The Workbench

OTI Employee

Table of Contents

Table of Contents

Table of Contents

Workbench

The term *Workbench* refers to the desktop development environment. The Workbench aims to achieve seamless tool integration and controlled openness by providing a common paradigm for the creation, management, and navigation of Workbench resources.

Each Workbench window contains one or more perspectives. Perspectives contain views and editors and control what appears in certain menus and tool bars. More than one Workbench window can exist on the desktop at any given time.

Related concepts

Features
Resources
Perspectives
Views
Editors
CVS Repositories

Related tasks

Installing new features with the update manager
Opening perspectives
Opening views
Switching between perspectives
Configuring perspectives

Related reference

Toolbar buttons

Resources

Resources is a collective term for the projects, folders, and files that exist in the Workbench. The Navigator view provides a hierarchical view of resources and allows you to open them for editing. Other tools may display and handle these resources differently.

There are three basic types of resources that exist in the Workbench:

Files
: Comparable to files as you see them in the file system.

Folders
: Comparable to directories on a file system. In the Workbench, folders are contained in projects or other folders. Folders can contain files and other folders.

Projects
: Contain folders and files. Projects are used for builds, version management, sharing, and resource organization. Like folders, projects map to directories in the file system. (When you create a project, you specify a location for it in the file system.)
A project is either open or closed. When a project is closed, it cannot be changed in the Workbench. The resources of a closed project will not appear in the Workbench, but the resources still reside on the local file system. Closed projects require less memory. Since they are not examined during builds, closing a project can improve build time.
When a project is open, the structure of the project can be changed and you will see the contents.

Folders and files directly below projects can be linked to locations in the file system outside of the project's location. These special folders and files are called linked resources.

Different tools that plug into the Workbench use their own specialized types of projects, folders, and files.

Related concepts

Workbench
Navigator view
Resource hierarchies
Linked resources
Builds

Related tasks

Importing resources into the Workbench
Creating a project
Creating a folder
Creating a file
Creating linked resources
Closing projects
Viewing resource properties
Finding a resource quickly
Copying resources
Moving resources
Comparing resources

Resource hierarchies

Resources are stored and displayed in the Workbench in hierarchies. Described below are the terms used when referring to resources that are stored and displayed in a hierarchical structure.

Root
: The top level of the Workbench contents (in the file system).

Parent resource
: Any resource that contains another resource. Only projects and folders can be parent resources.

Child resource
: Any resource that is contained within another resource. Only files and folders can be child resources.

Resource hierarchies are displayed in the Navigator view, which is one of the default views in the Resource perspective.

Related concepts

Resources
Navigator view

Related tasks

Finding a resource quickly
Narrowing the scope of the Navigator view
Sorting resources in the Navigator view
Showing or hiding files in the Navigator view

Linked resources

Linked resources are files and folders that are stored in locations in the file system outside of the project's location. These special resources must have a project as their parent resource. Linked folders and files can be used to add resources to your project that for some reason must be stored in a certain place outside of your project. For example, a linked folder can be used to store build output separately from your source files.

You can even use linked resources to overlap other resources in the workspace, so resources from one project can appear in another project. If you do want to have overlapping resources in your workspace, do so with caution. Keep in mind that this means changing a resource in one place will cause simultaneous changes in the duplicate resource. Deleting one duplicate resource will delete both!

Special rules apply when manipulating linked resources. Since they must be located directly below a project, you cannot copy or move linked resources into other folders. Deleting a linked resource will *not* cause the corresponding resource in the file system to be deleted. However, deleting child resources of linked folders *will* cause them to be removed from the file system.

Some plug−ins built on top of the Eclipse platform are not compatible with linked resources. If this is the case, you can completely disable the linked resource feature to prevent them from being created in your workspace. Linked resources can be disabled from the **Window > Preferences > Workbench > Linked Resources** preference page. Certain types of projects or team repository providers may also disallow linked resources from being created in some projects.

Related concepts

Workbench
Navigator view
Resources
Resource hierarchies

Related tasks

Creating linked resources
Deleting resources
Viewing resource properties

Related reference

Linked resources

Path variables

Path variables specify locations on the file system. The location of linked resources may be specified relative to these path variables. They allow you to avoid references to a fixed location on your file system.

By using a path variable, you can share projects containing linked resources with team members without requiring the exact same directory structure as on your file system.

You can load a project that uses path variables even if you do not currently have all the path variables defined in your workspace. A linked resource that uses a missing path variable is flagged using a special decorator icon. In addition, the **File > Properties > Info** property page and the **Window > Show View > Other > Basic > Properties** view for a linked resource indicate the variable and whether it is defined or not. A path variable can also specify a location that does not currently exist on the file system. Linked resources that use such a path variable are indicated using the same decorator icon mentioned above.

You can create new path variables and edit and remove existing path variables on the **Window > Preferences > Workbench > Linked Resources** preference page.

Related concepts

Linked resources

Related tasks

Creating linked resources
Viewing resource properties

Related reference

Linked resources

Working sets

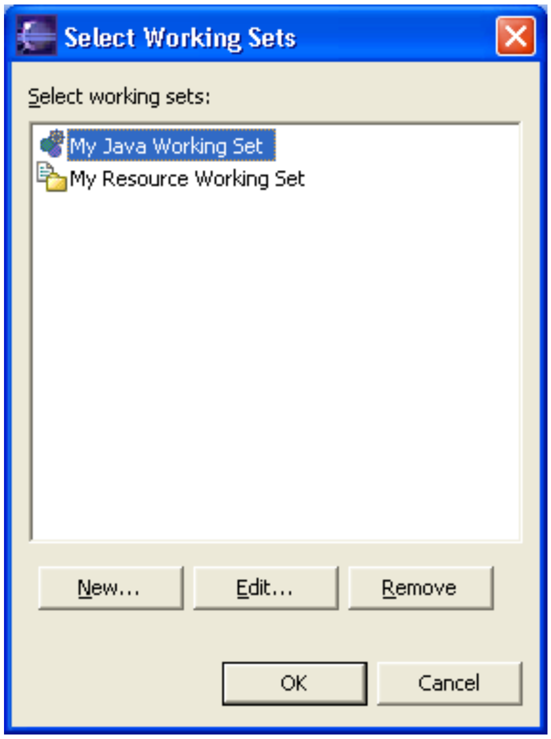
Working sets group elements for display in views or for operations on a set of elements.

The navigator view uses working sets to restrict the set of resources that are displayed. If a working set is selected in the navigator, only resources, children of resources, and parents of resources contained in the working set are shown.

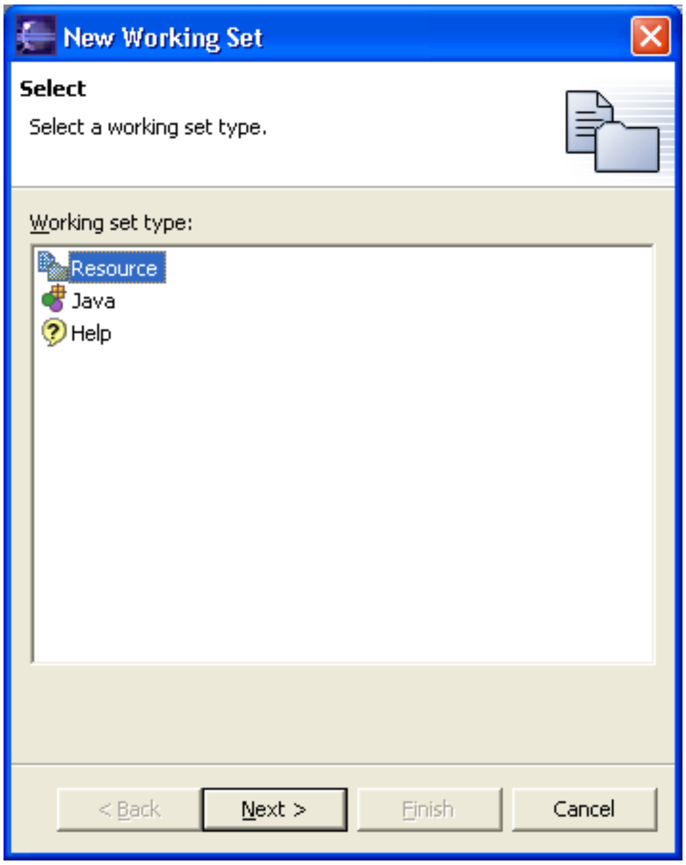
Working sets may be used in the task view to limit the display of tasks in a similar fashion as the navigator view.

