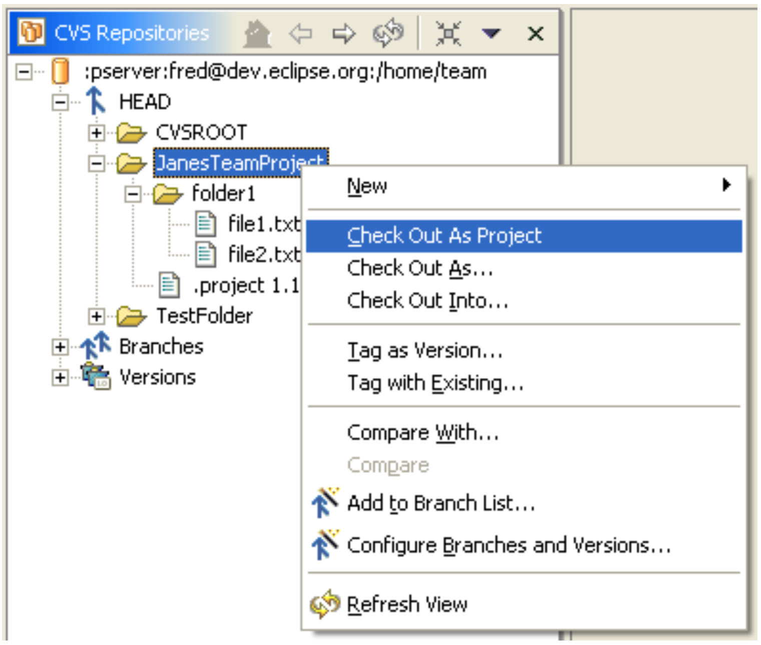
   Under HEAD you should see the project (JanesTeamProject) that Jane created.



4. Expand folder1 and you should see the three files that Jane created.



5. Select the project and choose **Checkout as Project** from the context menu.



6. Select the Navigator view and observe that it now includes the project JanesTeamProject.

Another user making changes

Now that Fred has the project in his Workbench he needs to modify several of the files and synchronize with the repository in order to commit them.

- Fred will make changes to file1.txt and file2.txt
- You will simultaneously make changes to file2.txt and file3.txt
- Fred will synchronize and commit his outgoing changes to the two files.

Fred should proceed as follows:

1. Modify file1.txt as follows

   Original contents:

   This is the contents
   of file 1.

   New contents (changes shown in bold):

   This is the contents
   Fred−update
   of file 1.
2. Modify file2.txt as follows

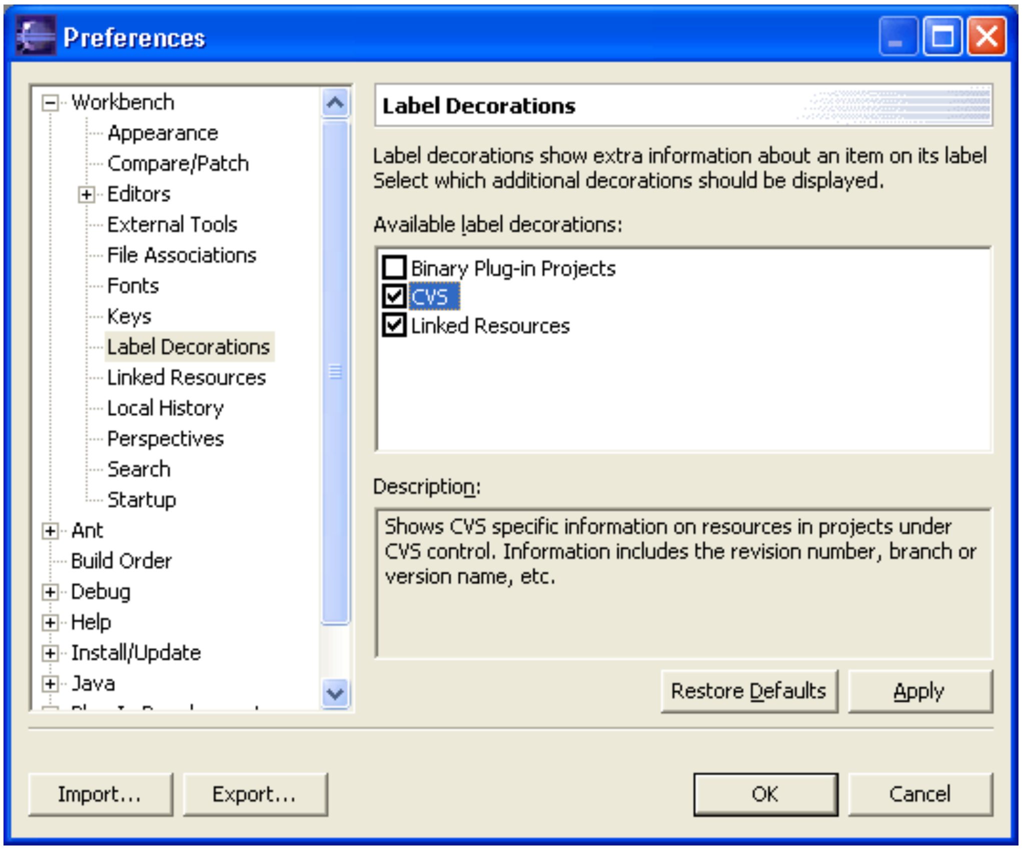
   Original contents:

File2 is a small file
with simple text.

New contents (changes shown in bold):

*File2 is a (**Fred was here)** small file*
with simple text.

3. Open the preferences (**Window > Preferences**) and turn on the CVS label decorations (on page **Workbench > Label Decorations** check off **CVS**).



Preferences

Workbench: Appearance, Compare/Patch, Editors, External Tools, File Associations, Fonts, Keys, Label Decorations, Linked Resources, Local History, Perspectives, Search, Startup; Ant, Build Order, Debug, Help, Install/Update, Java

Label Decorations

Label decorations show extra information about an item on its label
Select which additional decorations should be displayed.

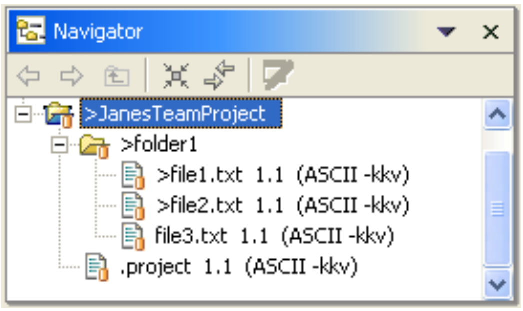
Available label decorations:

- [ ] Binary Plug-in Projects
- [✓] CVS
- [✓] Linked Resources

Description:

Shows CVS specific information on resources in projects under CVS control. Information includes the revision number, branch or version name, etc.

Observe that the Navigator updates the CVS state of a resource. Notice that the two files which Fred has changed are preceded by ">".



Navigator

```
>JanesTeamProject
  >folder1
    >file1.txt  1.1  (ASCII -kkv)
    >file2.txt  1.1  (ASCII -kkv)
    file3.txt  1.1  (ASCII -kkv)
  .project  1.1  (ASCII -kkv)
```

4. To commit his changes to the repository Fred can either:
   - ♦ Select the two files and choose **Team > Synchronize with Repository** or,
   - ♦ Select the project and choose **Team > Synchronize with Repository**

   Often choosing the project is the easiest thing to do. Let's do that. Select the project and choose **Team**

> **Synchronize with Repository** from its context menu.
5. When the Synchronize view opens Fred should commit his changes to file1.txt and file2.txt.

Making our own changes

Fred has made several changes to file1.txt and file2.txt and committed them to the repository. We now need to make some changes for ourselves and then synchronize with the repository. When we synchronize we expect to see the changes we have made along with changes that have been made by Fred.

1. Start by modifying file2.txt as follows

   Original contents:

   > *File2 is a **Jane−update** small file*
   > *with simple text.*

   New contents: (changes shown in bold)

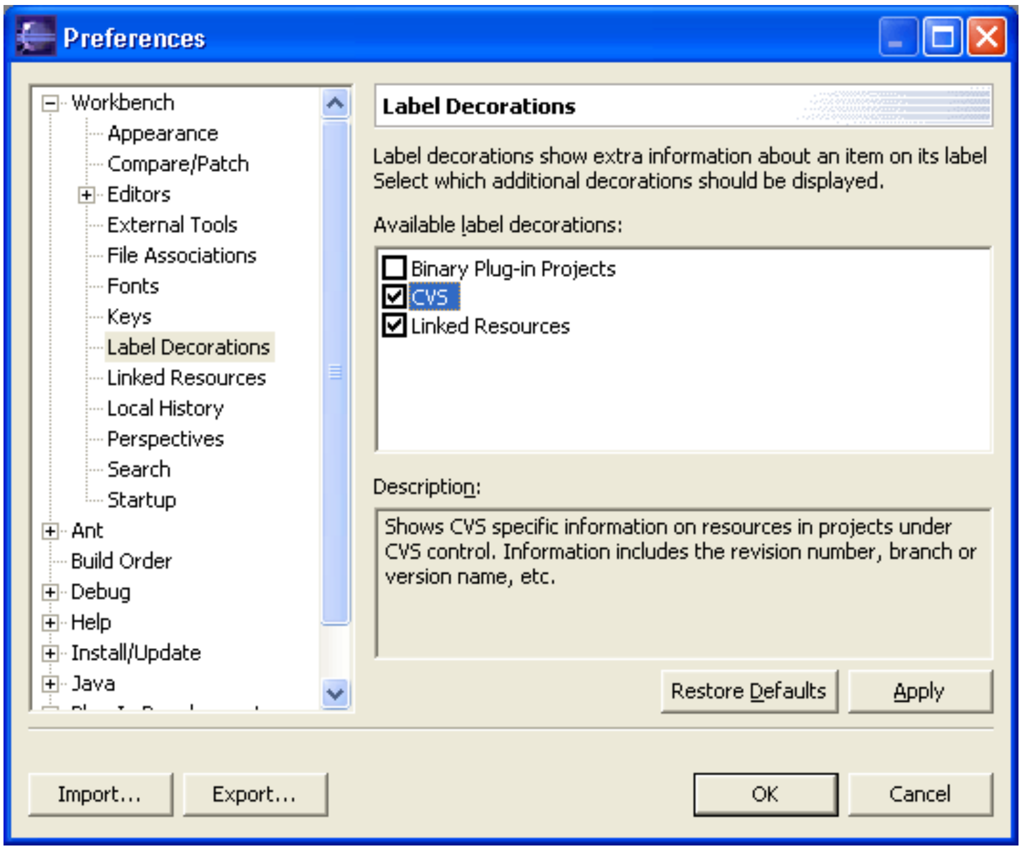
   > *This is the **Jane−update** contents*
   > *of file 2.*
2. Modify file3.txt as follows

   Original contents:

   > *This is the brief contents*
   > *of file 3*

   New contents:(changes shown in bold)

   > *This is the brief contents*
   > *of file **(Jane was here)** 3*
3. Open the preferences (**Window > Preferences**) and turn on the CVS label decorations (on page **Workbench > Label Decorations** and ensure **CVS** is checked).



Label Decorations

Label decorations show extra information about an item on its label Select which additional decorations should be displayed.

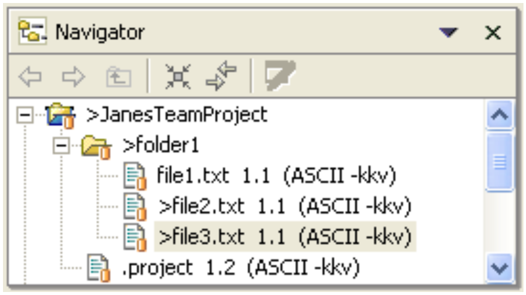
Available label decorations:

- [ ] Binary Plug-in Projects
- [✓] CVS
- [✓] Linked Resources

Description:

Shows CVS specific information on resources in projects under CVS control. Information includes the revision number, branch or version name, etc.

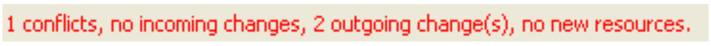
Observe that the Navigator updates the CVS state of a resource. Notice that the two files which Fred has changed are preceded by ">".