When using the Workbench search facility, working sets may be used to restrict the set of elements that are searched.

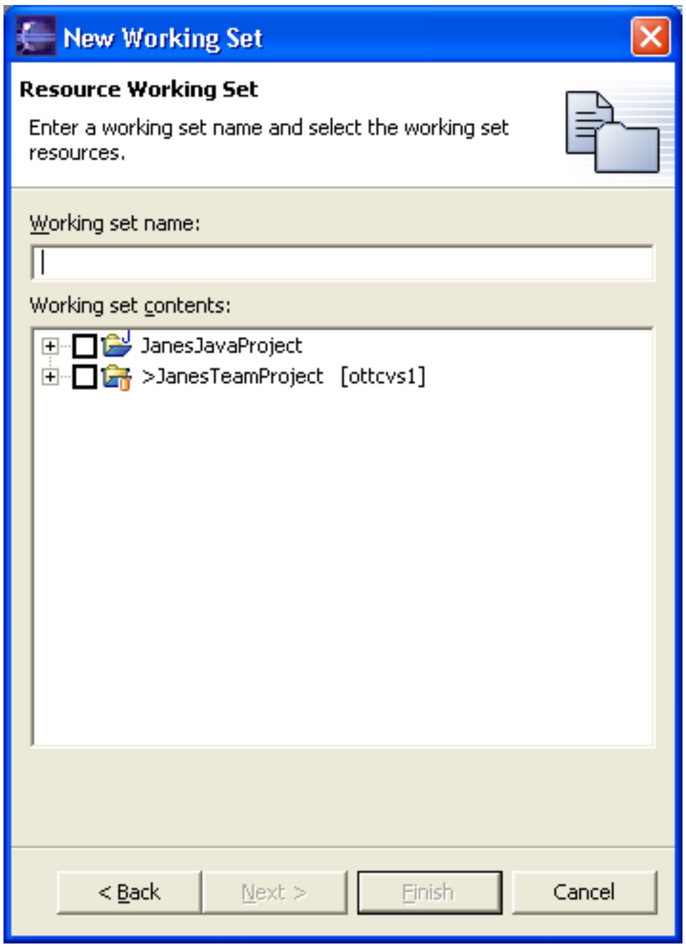
Different views provide different means to specify a working set. However, they typically use the following working set selection dialog to manage existing working sets and to create new working sets.



When you create a new working set you may be able to choose from different types of working sets. In the example below you can create a resource working set, a Java working set or a help working set.



If you create a new resource working set you will be able to select the working set resources as shown below. The same wizard is used to edit an existing working set. Different types of working sets provide different kinds of working set editing wizards.



Note: Newly created resources are not automatically included in the active working set. They are implicitly included in a working set if they are children of an existing working set element. If you want to include other resources after you have created them you have to explicitly add them to the working set.

Related concepts

Navigator view
Tasks view

Related tasks

Searching for text within a file
Showing or hiding resources in the Navigator view

Builds

A *build* is a process that derives new resources from existing ones, updates existing resources, or both.

In the Workbench, different *builders* are invoked for different types of projects. For example, when a build is triggered for a Java project, a Java builder converts each Java source file (.java files) into one or more executable class files (.class files). Builders usually enforce the constraints of some domain. For example, a Web link builder could update links to files whose name/location changes.

There are two kinds of builds:

- An *incremental build* leverages a previously built state and applies the transforms of the configured builders to the resources that have changed since the previous state was computed (that is, since the last build).
- A full build (or *rebuild*) discards any previously built state and transforms all requested resources according to the domain rules of the configured builders. The first incremental build is equivalent to a full build as there is no previous state to work from.

Full and incremental builds can be done over a specific set of projects or the workspace as a whole. Specific files and folders cannot be built. There are two ways that builds can be performed:

- Automatic builds are performed as resources are saved. Automatic builds are always incremental and always operate over the entire workspace. You can configure your Workbench preferences (**Window > Preferences > Workbench**) to perform builds automatically on resource modification.
- Manual builds are initiated when you explicitly select a menu item or press the equivalent shortcut key. Manual builds can be either full or incremental and can operate over collections of projects or the entire workspace.

Related concepts

External tools

Related tasks

Building resources
Performing builds manually
Performing builds automatically
Saving resources automatically before a manual build
Changing build order

Local history

A local edit history of a file is maintained when you create or modify a file. Each time you edit and save the file, a copy is saved so that you can replace the current file with a previous edit or even restore a deleted file. You can also compare the contents of all the local edits. Each edit in the local history is uniquely represented by the date and time the file was saved.

Only files have local history; projects and folders do not.

Related concepts

Versions

Related tasks

Comparing resources with local history
Replacing a resource with local history
Restoring deleted resources from local history
Setting local history preferences

Creating a version of a project

Perspectives

Each Workbench window contains one or more perspectives. A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of capabilities aimed at accomplishing a specific type of task or works with specific types of resources. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs. As you work in the Workbench, you will probably switch perspectives frequently.

Perspectives control what appears in certain menus and toolbars. They 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.

You can set your Workbench preferences to open perspectives in the same window or in a new window.

Related concepts

Workbench
Editors
Views

Related tasks

Opening perspectives
Opening views
Changing where perspectives open
Specifying the default perspective
Switching between perspectives
Configuring perspectives
Saving a user defined perspective
Resetting perspectives

Editors

Most perspectives in the Workbench are comprised of an editor area and one or more views.

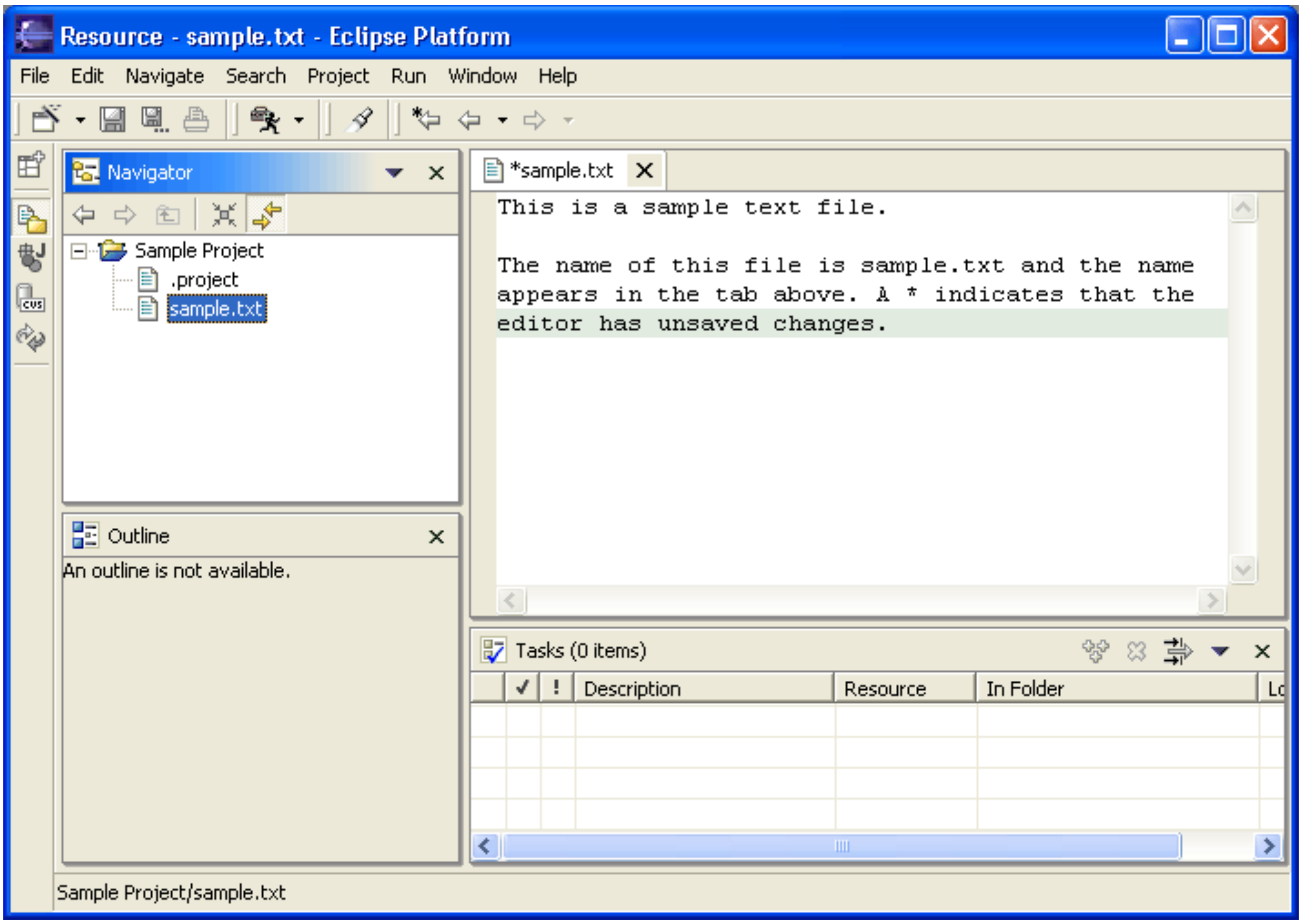
You can associate different editors with different types of files. For example, when you open a file for editing by double−clicking it in the Navigator view, the associated editor opens in the Workbench. If there is no associated editor for a resource, the Workbench attempts to launch an *external editor* outside the Workbench. (On Windows, the Workbench will first attempt to launch the editor in place as an OLE document. This type of editor is referred to as an *embedded editor*. For example, if you have a .doc file in the Workbench and Microsoft Word is registered as the editor for .doc files in your operating system, then opening the file will launch Word as an OLE document within the Workbench editor area. The Workbench menu bar and toolbar will be updated with options for Microsoft Word.)

Any number of editors can be open at once, but only one can be active at a time. The main menu bar and toolbar for the Workbench window contain operations that are applicable to the active editor.

Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes.

By default, editors are stacked in the editor area, but you can choose to tile them in order to view source files simultaneously.

Here is an example of a text editor in the Workbench:


Resource - sample.txt - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Sample Project
.project
sample.txt
Outline
An outline is not available.
*sample.txt
Tasks (0 items)
✓ ! Description Resource In Folder Lo
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.
```

The gray border at the left margin of the editor area may contain icons that flag errors, warnings, or problems detected by the system. Icons also appear if you have created bookmarks, added breakpoints for debugging, or recorded notes in the Tasks view. You can view details for any icons in the left margin of the editor by moving the mouse cursor over them.

Related concepts

Workbench
External editors
Bookmarks
Tasks view
Navigator view

Related tasks

Opening files for editing
Associating editors with file types
Editing files outside the Workbench

External editors

Sometimes you may need to use an external program to edit a file in the Workbench. This can occur, for example, when the Workbench has no editor for that file type.

The external program will be used if that program is registered as the system default editor for that file type and no other editor is registered for that file type in the Workbench. For example, on most systems the default "editor" for HTML files is a Web browser. If there is no other editor associated with .htm and .html files in the Workbench, then opening an HTML file from the Workbench would cause the file to be opened in a separate browser session.

Related concepts

Editors

Related tasks

Opening files for editing
Associating editors with file types
Editing files outside the Workbench

Views

Views support editors and provide alternative presentations as well as ways to navigate the information in your Workbench. For example, the Navigator view displays projects and other resources that you are working with.

Views also have their own menus. To open the menu for a view, click the icon at the left end of the view's title bar. Some views also have their own toolbars. The actions represented by buttons on view toolbars only affect the items within that view.

A view might appear by itself, or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the Workbench window.

Related concepts

Perspectives
Fast views
Editors

Related tasks

Opening views
Moving and docking views
Creating fast views
Maximizing a view or editor
Saving a user defined perspective
Resetting perspectives

Fast views

Fast views are hidden views that can be quickly opened and closed. They work like other views except they do not take up space in your Workbench window.

Fast views are represented by toolbar buttons on the *shortcut bar*, which is the vertical toolbar at the left side of the Workbench window. When you click the toolbar button for a fast view, that view opens temporarily in the current perspective. As soon as you click outside that view, it is hidden again.

You can create a new fast view by dragging any open view to the shortcut bar or by selecting **Fast View** from the menu that opens when you click the icon at the left end of the view's title bar.

Related concepts

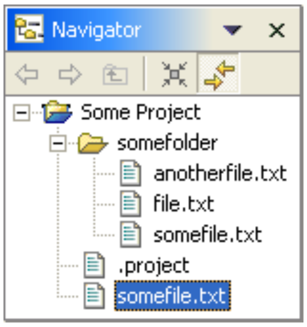
Workbench
Toolbars

Related tasks

Creating fast views
Moving and docking views
Maximizing a view or editor

Navigator view

The Navigator view provides a hierarchical view of the resources in the Workbench. From here, you can open files for editing or select resources for operations such as exporting.



Right−click on any resource in the Navigator view to open a pop−up menu that allows you to perform operations such as copying, moving, creating new resources, comparing resources with each other, or performing team operations. To see a description of what each menu item does, move selection highlight to that menu item and press F1.

By default, the Navigator view is included in the Resources perspective. To add it to the current perspective, click **Window** > **Show View** > **Navigator**.

Toolbar

The toolbar of the Navigator view contains the following buttons:

Back
: Displays the hierarchy that was displayed immediately prior to the current display.

Forward
: Displays the hierarchy that was displayed immediately after the current display.

Up
: Displays the hierarchy of the parent of the current highest level resource.

Collapse All
: Collapses the tree expansion state of all resources in the view.

Link with Editor
: Toggles whether the Navigator view selection is linked to the active editor. When this option is selected, changing the active editor will automatically update the Navigator selection to the resource being edited.

Menu
: Provides menu items that allow you to sort or filter the contents of the Navigator view as well as select a working set.

Icons

The following icons can appear in the Navigator view.

Icon	Description
📂	Project (open)

📁	Project (closed)
📂	Folder
📄	File

Related concepts

Resources
Views

Related tasks

Narrowing the scope of the Navigator view
Sorting resources in the Navigator view
Showing or hiding files in the Navigator view
Opening views
Moving and docking views
Creating fast views

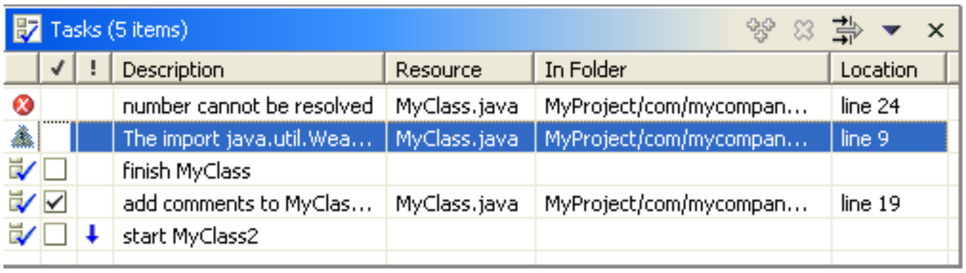
Related reference

Navigator view

Tasks view

As you work with resources in the Workbench, various builders may automatically log problems, errors, or warnings in the Tasks view. For example, when you save a Java source file that contains syntax errors, those will be logged in the Tasks view. When you double−click the icon for a problem, error, or warning, the editor for the associated resource automatically opens to the relevant line of code.

You can also add items to the Tasks view manually. For example, if you would like to record reminders to follow up on something later, add it to the tasks view. When you add a task manually, you have the option of associating it with a resource so that you can use the Tasks view to quickly open that resource for editing.



Tasks (5 items)

	✓	!	Description	Resource	In Folder	Location
✖			number cannot be resolved	MyClass.java	MyProject/com/mycompan...	line 24
⚠			The import java.util.Wea...	MyClass.java	MyProject/com/mycompan...	line 9
✓	☐		finish MyClass			
✓	☑		add comments to MyClas...	MyClass.java	MyProject/com/mycompan...	line 19
✓	☐	↓	start MyClass2			

The first column of the Tasks view displays an icon that denotes the type of line item. You can sort and filter line items in the task view, to view only high−priority tasks or only warnings and errors associated with a particular resource or group of resources.

By default, the Tasks view is included in the Resources perspective. To add it to the current perspective, click **Window** > **Show View** > **Tasks**.