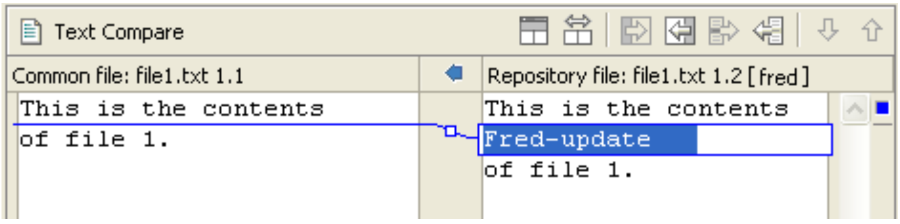
```
>JanesTeamProject
  >folder1
    file1.txt  1.1  (ASCII -kkv)
    >file2.txt  1.1  (ASCII -kkv)
    >file3.txt  1.1  (ASCII -kkv)
  .project  1.2  (ASCII -kkv)
```

4. Select the project (JanesTeamProject)
5. From the project's context menu, select **Team > Synchronize with Repository**.
6. There are a couple of other things worth observing. First, the icon next to file1.txt (in the structured compare area) indicates that file1.txt has an incoming change. This means a change was released to the server which you need to take. Looking at file2.txt we can see a red icon. This indicates that both yourself and the repository have changes to the same file. This means that someone else (namely Fred) made changes to the same resource as the one you want to commit.

   The second thing worth noting is that the status bar has red text indicating what changes need to be considered.

   1 conflicts, no incoming changes, 2 outgoing change(s), no new resources.
7. Select file1.txt (the incoming change). The comparison viewer shows:



Common file: file1.txt 1.1

```
This is the contents
of file 1.
```

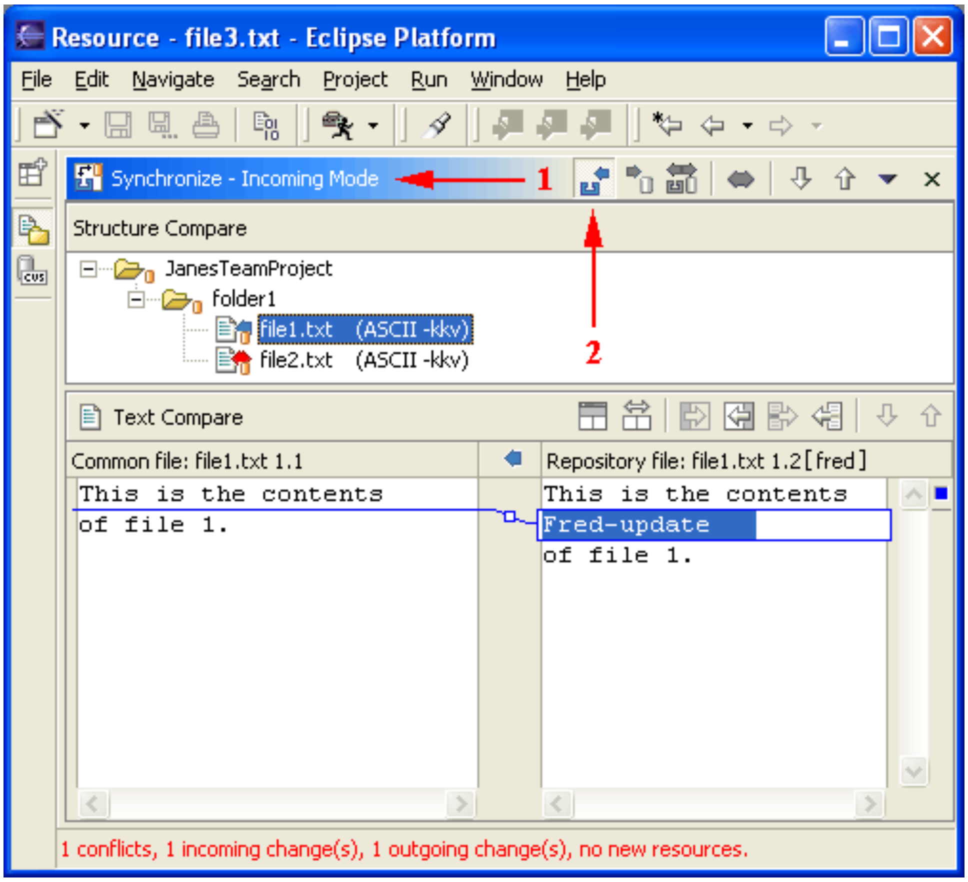
Repository file: file1.txt 1.2 [fred]

```
This is the contents
Fred-update
of file 1.
```

Updating

We will now use the Synchronize view to update our local resource to the latest contents in the repository and to commit our changes.

1. Confirm that the Synchronize view is in Incoming Mode (#1). If it is not then press the Incoming Mode button on the local toolbar (#2).



Common file: file1.txt 1.1

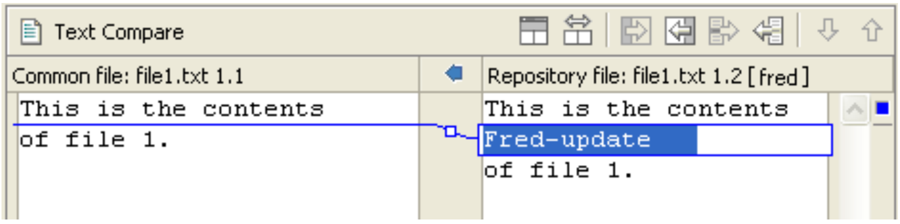
```
This is the contents
of file 1.
```

Repository file: file1.txt 1.2 [fred]

```
This is the contents
Fred-update
of file 1.
```

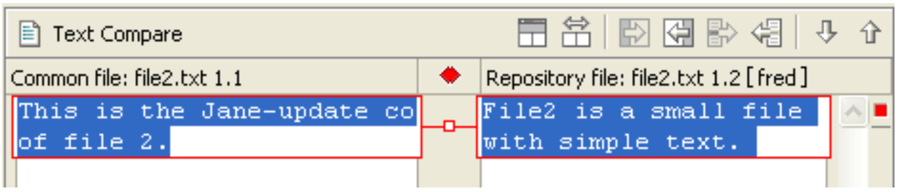
1 conflicts, 1 incoming change(s), 1 outgoing change(s), no new resources.

2. Open file1.txt (the incoming change) by double−clicking. The comparison viewer shows that the repository copy of file1.txt has a new line inserted (Fred−update).



```
Text Compare
Common file: file1.txt 1.1          Repository file: file1.txt 1.2 [fred]
This is the contents                This is the contents
of file 1.                          Fred-update
                                    of file 1.
```

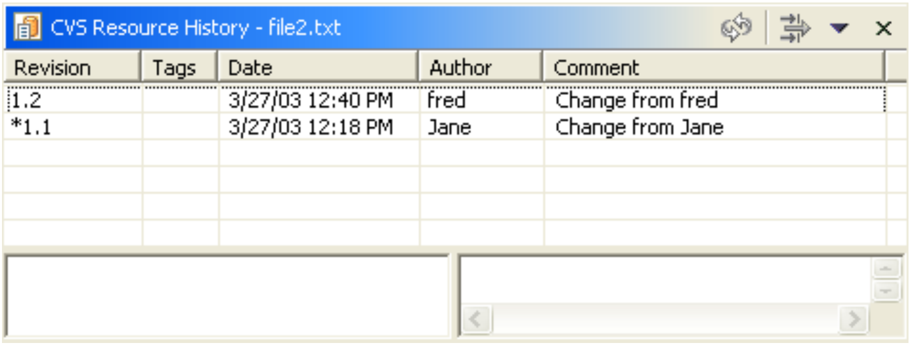
3. From the context menu of file1.txt choose **Update**. This will cause your Workbench to be updated with the repository copy of file1.txt.
4. Select file2.txt. The comparison viewer shows that the repository copy of file1.txt and our copy of file1.txt have both had the exact same line modified.



```
Text Compare
Common file: file2.txt 1.1          Repository file: file2.txt 1.2 [fred]
This is the Jane-update co          File2 is a small file
of file 2.                          with simple text.
```

5. If you are suspicious of the change you may want to find out some more information about it. To do this you first need to figure out who made the change and why.

   In the Structure compare pane of the Synchronize view select file2.txt and choose **Show in Resource History** from the context menu.



CVS Resource History - file2.txt

Revision	Tags	Date	Author	Comment
1.2		3/27/03 12:40 PM	fred	Change from fred
*1.1		3/27/03 12:18 PM	Jane	Change from Jane

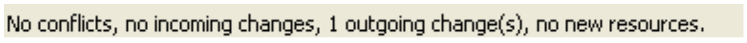
6. In the Resource History view (see above) we can see there is a * next to version 1.1. This indicates 1.1 is currently in our Workbench. We can also see the repository contains 1.2 of file2.txt. The author of 1.2 is Fred. Looks like it's time to ask Fred why he was changing file2.txt.

   If Fred had entered a meaningful description when releasing his changes, these descriptions would have appeared in the comment field of the Resource History view thereby providing us with some insight into the intent of the change.
7. You can use the four copy buttons to help you merge in conflicting changes made to the repository.

   For now we will assume that Fred's change turns out to be the right one.

   Select file2.txt and choose **Override and Update** from the context menu. When asked to confirm the replacement choose **Yes**.
8. The Update mode no longer has any remaining incoming or conflicting changes.

   Looking at the status bar we can see there is still one remaining outgoing change to deal with.

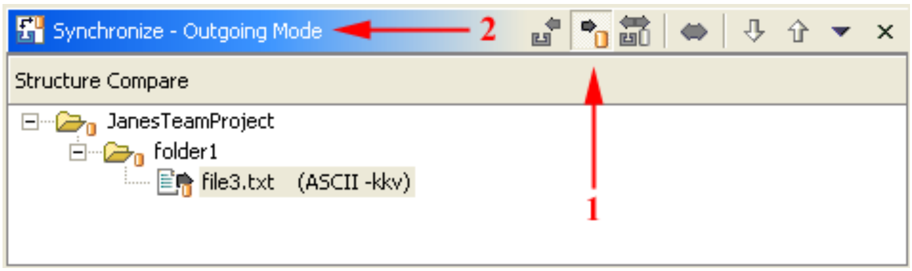


No conflicts, no incoming changes, 1 outgoing change(s), no new resources.

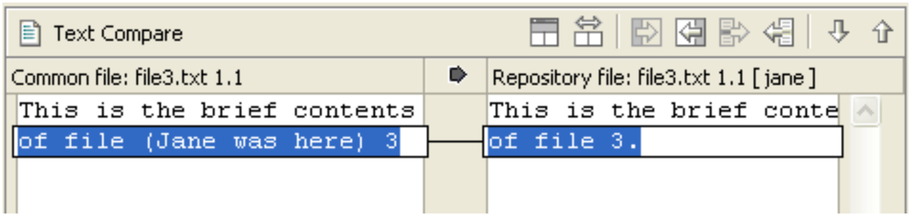
Committing

We will now use the Outgoing Mode of the Synchronize view to commit the one remaining outgoing change.

1. Place the Synchronize view into outgoing mode by clicking on the Outgoing Mode button (#1) and confirming that the title of the view now indicates we are in Outgoing Mode (#2).



2. Open file3.txt (the outgoing change) by double−clicking.
3. The comparison viewer shows that our copy of file3.txt has a change to the second line. The repository copy is shown in the right pane.