Related concepts

Bookmarks

Related tasks

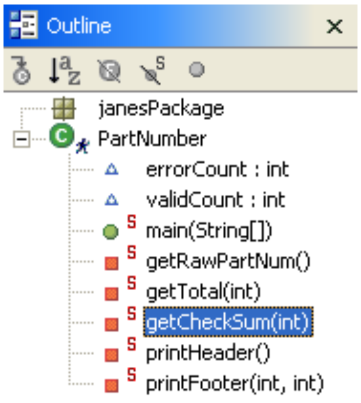
Opening views
Moving and docking views
Adding line items in the Tasks view
Associating a task with a resource
Filtering the Tasks view
Creating a bookmark for an entire file
Creating a bookmark within a file
Automatically fixing problems
Deleting tasks

Related reference

Tasks view

Outline view

The Outline view displays an outline of a structured file that is currently open in the editor area, and lists structural elements. The contents of the Outline view are editor specific. In the example below, which is for a Java source file, the structural elements are classes, fields, and methods. The contents of the toolbar are also editor specific.



Related concepts

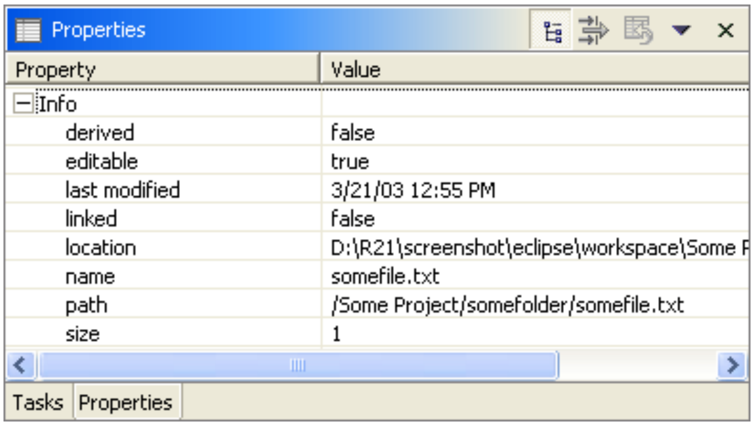
Views
Perspectives
Resources

Related tasks

Opening views
Moving and docking views
Creating fast views

Properties view

The properties view displays property names and values for a selected item such as a resource. Here is an example:



Property	Value
Info	
derived	false
editable	true
last modified	3/21/03 12:55 PM
linked	false
location	D:\R21\screenshot\eclipse\workspace\Some F
name	somefile.txt
path	/Some Project/somefolder/somefile.txt
size	1

Toolbar buttons allow you to toggle to display properties by category or to filter advanced properties. Another toolbar button allows you to restore the selected property to its default value.

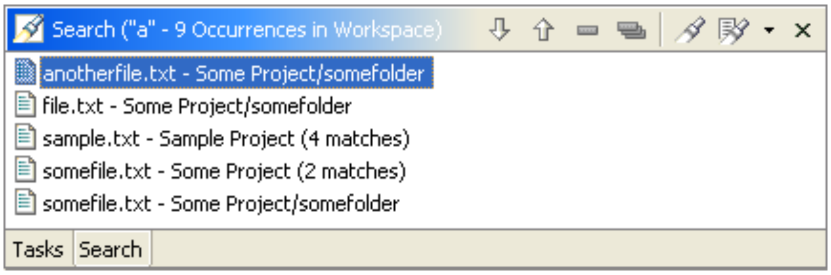
To see more detailed information about a resource than the Properties view gives you, right−click the resource name in the Navigator view and select Properties from the pop−up menu.

Search view

This view displays the results of a search.

Text searches will only search for expressions in files with extensions (file types) specified in the search dialog.

Here is what the Search view looks like:



Search ("a" - 9 Occurrences in Workspace)

anotherfile.txt - Some Project/somefolder
file.txt - Some Project/somefolder
sample.txt - Sample Project (4 matches)
somefile.txt - Some Project (2 matches)
somefile.txt - Some Project/somefolder

Toolbar

The toolbar in the Search view contains the following buttons:

Show Next Match
: This command highlights the next match of the search expression in the editor area, opening the file if required.

Show Previous Match
: This command highlights the previous match of the search expression in the editor area, opening the file if required.

Remove Selected Matches
: Removes all highlighted matches from the search results.

Search
: This command opens the Search dialog.

Previous Search Results
: This command allows you to browse previously conducted searches and repeat a previous search. You can select a previous search from the drop−down menu or clear the search history.

Related concepts

Resources
Views

Related tasks

Searching for text within a file
Searching for files

Related reference

Search view
File search

Toolbars

There are three kinds of toolbars in the Workbench.

The *main toolbar*, sometimes called the Workbench toolbar, is displayed at the top of the Workbench window directly beneath the menu bar. The contents of this toolbar change based on the active perspective. Items in the toolbar might be enabled or disabled based on the state of either the active view or editor. Sections of the main toolbar can be rearranged using the mouse.

There are also individual *view toolbars*, which appear in the title bar of a view. Actions in a view's toolbar apply only to the view in which they appear. Some view toolbars include a **Menu** button, shown as an inverted triangle, that contain actions for that view.

Finally, the *shortcut bar* is a vertical toolbar that appears along the left edge of the Workbench window. The shortcut bar allows quick access to perspectives that are currently open. It also contains *fast views*, which you can create as shortcuts to the views you use most often.

In all cases, you can find out what toolbar buttons do by moving your mouse pointer over the button and reading the tooltip that opens. See the list of related reference topics below for a table of all toolbar buttons.

Related concepts

Workbench
Views
Perspectives
Fast views

Related tasks

Creating fast views
Rearranging the main toolbar

Related reference

Toolbar buttons

Markers

Markers are objects that may be associated with Workbench resources. There are many uses of markers in the Workbench. Described below are some of the different types of markers that can be used in the Workbench.

Tasks

Two kinds of tasks exist and can be displayed in the Tasks view or on the marker bar in the editor area:

- Automatically−generated errors, warnings, or information associated with the resource
- Tasks that may or may not be associated with a resource

The different types of tasks that exist in the Workbench are described below.

Problems

Problem markers represent invalid states (i.e., errors, warnings, and/or information). The various kinds of problems are all considered tasks and can be displayed in the Tasks view.

Errors

Error markers are often used to indicate the source location of syntax or compilation errors. Errors are considered problem tasks and can be displayed in the Tasks view.

Warnings

Warning markers indicate the source location of, for example, compilation warnings. Warnings are considered problem tasks and can be displayed in the Tasks view.

Information Tasks

Information markers indicate the source location of information only tasks. Information markers are considered problem tasks and can be displayed in the Tasks view.

Bookmarks

Bookmarks place an anchor either at a specific line in a resource or on the resource itself.

Related concepts

Bookmarks
Task view

Related tasks

Creating a bookmark within a file
Creating a bookmark for an entire file

Bookmarks

You can place an "anchor" either on a resource within the Workbench, or at a specific line within a file, by creating a bookmark. Then you can use the Bookmarks view to return to those files quickly.

The Bookmarks view (**Window** > **Show View** > **Bookmarks**) displays all bookmarks that you have created.

Related concepts

Tasks view

Related tasks

Creating a bookmark for an entire file
Creating a bookmark within a file
Deleting a bookmark
Adding line items in the Tasks view
Associating a task with an editable resource

Related reference

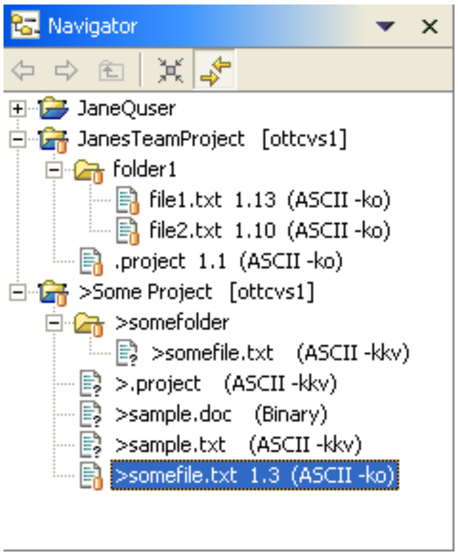
Bookmarks view

Label decorations

Label decorations are used to show extra information about an item by modifying its label or icon. They can be used to obtain information about the state of an item without having to look at its properties in the Properties view or open its Properties dialog.

A number of label decorators may be available. To control which decorators are visible, go to **Window > Preferences > Workbench > Label Decorations**. This preference page provides a selectable list and description of the available decorations.

The following is the Navigator view showing resources with CVS decorations:



```
JaneQuser
JanesTeamProject  [ottcvs1]
  folder1
    file1.txt  1.13  (ASCII -ko)
    file2.txt  1.10  (ASCII -ko)
  .project  1.1  (ASCII -ko)
>Some Project  [ottcvs1]
  >somefolder
    >somefile.txt   (ASCII -kkv)
  >.project   (ASCII -kkv)
  >sample.doc   (Binary)
  >sample.txt   (ASCII -kkv)
  >somefile.txt  1.3  (ASCII -ko)
```

Help system

The online help system lets you browse, search, and print system documentation. The documentation is organized into sets of information that are analogous to books. The help system also supplies a text search engine for finding the information you need by keyword and context−sensitive help for finding information to describe the particular function you are working with.

The help browser

When you first open the help browser by selecting **Help > Help Contents** in the Workbench, the first view shown is called Contents. Contents displays various groupings of documentation for the product. Click on one of the links to expand the navigation tree for that set of documentation. Browse the tree and click on topics to display them.

Search

If you want to find a particular piece of information in the online help, use Search. The help system will search the entire information set, or only a part of it, to find topics that meet the criteria you specify.

Context−sensitive help

If you are working through a task and encounter part of the interface that you do not understand, use infopops. Bring focus to the interface widget in question by clicking on it or using the Tab key, and then press F1. The infopop will provide you with information about the interface and, if appropriate, links to more information.

Related tasks

Accessing and navigating online help
Searching online help
Accessing context−sensitive help

External tools

External tools allow you to configure and run programs, batch files, Ant buildfiles, and others using the Workbench. You can save these external tool configurations and run them at a later time.

Output from external tools is displayed in the console view.

You can add external tools as part of the build process for a project. These external tools will run in the specified order every time a project is built.

The following variables are available when you configure an external tool. These variables are automatically expanded each time the external tool is run.

${workspace_loc} − The absolute path on the system's hard drive to Eclipse's workspace directory.

${workspace_loc:<*resource path*>} − The absolute path on the system's hard drive to the specified resource. The <*resource path*> is the full path of the resource relative to the workspace root. For example ${workspace_loc:/MyProject/MyFile.txt}. Note that the expanded result of this variable is not the same as ${workspace_loc}/MyProject/MyFile.txt if the project's contents directory for MyProject is outside the workspace directory.

${project_loc} − The absolute path on the system's hard drive to the currently selected resource's project or to the project being built if the external tool is run as part of a build.

${project_loc:<*resource path*>} − The absolute path on the system's hard drive to the specified resource's project. The <*resource path*> is the full path of the resource relative to the workspace root. For example ${workspace_loc:/MyProject/MyFile.txt}. Note that the expanded result of this variable is not the same as ${workspace_loc}/MyProject if the project's contents directory for MyProject is outside the workspace directory.

${container_loc} − The absolute path on the system's hard drive to the currently selected resource's parent (either a folder or project).

**${container_loc:<*resource path*>}** − The absolute path on the system's hard drive to the specified resource's parent (either a folder or project). The <*resource path*> is the full path of the resource relative to the workspace root. For example:${workspace_loc:/MyProject/MyFolder/MyFile.txt}. Note that the expanded result of this variable is not the same as ${workspace_loc}/MyProject/MyFolder if the project's contents directory for MyProject is outside the workspace directory.

${resource_loc} − The absolute path on the system's hard drive to the currently selected resource.

${resource_loc:<*resource path*>} − The absolute path on the system's hard drive to the specified resource. The <*resource path*> is the full path of the resource relative to the workspace root. For example ${workspace_loc:/MyProject/MyFile.txt}. Note that the expanded result of this variable is not the same as ${workspace_loc}/MyProject/MyFile.txt if the project's contents directory for MyProject is outside the workspace directory.

${project_path} − The full path, relative to the workspace root, of the currently selected resource's project or of the project being built if the external tool is run as part of a build.

${container_path} − The full path, relative to the workspace root, of the currently selected resource's parent (either a folder or project).

${resource_path} − The full path, relative to the workspace root, of the currently selected resource.

${project_name} − The name of the currently selected resource's project or of the project being built if the external tool is run as part of a build.

${container_name} − The name of the currently selected resource's parent (either a folder or project).

${resource_name} − The name of the currently selected resource.

${build_type} − The kind of build when the external tool is run as part of a build. The value can be one of "full", "incremental", or "auto". If the external tool is run outside of a build, the value is then "none".

Lets assume your Eclipse workspace directory is c:\eclipse\workspace and you have two projects, MyProject1 and MyProject2. The first project, MyProject1, is located inside the workspace directory, the second project, MyProject2, is located outside the workspace directory at c:\projects\MyProject2. Lets look at how the variable examples below will be expanded when an external tool is run, if the resource /MyProject2/MyFolder/MyFile.txt is selected.

Variable Examples	**Expanded Results**
${workspace_loc}	c:\eclipse\workspace
${workspace_loc:/MyProject1/MyFile.txt}	c:\eclipse\workspace\MyProject\MyFile.txt
${workspace_loc:/MyProject2/MyFile.txt}	c:\projects\MyProject2\MyFile.txt
${project_loc}	c:\projects\MyProject2
${project_loc:/MyProject1/MyFile.txt}	c:\eclipse\workspace\MyProject
${container_loc}	c:\projects\MyProject2\MyFolder
${resource_loc}	c:\projects\MyProject2\MyFile.txt
${project_path}	/MyProject2
${container_path}	/MyProject2/MyFolder
${resource_path}	/MyProject2/MyFolder/MyFile.txt
${project_name}	MyProject2
${container_name}	MyFolder
${resource_name}	MyFile.txt
${build_type}	none

Related concepts

External tools

Ant support
Builds

Related reference

External Tools preferences
External Tools and Ant icons

Related tasks

Running external tools
Running Ant buildfiles

Ant support

Apache Ant is an open source, Java−based build tool. See http://ant.apache.org for more details.

The Ant support allows you to create and run Ant buildfiles from the Workbench. These Ant buildfiles can operate on resources in the file system as well as resources in the workspace.

Output from an Ant buildfile is displayed in the console view in the same hierarchical format seen when running Ant from the command line. Ant tasks (for example "[mkdir]") are hyperlinked to the associated Ant buildfile, and javac error reports are hyperlinked to the associated Java source file and line number.

You can add classes to the Ant classpath and add Ant tasks and types from the ant runtime preference page, under **Window > Preferences > Ant > Runtime**.

Related concepts

Builds
External tools

Related tasks

Running Ant buildfiles
Modifying the Ant classpath
Adding new Ant tasks and types
Using a different version of Ant
antRunner application entry point

Related reference

Ant preferences
Ant editor preferences
Ant runtime preferences
Ant view
Ant editor
External Tools and Ant icons

Team programming with CVS

In the Concurrent Versions System (CVS) team programming environment, team members do all of their work in their own Workbenches, isolated from others. Eventually they will want to share their work. They do this via a CVS Repository.

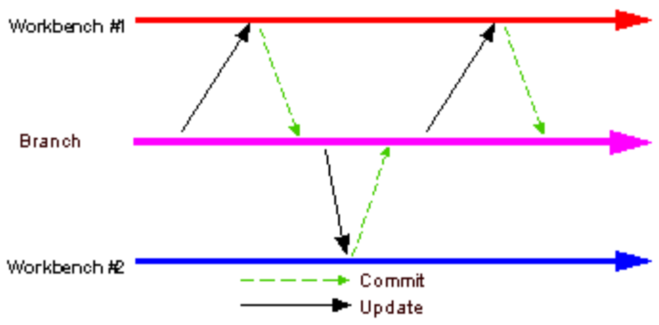
Branches

CVS uses a branch model to support multiple courses of work that are somewhat isolated from each other but still highly interdependent. Branches are where a development team shares and integrates ongoing work. A branch can be thought of as a shared workspace that is updated by team members as they make changes to the project. This model allows individuals to work on a CVS team project, share their work with others as changes are made, and access the work of others as the project evolves. A special branch, referred to as HEAD, represents the main course of work in the repository (HEAD is often referred to as the trunk).

Sharing work

As team members produce new work, they share this work by *committing* those changes to the branch. Similarly, when they wish to get the latest available work, they *update* their local workspaces to the changes on the branch. Thus the branch is constantly changing, moving forward as team members submit new work.

The branch effectively represents the current state of the project. At any point a team member can update their workspaces from the branch and know they are up to date.



Optimistic team model

CVS provides two important features required for working in a team:

- A history of the work submitted by the team
- A way to coordinate and integrate this work

Maintaining history is important so that one can compare the current work against previous drafts, revert to older work that is better, and so on. Coordinating the work is critical so that there exists one definition of the current project state, containing the integrated work of the team. This coordination is provided via the branch model.