Text Compare
Common file: file3.txt 1.1
Repository file: file3.txt 1.1 [jane]

```
This is the brief contents      This is the brief conte
of file (Jane was here) 3       of file 3.
```

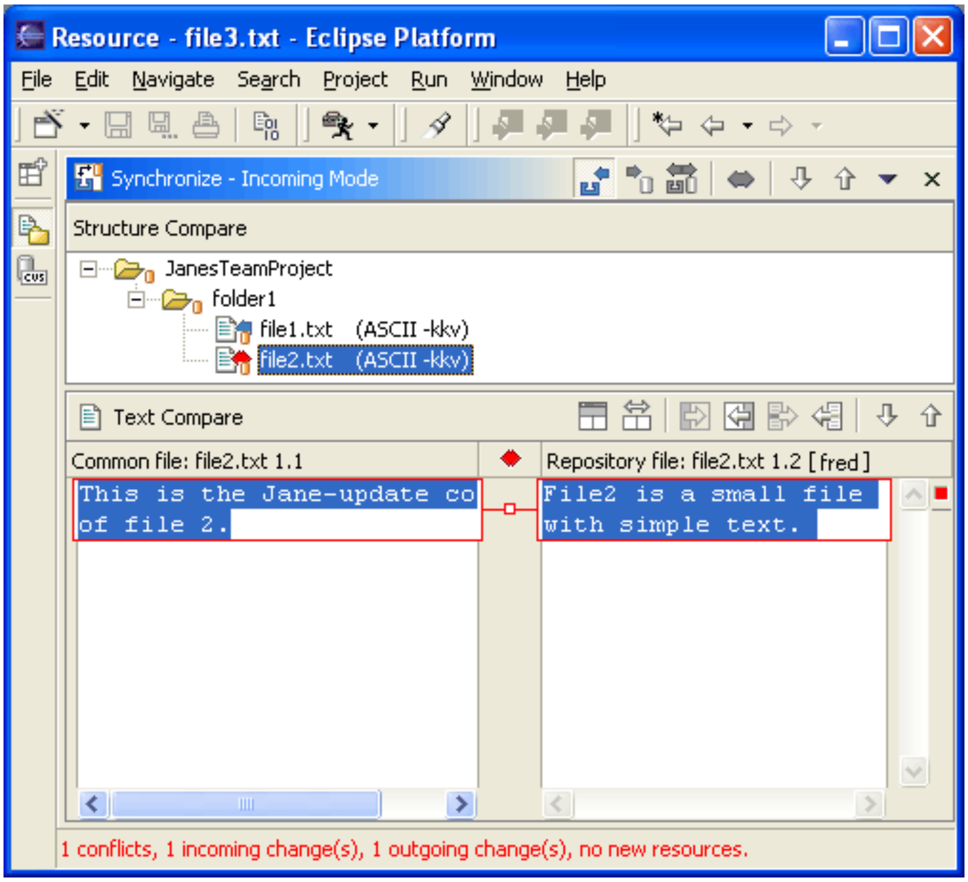
4. From the context menu of file3.txt choose **Commit**. Be sure to enter a meaningful description when prompted for one.

   Your Workbench copy is now committed into the repository.
5. At this point Fred may want to synchronize with the repository to update his workspace with the changes you just committed.

We have successfully collaborated with Fred by making simultaneous changes to the same project. The Synchronize view and CVS Resource History view allowed us to determine the specific changes each user was making and to quickly stay on top of the ongoing changes being committed into the repository.

Synchronize view

If you look closely at the title of the Synchronize view you will see that we have started in Incoming Mode (see #1). This is because there are changes in the repository that we should first integrate before committing our own changes.


Resource - file3.txt - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Synchronize - Incoming Mode
Structure Compare
JanesTeamProject
folder1
file1.txt (ASCII -kkv)
file2.txt (ASCII -kkv)
Text Compare
Common file: file2.txt 1.1
Repository file: file2.txt 1.2 [fred]

```
This is the Jane-update co
of file 2.
```

```
File2 is a small file
with simple text.
```

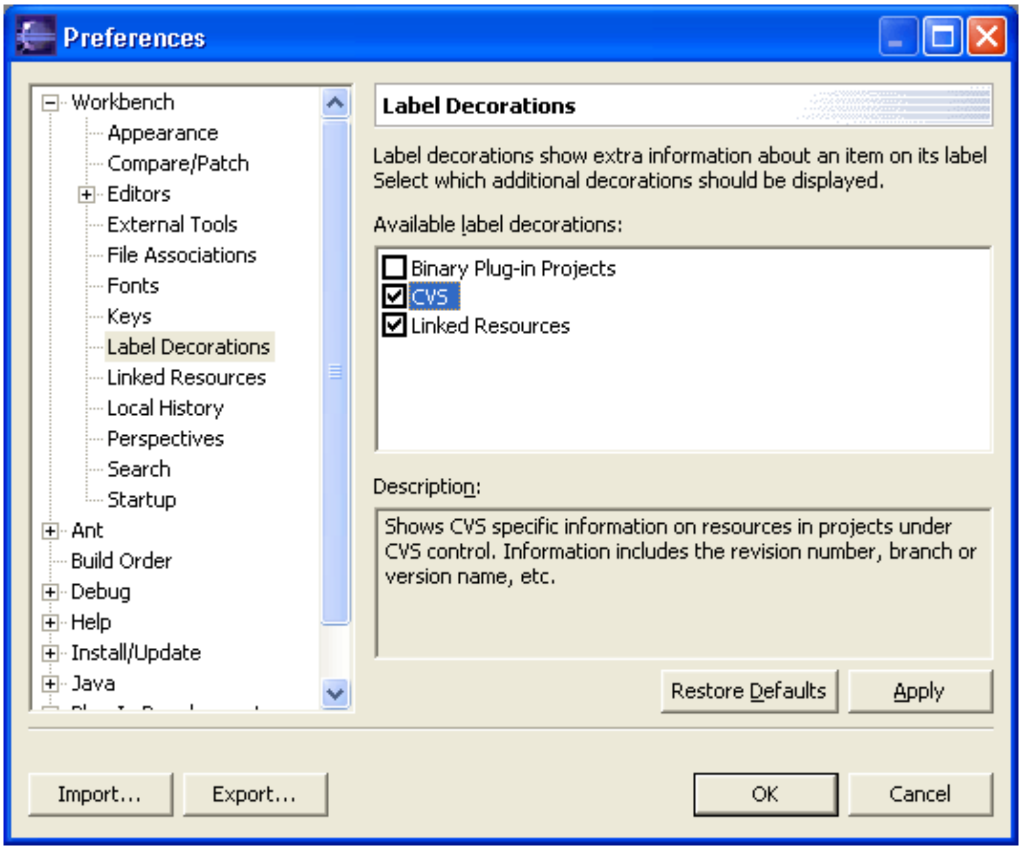
1 conflicts, 1 incoming change(s), 1 outgoing change(s), no new resources.

It is important to first update changes made to the repository, retest with those changes loaded locally with your soon to be committed changes and then commit your changes. By first taking the latest repository changes, and retesting, it helps to ensure that the changes you are about to commit will actually work with the current state of the repository.

Replacing

Suppose after further reflection we realize that the revision of file1.txt that we just received is incorrect and in fact we want an earlier revision. You can replace a Workbench resource with an earlier revision of the resource from the repository. To roll back to an earlier revision:

1. In the Navigator view select the file1.txt
2. From the file's context menu, select **Replace With** > **Revision**
3. In the replace with revision Compare editor that opens, choose the original (bottom) revision and, from the pop−up menu choose **Get Contents**.
4. Open the preferences (**Window** > **Preferences**) and turn on the CVS label decorations (on page **Workbench** > **Label Decorations** and ensure **CVS** is checked).


Preferences

Workbench
Appearance
Compare/Patch
Editors
External Tools
File Associations
Fonts
Keys
Label Decorations
Linked Resources
Local History
Perspectives
Search
Startup
Ant
Build Order
Debug
Help
Install/Update
Java

Label Decorations

Label decorations show extra information about an item on its label
Select which additional decorations should be displayed.

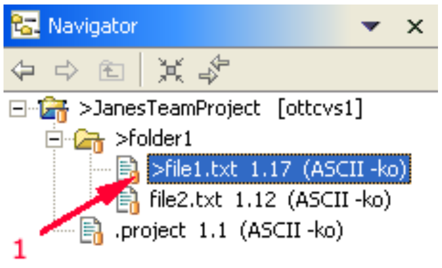
Available label decorations:

- [ ] Binary Plug-in Projects
- [x] CVS
- [x] Linked Resources

Description:

Shows CVS specific information on resources in projects under CVS control. Information includes the revision number, branch or version name, etc.

Observe that the Navigator updates to show the CVS state of a resource. Notice that the modified file is preceded by "> (see #1) indicating that we have changed file1.txt (by replacing it with the earlier version).


Navigator

```
>JanesTeamProject  [ottcvs1]
  >folder1
    >file1.txt  1.17  (ASCII -ko)
    file2.txt  1.12  (ASCII -ko)
  .project  1.1  (ASCII -ko)
```

5. Now that we have finished our coffee we realize that this older revision is not as good as it initially seemed and in fact the revision in the repository is better after all.

   Instead of choosing **Team > Synchronize with Repository**, let's choose **Replace With > Latest from Repository.**

   Observe that file1.txt is updated to be the contents from the repository, and that the leading indicator ">" has been removed since we now have the same revision as the repository.

We have seen how to synchronize with the repository, replace with revision or replace with the latest from the repository. You can also compare with these revisions/versions in a similar manner by choosing the **Compare With** menu operation from a resource's context menu in the Navigator.

A quick review

Now that we have experienced the fun of working in a repository it's time to step back and take note of some of the more important but subtle issues.

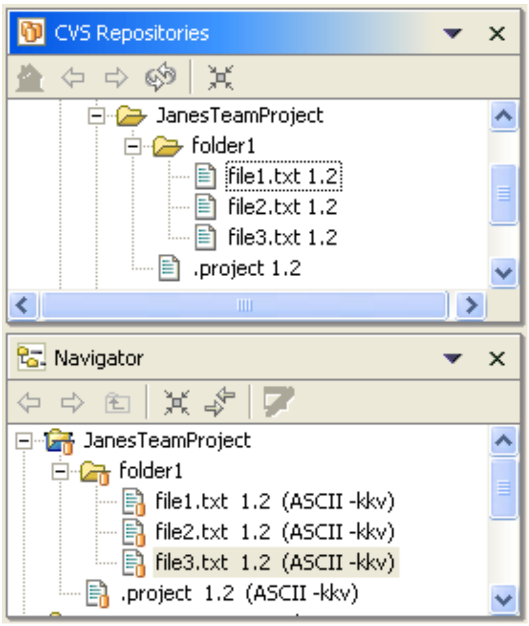
- When we versioned the project we did so by versioning the project as it appeared in our Workbench. For this reason it is important to synchronize the project with the repository (i.e. HEAD or the branch we are working in) prior to versioning it. Otherwise another user may have committed interesting changes to the project which you have yet to update to. If you proceed to version the project without updating, you will version it without these changes.
- The repository contains all projects in the repository. Individual users pick which projects they are interested in and check them out into the workspace. From that point on they are synchronizing those projects (only) with respect to the repository.
- The repository represents a large in−progress collection of all known projects. From the repository's perspective, everything in HEAD or on a branch is always open for change.
- The act of versioning a project effectively snapshots it and places it into the Versions section of the repository, however the repository branches are still open for change.
- It is important to first update to changes made to the repository, retest with those changes and your soon to be committed changes and then commit your changes. By first taking the latest changes in the branch, and retesting, it helps to ensure that the changes you are about to commit will actually work with the current state of the branch.
- Each project is associated with a specific repository. Different projects can be associated with different repositories that may be on completely different servers.

Versioning your project

Now that our project is complete it is time to version it. If you were paying close attention while we were committing our file resources you probably noticed that the repository automatically assigned them version numbers, or to be more accurate "revision numbers". As we committed our files the only information we needed to supply was a commit comment.

To version our project proceed as follows:

1. Select the project JanesTeamProject.
2. In the Navigator view select **Team** > **Tag as Version...**
3. When the Tag Resources dialog opens enter the version name A1 and click **OK**.
4. Select the CVS Repositories view.
5. Expand **Versions**, and JanesTeamProject. Observe that under JaneTeamProject there is now a version of this project whose version number is A1. Looking closely at the contents of folder1 you can also see that the version numbers of the individual files match the version numbers in our Workbench.


CVS Repositories
JanesTeamProject
folder1
file1.txt 1.2
file2.txt 1.2
file3.txt 1.2
.project 1.2
Navigator
JanesTeamProject
folder1
file1.txt 1.2 (ASCII -kkv)
file2.txt 1.2 (ASCII -kkv)
file3.txt 1.2 (ASCII -kkv)
.project 1.2 (ASCII -kkv)

Ant & external tools tutorial

This chapter covers Eclipse's external tools framework, and its integration with Ant, the build tool from the Apache Software Foundation. If you are not familiar with Ant, please look the Apache Ant manual for documentation. This chapter assumes you have a basic knowledge of Ant.

This chapter is split into three main sections. First, we cover the basics of working with Ant buildfiles in Eclipse. Next, we work through several Eclipse use cases in which Ant buildfiles can make life a little easier. Finally, we consider the entire external tools framework and how to use non−Ant tools.

Eclipse Ant basics

This section covers the basics of working with Ant buildfiles in Eclipse. Creating, editing, configuring and reusing Ant buildfiles will all be discussed in this section.

Creating Ant buildfiles

Ant buildfiles are just text files, so the most straightforward way to create an Ant buildfile in Eclipse is:

1. **File** > **New** > **File**.
2. Enter any name for the file, but make sure it has a .xml extension.
3. Click **Finish**.