An optimistic model is one where any member of the team can make changes to any resource he or she has access to. Because two team members can commit to the branch changes to the same resource, conflicts can occur and must be dealt with. This model is termed *optimistic* because it is assumed that conflicts are rare.

Recommended work flow

Usually resources do not exist in isolation, they typically contain implicit or explicit dependencies on other resources. For example, Web pages have links to other Web pages, and source code has references to artifacts described in other source code resources. No resource is an island.

As resources are committed to the branch, these dependencies can be affected. Ensuring the integrity of the dependencies is important because the branch represents the current project state: at any point a team member could take the branch contents as a basis for new work.

The ideal work flow therefore is one in which the branch integrity is preserved.

Ideal flow enumerated

The ideal work flow proceeds as follows:

1. Start fresh. Before starting work, update the resources in the workspace with the current branch state. If you are sure that you have no local work that you care about, the fastest way to get caught up is to select the projects you are interested in from the branch (or HEAD) and select **Checkout As Project** (or **Replace with > Latest from Repository** if the projects already exist locally). This will overwrite your local resources with those from the branch.
2. Make changes. Work locally in your Workbench, creating new resources, modifying existing ones, saving locally as you go.
3. Synchronize. When you are ready to commit your work, synchronize with the repository.

    a. Update. Examine incoming changes and add them to your local Workbench. This allows you to determine if there are changes which might affect the integrity of what you are about to commit. Resolve conflicts. Retest, run integrity checkers (for example, check for broken hypertext links, ensure your code compiles, and so on).
    b. Commit. Now that you are confident that your changes are well integrated with the latest branch contents, commit your changes to the branch. To be prudent, you may repeat the previous step if there are new incoming changes.

Of course this is an *ideal* workflow. Under certain conditions you may be confident that the incoming changes do not affect you and choose to commit without updating. However, in general team members should make an effort to follow a flow similar to the above in order to ensure that the branch integrity is not accidentally compromised.

You can find more information on CVS at http://www.cvshome.org.

Related concepts

CVS Repositories
Branches
Versions
Synchronizing with a CVS repository

Related tasks

Creating a CVS repository location

Checking out a project from a CVS repository
Replacing resources in the Workbench
Sharing a new project using CVS
Synchronizing with the repository
Updating
Resolving conflicts
Merging from a branch
Committing

Related reference

CVS

Versions

Resources are versioned in order to capture a snapshot of the current state of the resources at one specific point in time. Resources in CVS are versioned by tagging them with a version label. When a resource is versioned, it means that a non−modifiable copy of it can be retrieved from the repository.

Versioning a project saves the line up of all resource versions in the project. Resources other than projects (files and folders) can be versioned. However, it is more common to version entire projects together as the resources contained in a project are often highly interdependent. Projects can be versioned from the workspace or from the branch (including HEAD) in the CVS Repositories view. The difference between these two approaches is in deciding which child resource versions should be part of the project version.

When tagging a project as a version from the *Workbench*, the base revisions of the files in the Workbench are tagged as belonging to that version. This is the preferred method of versioning a project because you know exactly which file revisions will be in the version. This operation is allowed if you have outgoing changes or uncommitted changes. Uncommitted changes are simply ignored and resources with outgoing changes can still have their base revisions be part of the version. Versioning a project with uncommitted or outgoing changes is handy if you have to split the project at the point where you started making changes to the resources and commit the resources to another branch.

When tagging a project as a version from a *branch* in the CVS Repositories view, you are versioning whatever the latest resource versions are in the branch at that moment in time. You should not version your projects from the branch if you do not know what is committed in the branch. For this reason, versioning from the Workbench is often preferable.

Related concepts

CVS Repositories
Branches
Local history
Resources

Related tasks

Creating a version of a project
Versioning projects in the repository
Enabling the CVS resource decorations
Moving version tags

Related reference

CVS

Branches

In CVS, teams share and integrate their ongoing work in *branches*. Think of a branch as a shared work area that can be updated at any time by team members. In this way, individuals can work on a team project, share their work with others on the team, and access the work of others during all stages of the project. The branch effectively represents the current shared state of the project.

Resources can be changed in the Workbench without affecting the branch. Individuals must explicitly provide their changed resources to the branch.

Every CVS repository has at least one branch, referred to as HEAD. Under certain conditions, more than one branch may exist in a repository. For example, one branch may be for ongoing work, and another branch may be for maintenance work.

As you make changes locally in your Workbench, you are working alone. When you are ready to make your local resource changes available to other team members, you'll need to *commit* your work to the branch. All such changes are classified as outgoing changes when you do a synchronization.

Ideally, you should update your local workspace with any changes others have made in a branch before committing to it. This ensures that you have the very latest work from the other team members. After you have updated from the branch, merged any conflicting changes in your local Workbench, and tested your changes locally, you can more easily commit your Workbench's changes to the branch.

When you commit changes to the branch, your changes are copied from your local Workbench to the branch. As a result, these changes are then seen as incoming changes when other developers update from the branch later.

Related concepts

Team programming with CVS
CVS Repositories
Synchronizing with a CVS repository

Related tasks

Checking out a project from a CVS repository
Sharing a new project using CVS
Branching
Synchronizing with the repository
Updating
Committing
Resolving conflicts

Related reference

CVS
CVS Repositories view

CVS Repositories

A Concurrent Versions System (CVS) repository is a persistent data store that coordinates multi−user access to projects and their contents. Projects in a repository can be of two forms: immutable (a project version) or modifiable (a project in a branch). Communication between the repository and Workbench clients is possible over local or wide area networks. Currently the Workbench supports two internal authentication protocols for CVS: pserver and ssh. Authentication protocols provided by external tools can also be used by CVS.

You can find more information on CVS at http://www.cvshome.org.

Related concepts

Team programming with CVS
Branches
Versions
Local history

Related tasks

Creating a CVS repository location

Related reference

CVS
CVS Repositories view

Three way comparisons

Three way comparisons show the differences between three different versions of a resource. This feature is most useful when merging resources or when there is a conflict during synchronization. Conflicts occur when two developers add a version from the same branch to their Workbench, then each developer modifies it, then one developer attempts to commit the resource after the other developer has already committed it.

When this situation arises, you can view the differences between three resource versions: the resource in the Workbench, the version of the resource that is committed in the branch, and the *common ancestor* from which the two conflicting versions are based. If a common ancestor cannot be determined, for example because a resource with the same name and path was created and committed by two different developers, the comparison becomes a two−way comparison.

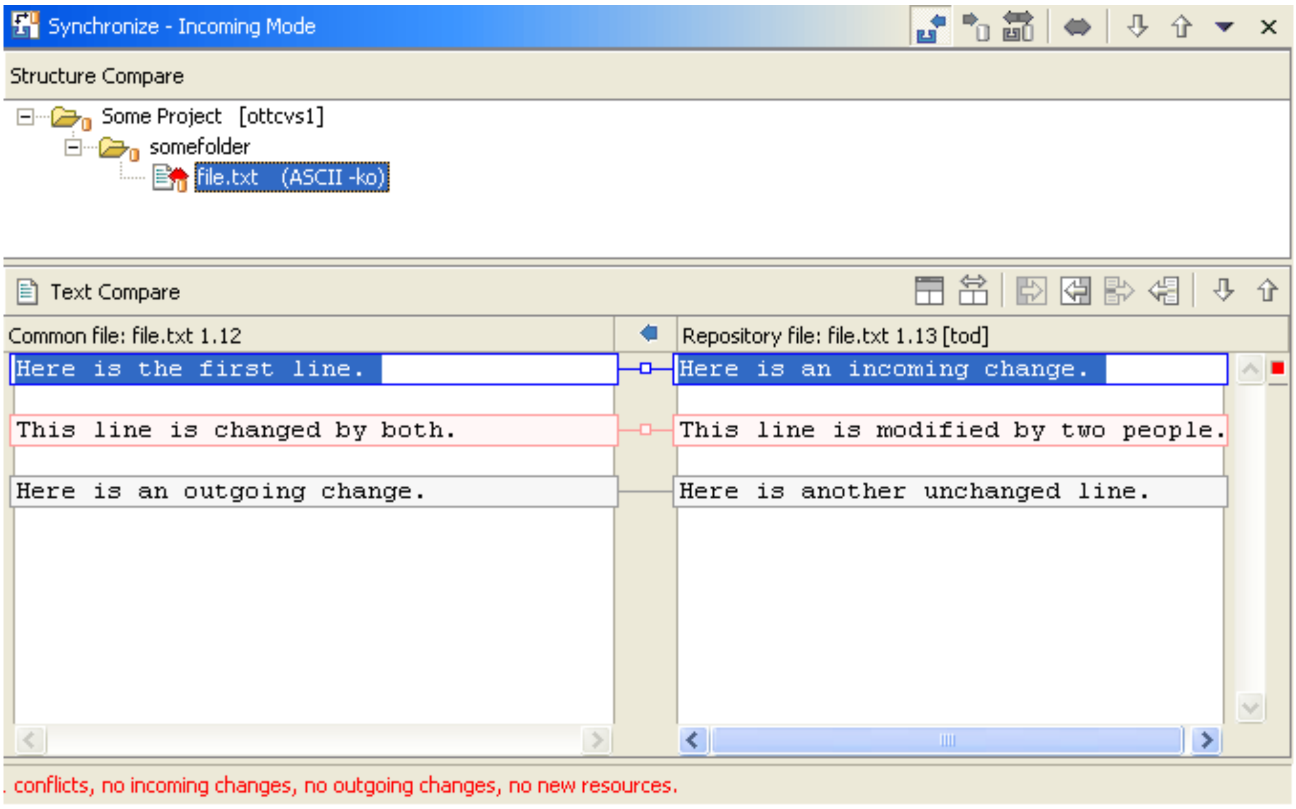
Interpreting compare results

The Synchronize view allows you to view the differences between two or three files. If a common ancestor is available, the sync view performs a three way comparison. It is possible that a common ancestor for two conflicting resource versions cannot be determined, (e.g. a resource with the same name and path is created and committed by two different developers). In this case the compare becomes a regular two way compare.

In a three way compare the Workbench shows you:

- what has been changed in the first child in comparison to the common ancestor
- what has been changed in the second child in comparison to the common ancestor.

In the picture below, the common ancestor is displayed in the top pane of the text compare pane. The differences that you see highlighted are what has changed in the Workbench resource as compared to the common ancestor, and what has been changed in the branch resource as compared to the common ancestor. The sections that differ in all three files are highlighted as differences. Conflicts are shown in red, incoming changes in blue, and outgoing changes in gray.



Common file: file.txt 1.12

```
Here is the first line.

This line is changed by both.

Here is an outgoing change.
```

Repository file: file.txt 1.13 [tod]

```
Here is an incoming change.

This line is modified by two people.

Here is another unchanged line.
```

. conflicts, no incoming changes, no outgoing changes, no new resources.

Related concepts

Synchronizing with a CVS repository

Related tasks

Comparing resources
Synchronizing with a repository
Merging changes in the compare editor

Resolving conflicts
Setting preferences for comparing files
Comparing resources with repository versions

Related reference

Compare editor
CVS Synchronize view

Synchronizing with a CVS repository

In the CVS team programming environment, there are two distinct processes involved in synchronizing resources: *updating* with the latest changes from a branch and *committing* to the branch.

When you make changes in the Workbench, the resources are saved locally. Eventually you will want to commit your changes to the branch so others can have access to them. Meanwhile, others may have committed changes to the branch. You will want to update your Workbench resources with their changes.

Important!: It is preferable to update *before* committing, in case there are conflicts with the resources in your Workbench and the resources currently in the branch.

The synchronize view contains filters to control whether you want to view only i*ncoming changes* or *outgoing changes*. Incoming changes come from the branch. If accepted, they will update the Workbench resource to the latest version currently committed into the branch. Outgoing changes come from the Workbench. If committed, they will change the branch resources to match those currently present in the Workbench.

Regardless of which mode (filter) you select, the Synchronize view always shows you conflicts that arise when you have locally modified a resource for which a more recent version is available in the branch. In this situation you can choose to do one of three things: update the resource from the branch, commit your version of the resource to the branch ,or merge your work with the changes in the branch resource. Typically you will want to merge, as the other two options will result in loss of work.

Related concepts

Team programming with CVS
Branches

Related tasks

Synchronizing with the repository
Updating
Resolving conflicts
Merging from a branch
Committing

Related reference

CVS
CVS Synchronize view

Watch/Edit

CVS provides a notification scheme which allows a group of developers to know if somebody is working on a given file. This facility is known as *watches*. By setting a *watch* on a file, you can have CVS notify you via email if someone else starts to *edit* this file. This mechanism is notification based only; the file is not locked in any way on the server, and several people are allowed to edit the same file at the same time.

In addition to watch list notification, *edit* on its own is useful for discovering if others are also editing that file. This is because when you edit a file, you will be informed if someone else is already editing it.

Normally with CVS clients, an explicit *edit* would need to be issued by the user. With Team CVS support however, an *edit* is automatically issued by the client when you start to modify a file. In addition, when *editing* a file, Team CVS provides you with the list of people already editing a file. This allows you to find out who is working on a file before you start to edit it.

Related concepts

Team programming with CVS

Related tasks

Finding out who's working on what: watch/edit

Related reference

CVS

Accessibility features in Eclipse

Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or those with special needs to use software products successfully. These are the major accessibility features in Eclipse:

- Eclipse uses Microsoft Active Accessibility (MSAA) APIs to render user interface elements accessible to assistive technology.
- You can operate all features using the keyboard instead of the mouse. See the related task.
- You can use screen−reader software such as Freedom Scientific's JAWS ™ and a digital speech synthesizer to hear what is displayed on the screen. You can also use voice recognition software, such as IBM ViaVoice ™ to enter data and to navigate the user interface.
- You can magnify what is displayed on your screen in the graphical views.
- Any fonts or colors defined by Eclipse can be set using the ***Window*** > ***Preferences*** dialog**.** See the related link.

Note: The Accessibility features mentioned in this document apply to the Windows operating system.

Related tasks

Navigating the user interface using the keyboard

Related reference

Keys
Font and color settings in Eclipse

Navigating the user interface using the keyboard

The user interface is navigable using the keyboard. The Tab key is used to iterate through the controls in a particular scope (for example, a dialog or a view and its related icons). To navigate to the main controls for the Workbench window or to tab out of views that use the Tab key (such as editors) use Ctrl+Tab.

Menus

Most menus are assigned mnemonics for each entry which allow you to select them by typing the underlined letter instead of the mouse. You can also select an item by moving through the menus and sub−menus with the arrow keys.

The various menus available can be accessed using the keyboard in the following ways:

- F10 accesses the menus on the main menu bar.
- Shift + F10 pops up the context menu for the current view.
- Ctrl+F10 will pull down the pull down menu for the current view if there is one.
- Alt − pops up the main view menu.
- Alt + mnemonic will activate the Workbench menu for a particular entry (i.e. Alt + W will bring down the Window menu).

Controls

Mnemonics are assigned to most labels of controls (for example, buttons, check boxes, radio buttons) in dialog boxes, preference pages, and property pages. To access the control associated with a label, use the Alt key along with the letter that is underlined in the label.

Navigation Context

Navigation context is saved for the packages, navigator views, Workbench preferences and properties dialogs. The expanded state of the tree in the packages and resource views are saved between Workbench sessions. The selected page for the preferences and properties dialog is saved between invocations of the dialog but are not saved between workbench invocations.

Cycling Editors, Views and Perspectives

To switch between editors, views and perspectives, the workbench provides a cycling function that is invoked by Ctrl and a function key. All of these cycling functions recall the last thing selected to allow for rapid cycling back and forth between two items. The cycling functions are

- Ctrl+F6 − Cycle to Editor
- Ctrl+F7 − Cycle to View
- Ctrl+F8 − Cycle to Perspective

Accelerators

Many of the actions in Eclipse have an accelerator assigned to them. For the full list of these accelerators see the Keyboard shortcuts reference below.

Help system

You can navigate the help system by keyboard using the following key combinations:

- Pressing Tab inside a frame (page) takes you to the next link, button or topic node.
- To expand/collapse a tree node, press Right/Left arrows.
- To move to the next topic node, press Down arrow or Tab
- To move to the previous topic node, press Up arrow or Shift+Tab
- To display selected topic, press Enter.
- To scroll all the way up or down press Home or End.
- To go back press Alt+Left arrow; to go forward press Alt+Right arrow.
- To go to the next frame, or toolbar press Ctrl+Tab (Ctrl+F6, if using browser based on Mozilla 1.2 or newer).