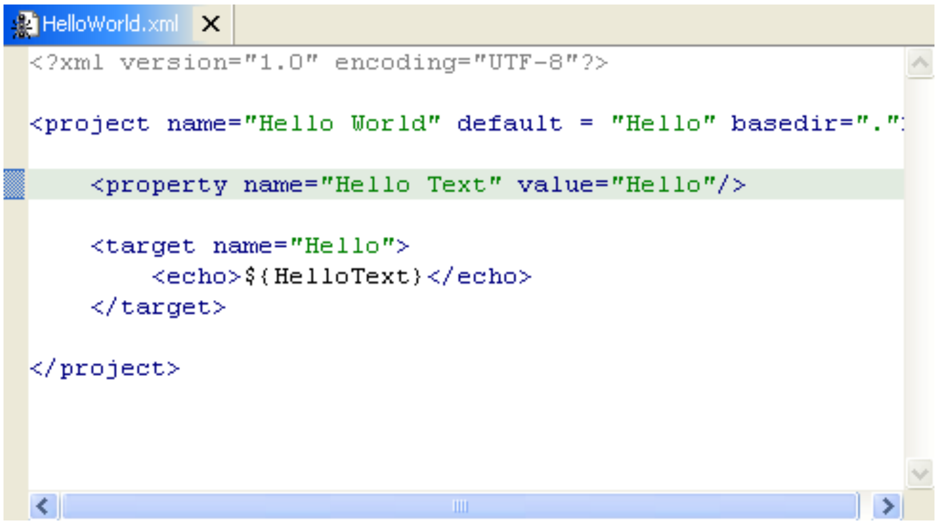
As long as the file has a .xml extension, Eclipse will consider it to be a possible Ant buildfile, and will enable Ant−related actions when it is selected.

As we will see in a later section, you can create an Ant buildfile for an Eclipse plug−in that contains predefined targets related useful for deploying the plug−in.

Editing Ant buildfiles

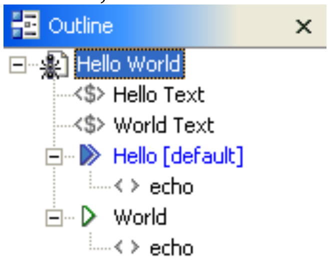
Because Ant buildfiles are simple text files, you can use any text editor to edit them. But there are several advantages to using the Eclipse Ant editor, including syntax coloring, content assist and an outline view. To get familiar with the Eclipse Ant editor, we will create a simple Ant buildfile using this editor.

1. Create an Ant buildfile called HelloWorld.xml (see the previous page if necessary).
2. Open the Ant editor on the file by selecting **Open With>Ant Editor** from the file's context menu. Note that the default editor for a .xml file is a simple text editor, but this can be changed in the **Window** > **Preferences** > **Workbench** > **File Associations**.
3. Enter the following content in the editor.

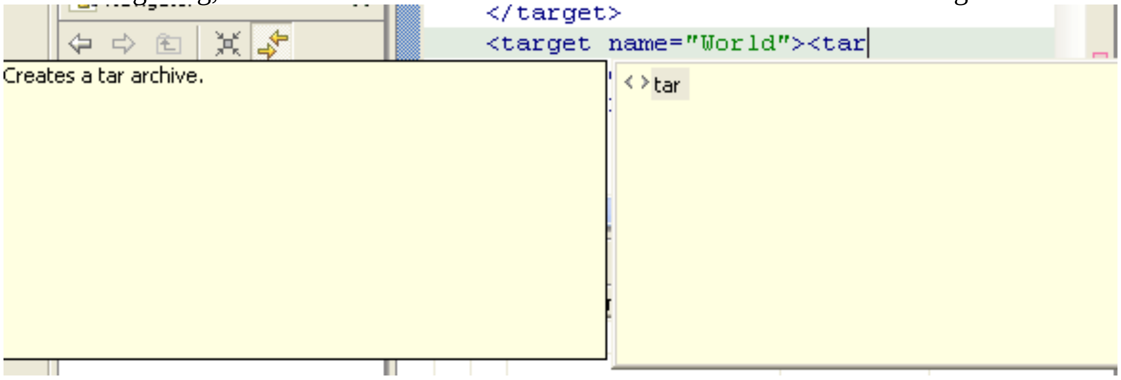


```
<?xml version="1.0" encoding="UTF-8"?>

<project name="Hello World" default = "Hello" basedir=".":

    <property name="Hello Text" value="Hello"/>

    <target name="Hello">
        <echo>${HelloText}</echo>
    </target>

</project>
```

4. Notice the syntax coloring for property values.
5. Begin to enter a second target by typing '<tar', then hit Ctrl−Space to activate content assist. You are presented with a list of valid completions. In this case there is only one, the <target> tag. Select this completion and notice that the editor inserts both the opening and closing tags and leaves the cursor positioned for you to enter properties for this tag.
6. Name this target 'World'.
7. Enter an 'echo' task for this target similar to that for the Hello target, but change the text to 'World'.
8. Save the changes to HelloWorld.xml.
9. Make the Outline view visible and notice that there are entries for each property and each target. In addition, each task under a target has an entry.



10. Clicking on an entry in the Outline view will scroll the editor to that entry. In addition, when the Outline view has focus, typing a character will move the selection in the Outline view to the next visible entry beginning with that character.
11. Position the cursor just past the end of one of the '<target>' tags, type '<tar', then hit Ctrl−Space to activate content assist. Notice that now the only valid completion is the 'tar' tag. This is because the Ant editor knows that nested targets are not allowed. Previously, when we used content assist to create a target tag, the editor knew that a tar task was not allowed outside of a target.



```
</target>
<target name="World"><tar
```

Creates a tar archive.

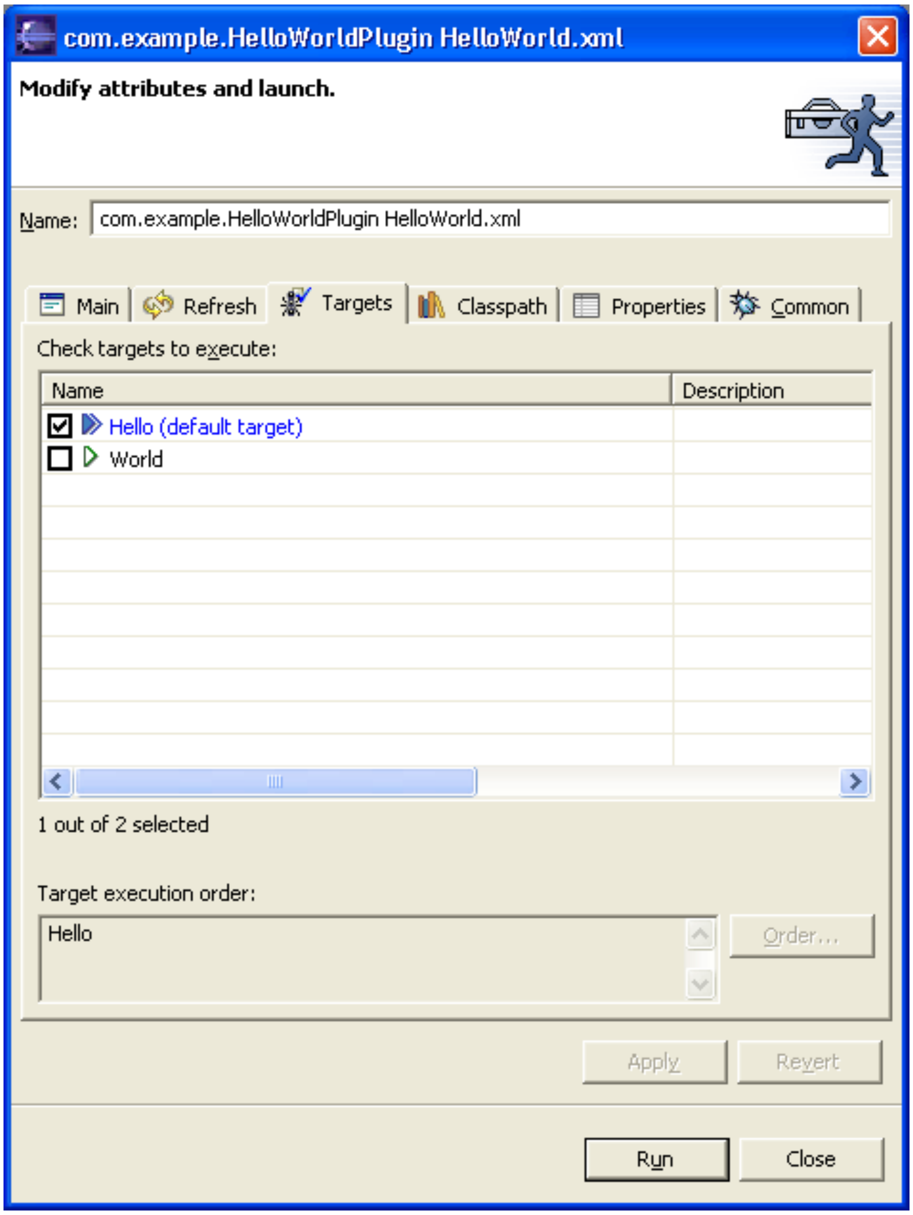
12. Close the editor and do not save changes.

Running Ant buildfiles

Any file with a .xml extension can be run as an Ant buildfile. Of course, not all such files really are Ant buildfiles, but no harm is done if you mistakenly attempt to run a non−Ant .xml file as an Ant buildfile.

In this section, we will cover the basic mechanisms for running Ant buildfiles in Eclipse, using the HelloWorld.xml file created in the last section.

1. Select HelloWorld.xml in the Navigator and choose **Run Ant...** from its context menu.
2. The Run Ant dialog appears.


com.example.HelloWorldPlugin HelloWorld.xml
Modify attributes and launch.
Main
Refresh
Targets
Classpath
Properties
Common
Order...
Apply
Revert
Run
Close

Name: com.example.HelloWorldPlugin HelloWorld.xml

Check targets to execute:

Name	Description
☑ Hello (default target)	
☐ World	

1 out of 2 selected

Target execution order:

Hello

3. This dialog allows you to configure many aspects of the way your Ant buildfile is run, but for now we will concentrate on the **Targets** tab which allows you to select the Ant targets to run and their order.

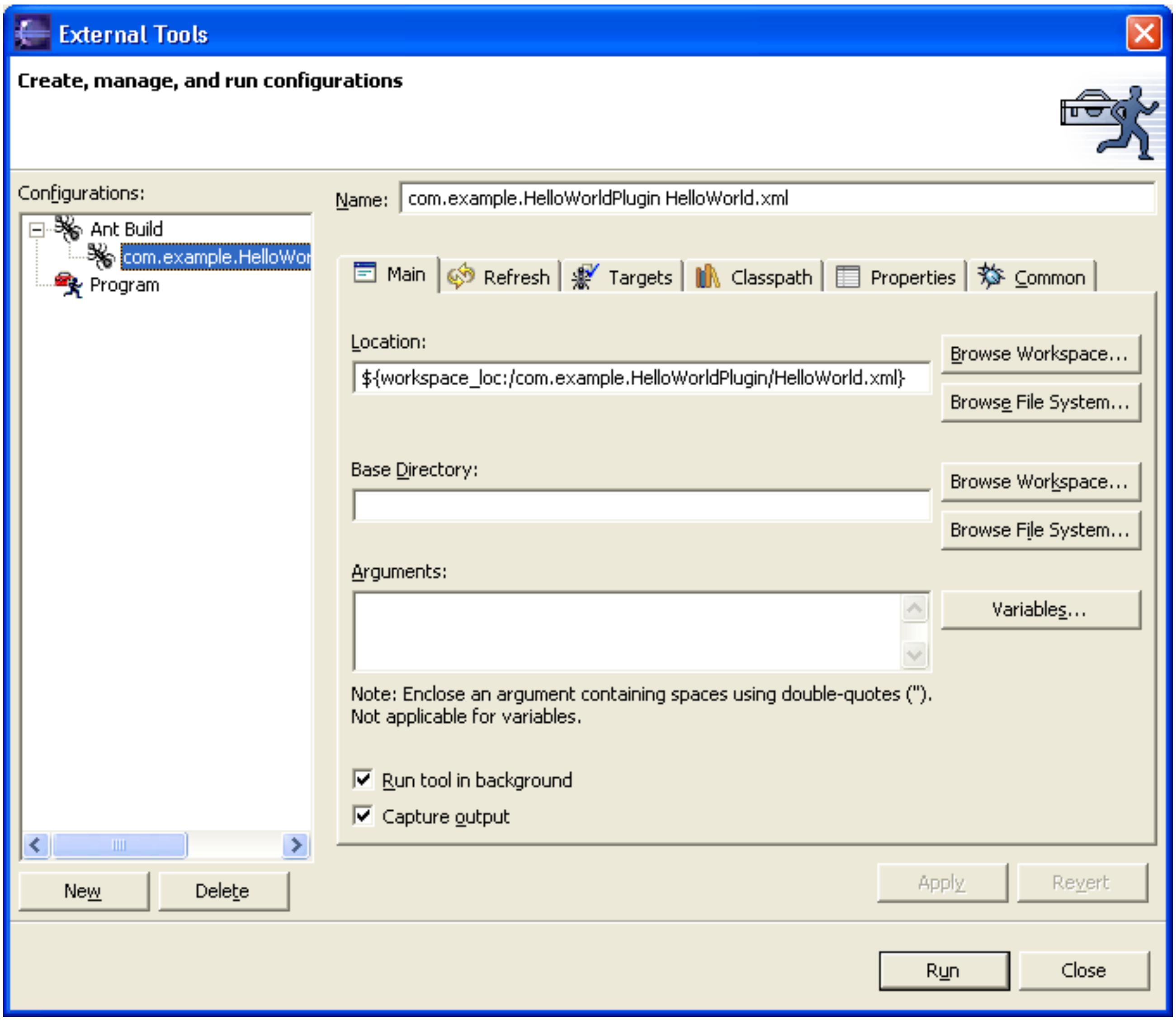
Select both targets and leave the order as the default.
4. Click **Run**.
5. The Ant buildfile is run, and the output is sent to the Console view.