- To move to previous frame, press Shift+Ctrl+Tab. (Shift−Ctrl+F6, if using browser based on Mozilla 1.2 or newer).
- To move between tabs, press Right/Left arrows.
- To switch view, select a tab and press Enter.
- To switch and move to a view, select a tab and press Up arrow.
- To print the current page or active frame, press Ctrl+P.

Related concepts

Accessibility features in Eclipse
Changing the key bindings
Keys
Online help system

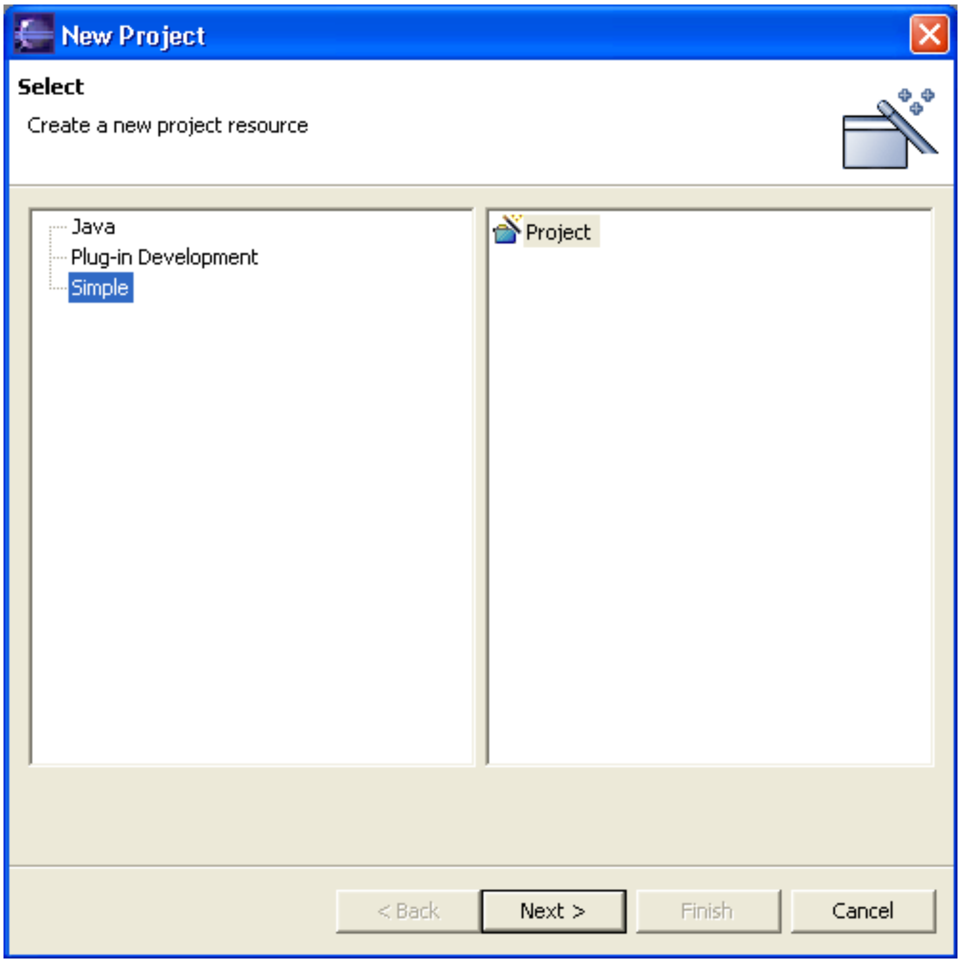
Related reference

Font and color settings in Eclipse

Fonts and colors in Eclipse

Eclipse uses the fonts and colors provided by the operating system as much as possible. On Windows the platform color and font settings are found on the **Properties** > **Appearance** page. The font used by most widgets in Eclipse is the one set in the Message Box settings of the properties. However, operating systems do not provide enough colors to handle all of the extra information that colors and fonts provide in Eclipse.

Fonts



There are 4 main fonts in use by the Eclipse platform. They are:

Banner Font
: Used in PDE editors, welcome pages and in the title area of many wizards. For instance the New Project wizard uses this font for the top title,

Header Font
: Used as a section heading. For instance the Welcome page for the Eclipse Platform uses this font for the top title,

Text Font
: Used in text editors.

Dialog Font
: Used in dialogs.

These fonts can be set via the **Workbench** > **Font** preference. As well as these 4 fonts there are several other secondary font settings. These default to the text font. They can be found on the font preference page:

- **Compare Text Font**
- **CVS Console Font**
- **Debug Console Font**
- **Detail Pane Text Font**
- **Java Compare Text Font**
- **Java Editor Text Font**

Colors

Eclipse uses colors as an information enhancement in many places. Whenever possible the operating system color settings are used, but in cases where the operating system settings are not enough, Eclipse defines other colors. All of these colors can be adjusted via the following preference pages:

- **Workbench** > **Appearance** (Error text, hyperlink text, active hyperlink text)
- **Workbench** > **Search** (Foreground for inexact matches)
- **Debug** > **Console** (Standard Out, Standard Error, Standard In)
- **Debug** > **Variable Views** (Changed variable)
- **Ant** (Error, Warning, Information, Verbose, Debug)
- **Install/Update** (Section Title Color)
- **Java** > **Editor** > **Appearance** (Line number, matching brackets, current line, problems, print, find scope, linked position)
- **Java** > **Editor** > **Syntax** (Multi line comment, single line comment, keywords, strings, others, javadoc keywords, javadoc html, javadoc links, javadoc others, background)
- **Java** > **Editor** > **Code Assist** (background and foreground for completion proposals and method parameters)
- **Plug−in Development** > **Editors** (Text, processing instructions, constant strings, tags ,comments)
- **Team** > **CVS** > **Console** (Command line, message, error)

WIN Accessibility and the Windows Color Dialog

For color selection, Eclipse uses a dialog provided by the operating system. On windows, the color selection dialog does not respond properly to assitive technology. When you first get into the dialog, focus is on one of the basic colors, but the dialog provides no indication of this through assistive technology. You can select colors in Eclipse with this dialog in the following way:

1. Select to customize the color of something in Eclipse, for example the color of Error Text in your Workbench Appearance preferences.
2. In the color selection dialog, tab twice to go from the Basic Color matrix to the Define Custom Colors button and press Enter.
3. You can now enter the basic colors using an HSL or RGB specification according to the following definitions. See the Windows Color Dialog Reference for a tables and values for these colors.

Related tasks

Accessibility Features in Eclipse

Navigating the user interface by using the keyboard

Related reference

Keys
Windows Color Dialog Reference

Features

On disk, an Eclipse based product is structured as a collection of *plug−ins*. Each plug−in contains the code that provides some of the product's functionality. The code and other files for a plug−in are installed on the local computer, and get activated automatically as required. A product's plug−ins are grouped together into features. A *feature* is the smallest unit of separately downloadable and installable functionality. (The notion of a feature is new for Eclipse 2.0; it replaces the similar notion of component in Eclipse 1.0.)

The fundamentally modular nature of the Eclipse platform makes it easy to install additional features and plug−ins into an Eclipse based product, and to update the product's existing features and plug−ins. You can do this either by using traditional native installers running separately from Eclipse, or by using the Eclipse platform's own update manager. The Eclipse *update manager* can be used to discover, download, and install updated features and plug−ins from special web based Eclipse update sites.

The basic underlying mechanism of the update manager is simple: the files for a feature or plug−in are always stored in a sub−directory whose name includes a version identifier (e.g., "2.0.0"). Different versions of a feature or plug−in are always given different version identifiers, thereby ensuring that multiple versions of the same feature or plug−in can co−exist on disk. This means that installing or updating features and plug−ins requires adding more files, but never requires deleting or overwriting existing files. Once the files are installed on the local computer, the new feature and plug−in versions are available to be configured. The same installed base of files is therefore capable of supporting many different *configurations* simultaneously; installing and upgrading an existing product is reduced to formulating a configuration that is incrementally newer than the current one. Important configurations can be saved and restored to active service in the event of an unsuccessful upgrade.

Large Eclipse based products can organize their features into trees starting from the root feature that represents the entire product. This root feature then includes smaller units of functionality all the way down to leaf features that list one or more plug−ins and fragments. The capability to group feature hierarchically allows products to be stacked using a 'Russian doll' approach − a large product can build on top of a smaller one by including it and adding more features.

Some included features may be useful add−ons but not vital to the proper functioning of the overall product. Feature providers can elect to mark them as **optional**. When installing optional features, users are provided with a choice of whether they want them or not. If not installed right away, optional features can be added at a later date.

The **About** option on the **Help** menu provides information about installed features and plug−ins. The **Software Updates** submenu on the **Help** menu groups together items for updating existing features, and for finding, downloading, and installing new features.

Related concepts

Workbench

Related tasks

Inspecting the current configuration
Installing new features with the update manager
Installing several features at once using group updates
Picking up pending changes
Updating features with the update manager

Saving a configuration
Restoring a saved configuration