But what if you sometimes want to run the buildfile so the Hello target executed first, and other times so the World target was first? Would you have to bring up the dialog each time and change the ordering? Fortunately, as we'll see in the next section, the answer is no.

Saving & Reusing Ant options

When we ran the HelloWorld.xml Ant buildfile in the last section, the choice of targets, along with all other options in the Run Ant dialog were saved in an entity called a 'launch configuration'. Launch configurations contain all details necessary to run a single Ant buildfile in a particular way. It is perfectly valid to have multiple launch configurations associated with a single Ant buildfile. So, in addition to the launch configuration that was created in the last step, specifying that our HelloWorld.xml buildfile should execute the targets Hello & World in that order, we could create a second launch configuration for this same buildfile specifying the same targets but in the reverse order. So far so good. But the really nice thing about launch configurations is that now you can quickly run your Ant buildfile in either configuration by simply specifying the corresponding launch configuration.

1. From the External Tools drop down on the Workbench toolbar, select **External Tools...**.
2. The External Tools dialog opens. This should look a lot like the Run Ant dialog we saw previously. In fact, it's identical except that in the External Tools dialog, you have a choice of which launch configuration you wish to view and edit. The launch configuration we created when we ran the build.xml buildfile is selected in the tree at the left, and the tabs on the right show the options for this launch configuration.



External Tools

Create, manage, and run configurations

Configurations:

- Ant Build
  - com.example.HelloWor
- Program

Name: com.example.HelloWorldPlugin HelloWorld.xml

Main | Refresh | Targets | Classpath | Properties | Common

Location:
${workspace_loc:/com.example.HelloWorldPlugin/HelloWorld.xml}

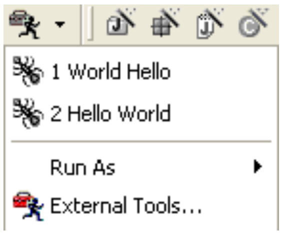
Base Directory:

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").
Not applicable for variables.

☑ Run tool in background
☑ Capture output

3. At the top of the dialog, change the Name to 'Hello World' and **Apply** the change.
4. In the tree at left, bring up on the context menu on the selected launch configuration and choose **Duplicate**. A copy of the launch configuration for the Hello World buildfile is created, '(1)' is appended to the name, and the new launch configuration is selected in the tree.
5. Rename the new configuration to 'World Hello'.
6. In the **Targets** tab, click the **Order...** button, change the order of the targets so that the World target executes first, and **Apply** the change.
7. Click **Run**.
8. As before, the HelloWorld.xml buildfile runs and sends its output to the Console view. This time however, because the targets were reversed, the output is reversed as well.
9. Go back to the External Tools drop down in the toolbar.

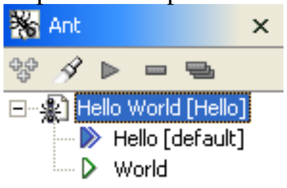


Notice now that there are two entries in the history, one for Hello World and one for World Hello. In order to rerun the Ant buildfile so that it outputs Hello World, just select this launch configuration in the history. To rerun the launch configuration that outputs World Hello, select this configuration in the history. Note that the history is ordered so that the most frequently run configurations are at the top of the dropdown.

This concludes our quick look at the basics of Ant integration in Eclipse. In the next chapters, we consider several real−world use cases for running Ant buildfiles inside Eclipse.

Using the Ant view

Eclipse provides a standard view, the Ant view, that lets you work with your Ant buildfiles. This view is tree−structured, showing Ant buildfiles as top−level entries and targets & internal targets as children. The main advantage of this view is that you can work with all of your Ant buildfiles in one place, as opposed to hunting them down in the Navigator.

1. Open the Ant view from the workbench menu by selecting **Window** > **Show View** > **Other...** > **Ant** > **Ant**.
2. By default, the Ant view is empty. There are two ways to add Ant buildfiles to this view:
    - Click the 'Add Buildfile' button. This brings up a dialog in which you explicitly select those Ant buildfiles you want to add
    - Click the 'Add Buildfiles with Search' button. This brings up a search dialog in which you can specify a filename pattern for your Ant buildfiles and search within the entire workspace or a specified working set
3. Once added to the Ant view, Ant buildfile entries remain in the view across workbench invocations until explicitly removed.
4. Click the Add Buildfiles with Search button. Suppose you only remember that the buildfile you want to work with starts with 'H'. Enter 'H*.xml' for the buildfile name. Make sure Workspace is selected for the scope, then click Search. The HelloWorld.xml file is found and placed in the Ant view.
5. Expand the top−level entry to see the default target Hello, and the internal target World.


6. Select the World internal target and click the Play button. Notice that just the World target gets executed.
7. Select the top−level HelloWorld buildfile and click Play. Notice that just the default target, Hello, gets executed.
8. To edit your buildfile, bring up the context menu on the HelloWorld file and select **Open With** > **Ant Editor**.
9. To edit the default launch configuration, select Properties... from the context menu.
10. The Run Ant dialog appears. Here you can modify the way in which the buildfile is run from the Ant view. Note that the choice and order of targets is ignored when running from the Ant view. If the top−level file is selected, just the default target is run and if one of the targets or internal targets is selected, just that target is run. It is not possible to run multiple targets from the Ant view.

11. Select the HelloWorld file, then click the Remove button. The buildfile is removed from the view. Note that this does not delete the file from the workspace.

Use cases for Ant in Eclipse

Now that we've seen how to work with Ant buildfiles in Eclipse, let's look at two ways that Ant buildfiles can be useful in Eclipse.

- Deploying Eclipse plug−ins
- Building projects

We'll start by using an Ant buildfile to handle various aspects of deploying Eclipse plug−ins.

Deploying Eclipse plug−ins

The first use of Ant within Eclipse discussed here will be as a plug−in deployment tool. When developing Eclipse plug−ins, it's frequently handy to build the jar files for your plug−in and test them as part of an Eclipse install. The Ant buildfile we'll look at in the next section will do this and much more.

Creating a HelloWorld plug−in

To work through this example, we need to create a simple 'Hello World' Eclipse plug−in. Creating Eclipse plug−ins is covered in more detail elsewhere in this documentation, so we will just create a simple HelloWorld plug−in as follows:

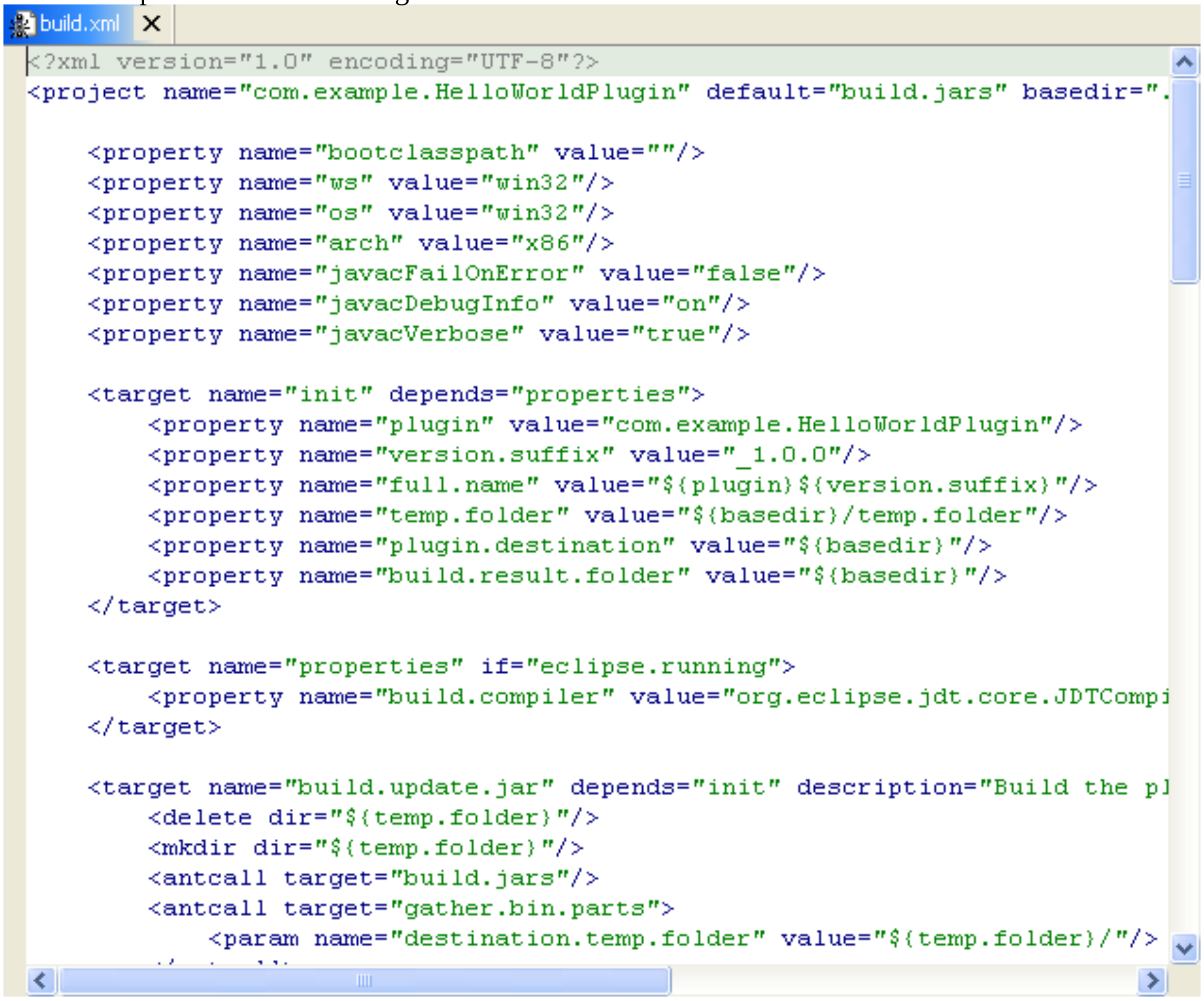
1. Create a new plug−in project by menuing to **File** > **New** > **Other...**
2. Select **Plug−in Development** in the left pane, and **Plug−in Project** in the right and click **Next**.
3. Enter a name for your project, then click **Next**.
4. On the next page, accept all defaults and click **Next**.
5. Make sure **Create a plug−in project using a code generation wizard** is selected, then select **Hello, World** from the list below and click **Next**.
6. Accept all the defaults on this page and click **Finish**.

Now that we have a plug−in project, we can create a build.xml file for it to manage plug−in deployment.

Generating the build.xml file

Open a Navigator view and locate your new plug−in project.

1. Select the file 'plugin.xml' underneath your new project and right−click **Create Ant Buildfile**.
2. A new file, 'build.xml' will be created underneath your project.
3. Open the Ant editor on the build.xml file by double clicking it in the Navigator. Notice that for this file, the Ant editor is the default editor. This is because by default, there is a file association in the workbench preferences associating files named 'build.xml' with the Ant editor.



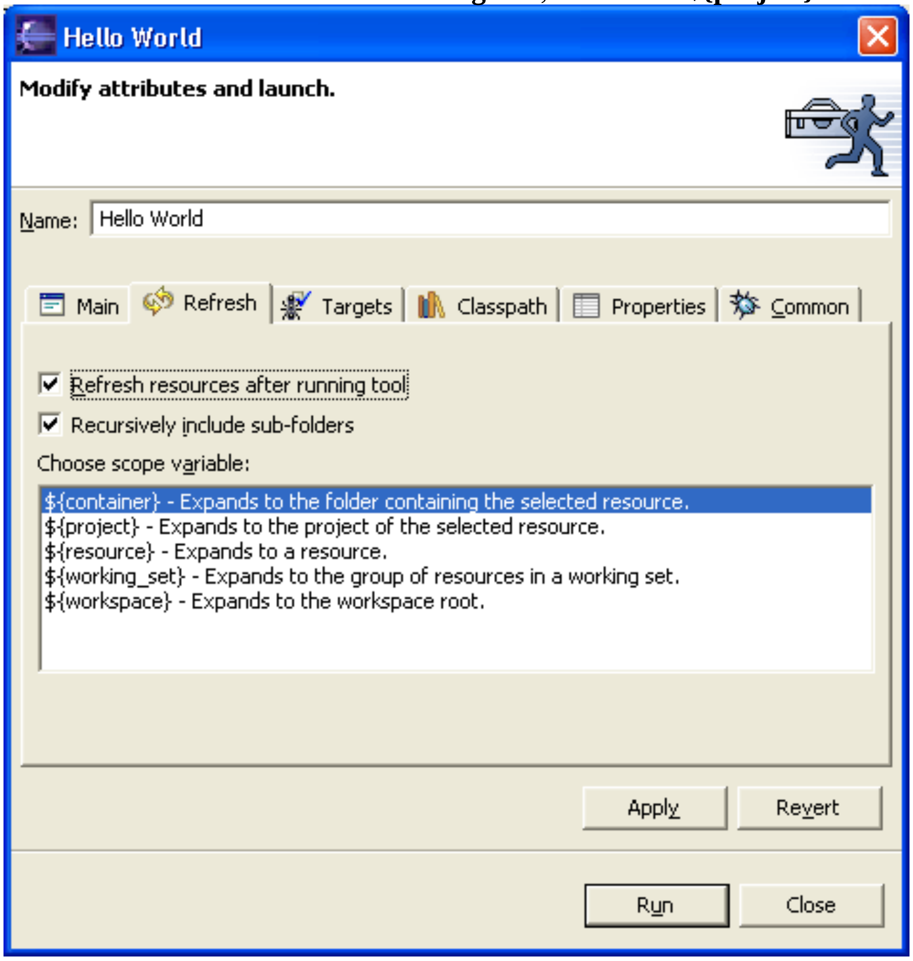
```
<?xml version="1.0" encoding="UTF-8"?>
<project name="com.example.HelloWorldPlugin" default="build.jars" basedir=".

    <property name="bootclasspath" value=""/>
    <property name="ws" value="win32"/>
    <property name="os" value="win32"/>
    <property name="arch" value="x86"/>
    <property name="javacFailOnError" value="false"/>
    <property name="javacDebugInfo" value="on"/>
    <property name="javacVerbose" value="true"/>

    <target name="init" depends="properties">
        <property name="plugin" value="com.example.HelloWorldPlugin"/>
        <property name="version.suffix" value="_1.0.0"/>
        <property name="full.name" value="${plugin}${version.suffix}"/>
        <property name="temp.folder" value="${basedir}/temp.folder"/>
        <property name="plugin.destination" value="${basedir}"/>
        <property name="build.result.folder" value="${basedir}"/>
    </target>

    <target name="properties" if="eclipse.running">
        <property name="build.compiler" value="org.eclipse.jdt.core.JDTCompi
    </target>

    <target name="build.update.jar" depends="init" description="Build the pl
        <delete dir="${temp.folder}"/>
        <mkdir dir="${temp.folder}"/>
        <antcall target="build.jars"/>
        <antcall target="gather.bin.parts">
            <param name="destination.temp.folder" value="${temp.folder}/"/>
```

The build.xml file is a default Ant buildfile with many targets useful for deploying your plug−in. For example, there are individual targets to build all necessary jars for your plug−in, create a .zip file containing everything in your plug−in, cleanup any .zip files that may have been created and so on.

Now that we have the build.xml file, we will use it to build a .jar file for our plug−in.

Building a .jar file for the plug−in

Now we are ready to use the build.xml Ant buildfile that was created in the last section:

1. Select the build.xml file in the Navigator and choose **Run Ant...** from the context menu.
2. In the Run Ant dialog, the Targets tab should be visible. Make sure that 'build.jars' is the only item in the list selected.
3. Switch to the **Refresh** tab. The options on this tab control if and how the workspace looks for new, changed and deleted resources after the buildfile finishes execution. Because refreshing can be an expensive operation, you should refresh the smallest subset of the workspace necessary for your buildfile.
4. Check **Refresh resources after running tool**, then select **${project}** in the list.

Hello World

Modify attributes and launch.

Name: Hello World

[✓] Refresh resources after running tool
[✓] Recursively include sub-folders

Choose scope variable:

${container} - Expands to the folder containing the selected resource.
${project} - Expands to the project of the selected resource.
${resource} - Expands to a resource.
${working_set} - Expands to the group of resources in a working set.
${workspace} - Expands to the workspace root.

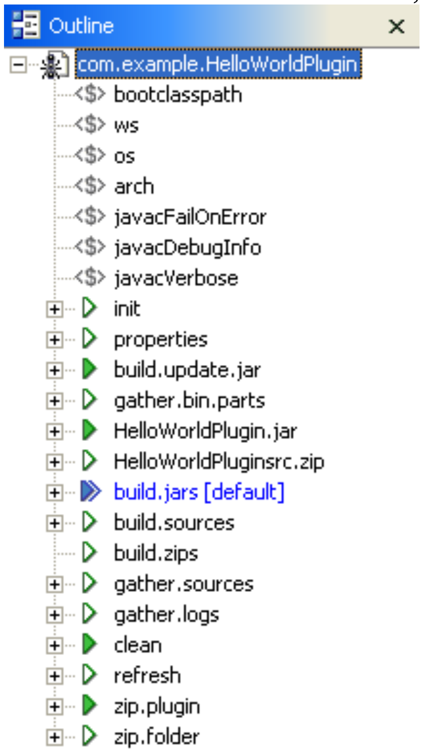
5. Click **Run**.
6. The Ant buildfile runs, executing the build.jars target. Output from the buildfile is visible in the Console view.
7. When the buildfile finishes, notice that there is a new .jar file under your project containing the two class files that comprise your plug−in. If we had chosen not to refresh the project, the .jar file would still have been created, but we would not see it in the Navigator.

We've run the default target for the build.xml Ant buildfile, but there are many other targets and options to be explored.

More plug−in deployment options

Now that we've run the default target for our build.xml buildfile, it's time to explore some more options.

As can be seen in the Outline view, the default build.xml Ant buildfile has a number of targets.



Solid green arrows denote targets, while internal targets are represented by hollow green arrows. The following are the useful top−level targets:

- **build.update.jar** This target builds jars for the plug−in, then zips them into a form suitable for use with the Eclipse Update Manager.
- **HelloWorldPlugin.jar** This target does the work of the building jars for the plug−in.
- **build.jars** This is the default target we ran in the last section. It defers to the HelloWorldPlugin.jar target to do its work.
- **clean** This target deletes all zips, jars and temporary directories that may have been created by other targets in this buildfile.
- **zip.plugin** This target builds executable and source jars, then zips everything into a single archive.

Remember that this build.xml file is simply the *default* deployment buildfile for an Eclipse plug−in. There will be times when you will want/have to modify this buildfile to handle your particular projects. For example, if you have a directory that contains resources necessary for your plug−in, you will want to update the build.xml file to include this directory in the jars and zips it creates.

This completes our look at using Ant buildfiles to deploy Eclipse plug−ins. The key points to remember are that you can easily create a default deployment buildfile for a plug−in that contains many useful targets, and that while this default buildfile is useful, you may have to modify it to suit your needs.

Ant buildfiles as project builders

Our second practical example of using Ant buildfiles in Eclipse is a 'project builder'. This is an Ant buildfile that has been designated to run whenever a project is built. The uses for such a buildfile are many:

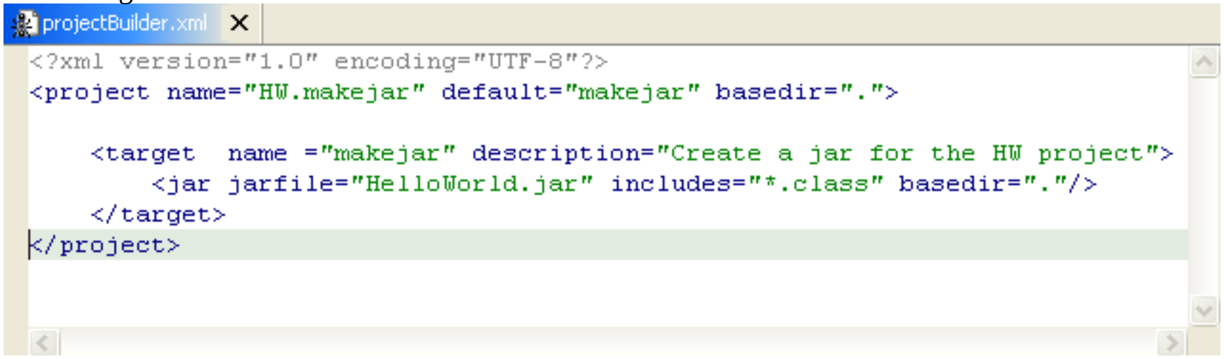
- Generate a .jar file containing class files from your project
- Perform some type of pre or post build processing on source or binary files in your project. For example:
    - Pre−processing source files to instrument them for performance analysis
    - Obfuscating binary files to prevent reverse engineering
- Copy class files to some location (for instance, on a network)

For this example, we will create an Ant buildfile that creates a .jar archive of the class files in a project. Note that this is similar to the example in the last chapter, in which we used an Ant buildfile to generate .jar files for an Eclipse plug−in. This example differs in that it works for any Eclipse project, whether or not it is also an Eclipse plug−in.

Creating a project builder Ant buildfile

To see how project builders work, we will create a simple project with a single source file and an Ant buildfile that jars up the single class file. Though this example uses Java, it should be noted that project builders are available for all projects, Java or otherwise.

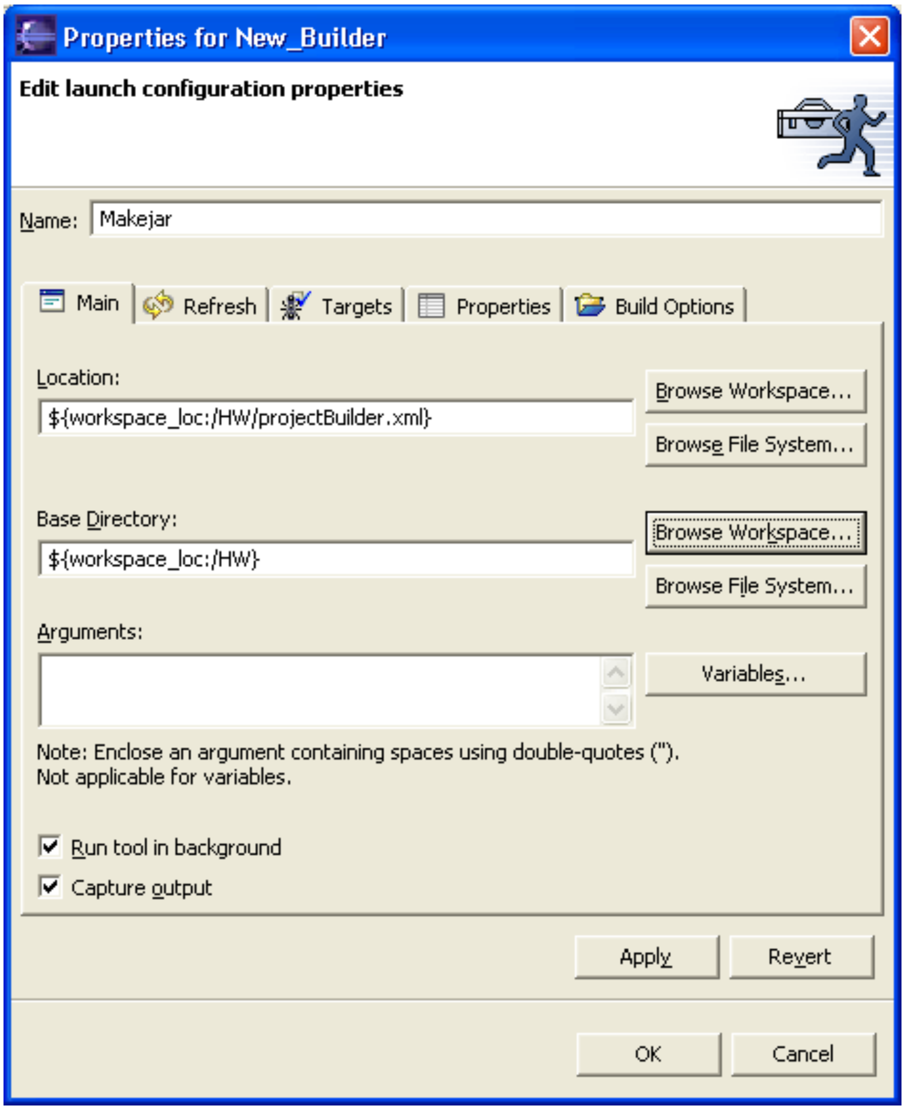
1. Create a Java project named 'HW'.
2. Create a Java source file named 'HelloWorld' with a main method.
3. Put a single 'System.out.println()' statement in the main method, and make it print a greeting of your choice.
4. Save changes.
5. Create a file named 'projectBuilder.xml', open the Ant editor on it, enter the following content, and save changes.



projectBuilder.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="HW.makejar" default="makejar" basedir=".">

    <target  name ="makejar" description="Create a jar for the HW project">
        <jar jarfile="HelloWorld.jar" includes="*.class" basedir="."/>
    </target>
</project>
```

6. In the Navigator, select the HW project and choose **Properties** from its context menu.
7. In the project properties dialog, select **External Tools Builders**, then click **New...**.
8. In the 'Choose configuration type' dialog, make sure 'Ant build' is selected, and click **OK**.
9. The External Tools dialog appears.Set the name to 'Makejar'. In the Main tab, use the first **Browse Workspace...** button to set the **Location** to be the projectBuilder.xml buildfile created above. Then use the second **Browse Workspace...** button to set the Base Directory to be the HW project.


Properties for New_Builder

Edit launch configuration properties

Name: Makejar

Location:
${workspace_loc:/HW/projectBuilder.xml}

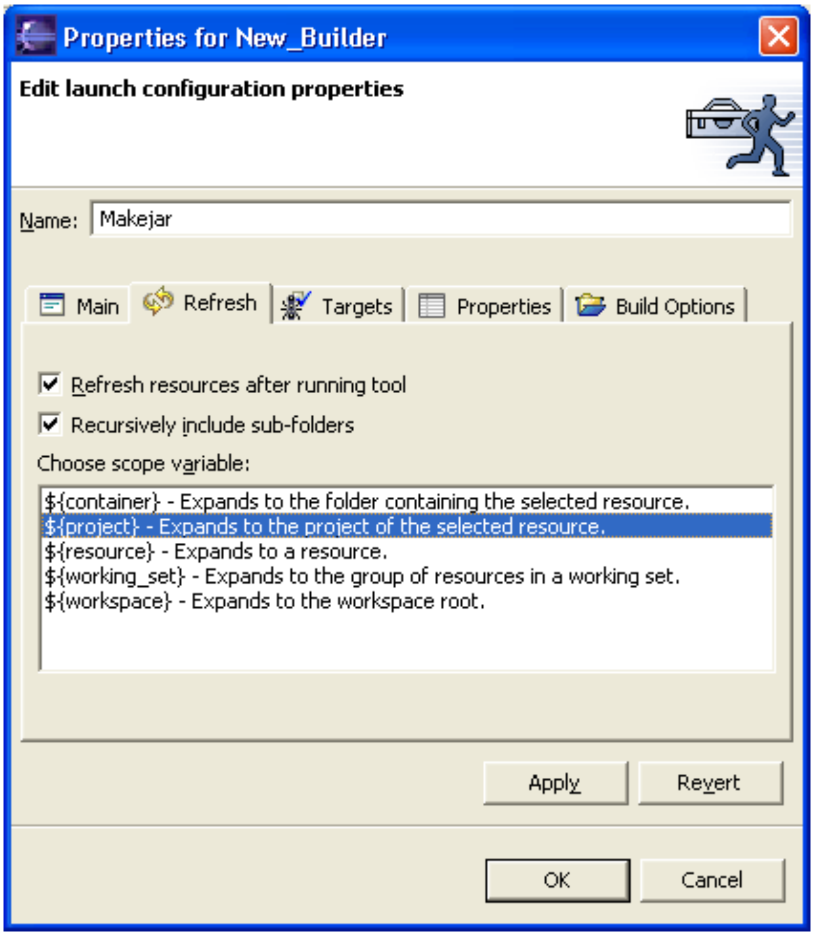
Base Directory:
${workspace_loc:/HW}

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").
Not applicable for variables.

☑ Run tool in background
☑ Capture output

10. In the **Refresh** tab, we want to be sure that when our HelloWorld.jar is created, we see it in Eclipse. By default, no refreshing is done when a project builder finishes running, so check **Refresh resource after running tool**, then select **${project}** in the list of scope variables. Because refreshing can be expensive, you should in general refresh the smallest entity that contains all resources that will be affected by your buildfile.


Properties for New_Builder

Edit launch configuration properties

Name: Makejar

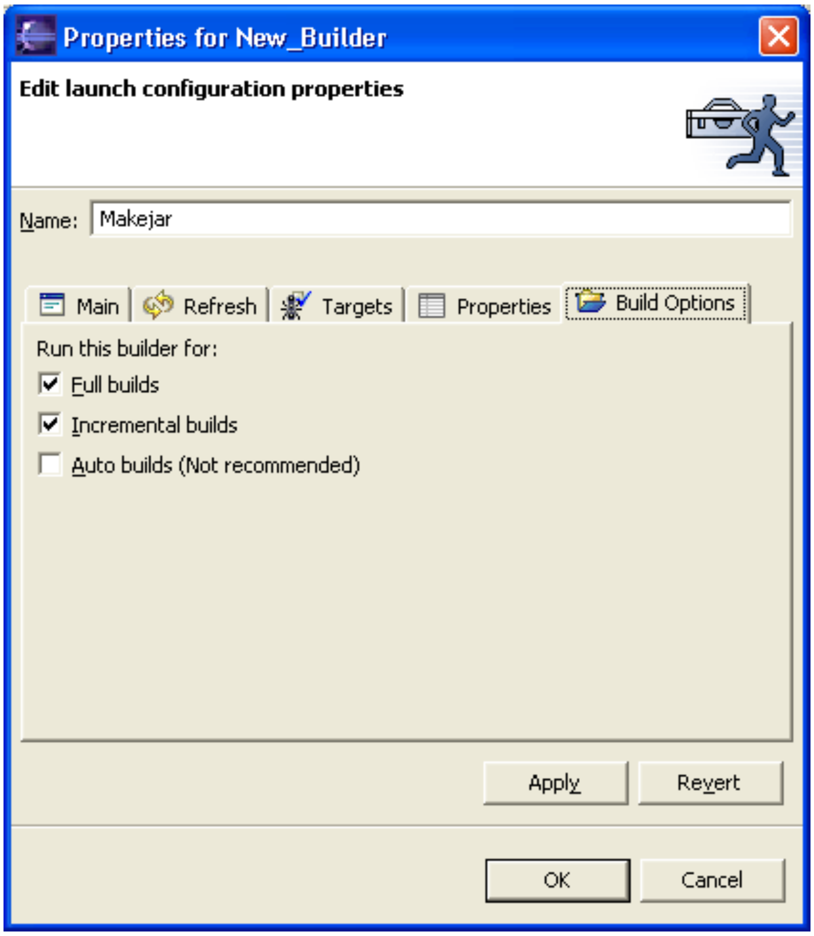
☑ Refresh resources after running tool

☑ Recursively include sub-folders

Choose scope variable:

```
${container} - Expands to the folder containing the selected resource.
${project} - Expands to the project of the selected resource.
${resource} - Expands to a resource.
${working_set} - Expands to the group of resources in a working set.
${workspace} - Expands to the workspace root.
```

11. In the **Targets** tab, the default target should be selected.
12. In the **Build Options** tab, you can specify when this project builder is executed. By default, this is set to full builds and incremental builds. Running your project builder during auto builds is possible, though not recommended because of performance concerns.


Properties for New_Builder
Edit launch configuration properties

Name: Makejar

Main
Refresh
Targets
Properties
Build Options

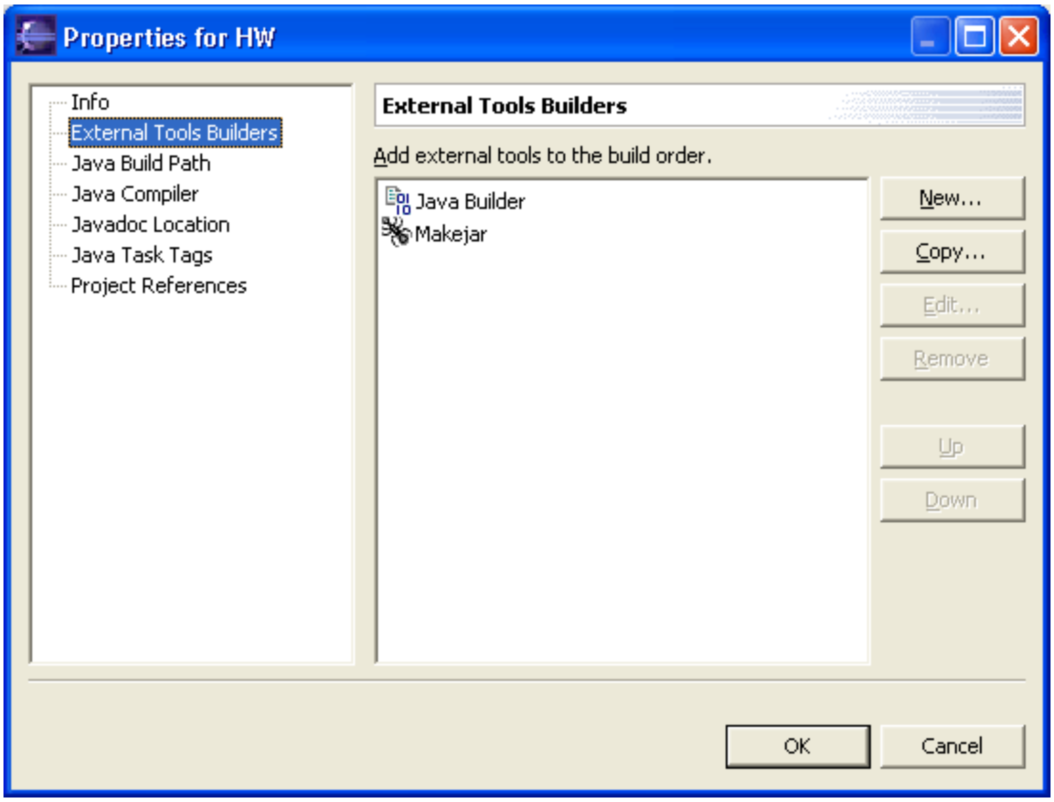
Run this builder for:

[✓] Full builds

[✓] Incremental builds

[ ] Auto builds (Not recommended)

13. Apply the changes and click **OK**.
14. Back in the project properties dialog, you will now see a project builder named 'Makejar' that is set to run after the default Java Builder. Note that you can change the order so that your Ant buildfile runs before the Java builder, though that wouldn't make sense in this example. Click **OK** to save the project builder and close the dialog.



For a Java project, the default Java Builder will always be run and cannot be removed. The Java Builder runs the internal Eclipse Java compiler which in turn is responsible for indexing your source so that searching, refactoring and many other features are available. Thus it is not possible to replace the internal Eclipse Java compiler by using a project builder. Your only option with the Java Builder is when it runs with respect to the project builders that you define.

Executing project builders

The whole point of project builders is that they are not explicitly run by the user, but instead are run any time a qualifying build takes place for the project that owns the buildfile. Remember that the types of builds that trigger project builders are defined in the External Tools dialog and can be any combination of full, incremental or auto builds. Let's see how this works.

1. Select the HW project in the Navigator. In the workbench menu bar, choose **Project > Rebuild Project**
2. The project is rebuilt and the projectBuilder.xml buildfile is run. Notice the output from this buildfile in the Console view.
3. Make sure the Autobuild preference is turned on, them make some trivial change to HelloWorld.java and save the change. The save triggers an auto build, but the auto build does not trigger the project builder.
4. Suppose we don't want to see the buildfile output every time it runs. Go back to the External Tools Builders page of the project properties dialog on HW. Select the Makejar entry and click **Edit...**. On the **Main** tab, uncheck the **Capture Output** option, apply the change and exit back to the workbench.

This concludes our look at Ant buildfiles as project builders in Eclipse. It's worth repeating that though this example used a Java project, project builders are not tied to Java, and may be used for any type of project.

External tools

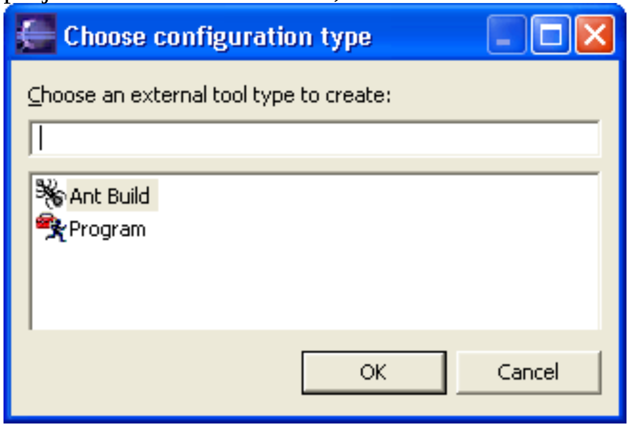
Up to this point in the tutorial, we've discussed Ant as if it were the only type of external tool available for use in Eclipse. The fact is that Eclipse's external tools framework handles two broad classes of external tools:

- Ant buildfiles
- Everything else

In this section, we look at ways to set up and use non−Ant external tools.

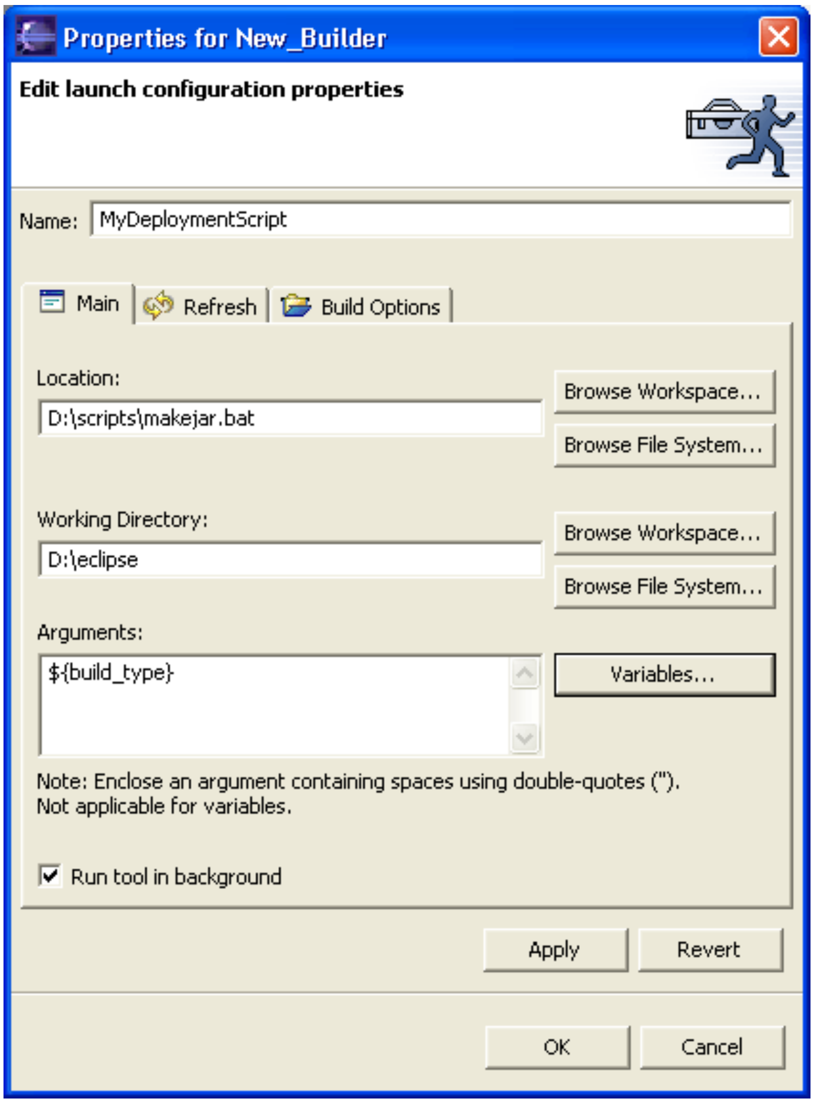
Non−Ant project builders

When we worked through our project builder example, you may have noticed that when we created our project builder Ant buildfile, we had a choice of external tool type:


Choose configuration type
Choose an external tool type to create:
Ant Build
Program
OK
Cancel

The **Program** option is essentially a catch−all, allowing you to define an external tool for any executable file that is accessible on your local or network file system. Suppose that instead of Ant, you prefer to use your own shell scripts or Windows .bat files to jar up and deploy your Eclipse projects. You would then create a Program external tool that specified where and how to execute your script.

1. Create a script that performs your preferred deployment steps.
2. Select the project that you wish to build in the Navigator, and choose **Properties** from the context menu.
3. Select **External Tools Builders**, click **New...**, select **Program** and click **OK**.
4. The External Tools dialog appears, configured for Program type tools.
5. Enter the location of your script, its working directory, and any required arguments.


Properties for New_Builder
Edit launch configuration properties

Name: MyDeploymentScript

Location: D:\scripts\makejar.bat

Working Directory: D:\eclipse

Arguments: ${build_type}

Note: Enclose an argument containing spaces using double-quotes (").
Not applicable for variables.

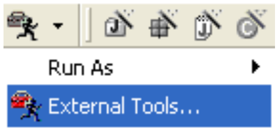
☑ Run tool in background

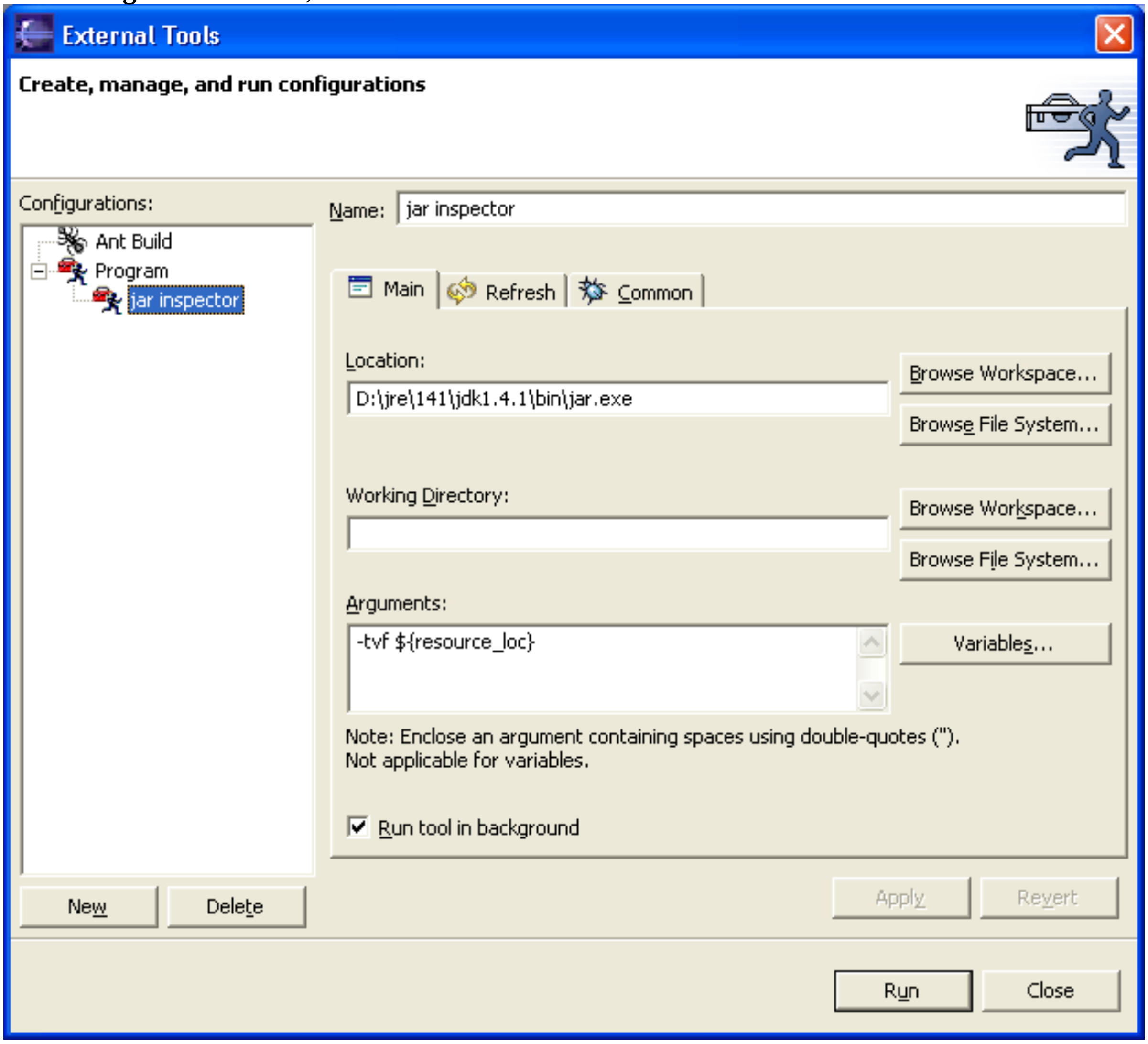
6. In this case, the script is a Windows .bat file, but it could be a Linux shell script, a Perl script or just about anything else that can be executed on your system.
7. The **Refresh** and **Build Options** tabs are identical to the tabs we saw for Ant project builders. In particular, the **Build Options** tab allows us to control what types of builds trigger our project builder buildfile.
8. Apply the changes, and click **OK**.
9. As with Ant project builders, we can control the ordering of this project builder with respect to other project builders (such as the default Java Builder for Java projects).
10. Rebuild your project. This will trigger your script to execute. Any output it generates will be sent to the Console view.

Ant is a popular tool for configuring and deploying projects. But if you prefer some other tool, or prefer to do it yourself, you can set up a Program external tool project builder. This allows you customize the deployment of your project as you see fit, while keeping the convenience of automatically running your script every time your project is built.

Stand−alone external tools

For the ultimate in external tool flexibility, create a 'stand−alone' external tool launch configuration. This is similar to the project builder launch configurations discussed in the last section, except that it need have nothing to do with project building, and you can explicitly run it whenever you choose. Suppose you wanted to have a way to quickly see the contents of a .jar file in your workspace using the jar utility.

1. Select some .jar file in your workspace.
2. Select **External Tools...** from the External Tools workbench toolbar drop down.


3. Select **Program** in the tree, then click **New**.

External Tools

Create, manage, and run configurations

Configurations:
- Ant Build
- Program
  - jar inspector

Name: jar inspector

Main | Refresh | Common

Location:
D:\jre\141\jdk1.4.1\bin\jar.exe

Working Directory:

Arguments:
-tvf ${resource_loc}

Note: Enclose an argument containing spaces using double-quotes (").
Not applicable for variables.

☑ Run tool in background

4. Name the launch configuration 'jar inspector'.
5. Use the first **Browse File System...** button to locate the jar executable.
6. In the **Arguments** field, type '−tvf' and a space, then click **Variables...**.
7. In the Select Variable dialog, you will see a number of variables you can pass as arguments to the program specified in Location. Select **${resource_loc}** and click **OK**.

8. When this buildfile is run, the absolute path of the resource selected in the workbench will be passed to the jar utility in the position specified.
9. Click **Run**.
10. Notice that the buildfile sends the jar utility output to the Console view.
11. Select a different .jar file in your workspace.
12. Click the External Tools button in the toolbar. Notice the contents of this jar are sent to the Console view as well. Now you have a quick and easy way to see the output of the jar utility for any .jar file in your workspace.

This example has only scratched the surface of what you can do with external tools. The important things to remember are that you can create an external tool for anything you can run on your system, and that you can pass arguments to the external tool related to the current workbench selection. In many cases, this allows you to loosely integrate tools that do not have corresponding Eclipse plug−ins.