Forte™ for Java™ (Community Edition) User's Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 U.S.A.
650 960-1300 Fax 650 969-9131

Part no.: 806-4570-10
Revision 1, February 2000

Contents

Preface

This book describes how to develop with and customize Forte™ for Java™, Community Edition. This document is intended for software engineers who implement Java™ applications and applets with Forte for Java, Community Edition.

Before you read this book

It is assumed that the reader has a general knowledge of Java. Java beginners can use this guide in conjunction with documentation for the Java™ 2 Platform, Standard Edition, v. 1.2.

This manual is not a comprehensive guide to the Java 2 Platform. For more detailed information about Java, download the documentation that comes with your implementation of the Java 2 Platform. For the Java 2 SDK, Standard Edition, v. 1.2.2, you can download the documentation directly from the Sun web site at http://java.sun.com/products/jdk/1.2/download-docs.html.

How this book is organized

After the introduction, the User's Guide is divided into the following chapters.

- "IDE Essentials" on page 21 provides a quick tour of major components and features of the IDE's user interface. This is a good place to start if you are using the IDE for the

first time and need general information about how to navigate and find commands.

- "Developing Java Applications" on page 33 is a "how to" chapter, which provides an overview of developing, running, and debugging applications using the Forte for Java IDE.

- "The Explorer and Object Browser" on page 73 provides an in-depth look at the Explorer and Object Browser and gives information about their many features, including helpful code generation options.

- "Developing and Customizing JavaBeans Components" on page 98 provides information about creating and customizing JavaBeans components, as well as making them available in the IDE for development .

- "Designing Visual Classes with the Form Editor" on page 111 focuses on all aspects of visual development in the IDE.

- "Developing Java Server Pages" on page 148 shows how you, with the help of templates and a wizard, can quickly create Java Servlet Pages (JSPs) with the IDE.

- "Organizing work into projects" on page 153 details Forte for Java's project functionality and how you can save different IDE configurations for each project.

- "Integrating a CVS version control system" on page 162 provides information on how you can institute a version control system for applications you and others develop with the IDE.

- "Configuring and customizing the IDE" on page 177 provides a more thorough explanation of the parts and features of the Forte for Java IDE and shows how you can customize it to best suit the way you work.

The appendices provide reference information, such as lists of keyboard shortcuts and Editor abbreviations, as well as brief descriptions of menu and toolbar items, properties of Project Settings and Global Options items, and all available IDE actions.

Conventions Used In This Document

The following conventions are used in the text:

- **This font is used to denote**

  — items you can select in the GUI, such as buttons and menu choices

- `This monospaced font is used to denote`

- Examples of Java code
- File names
- Explorer nodes
- Property names and values on the property sheet

- Meta-names (for example, words like *YourName*), which describe the type of text to be entered rather than the literal string, are *italicized*.

"Press ENTER" means that you should press the Enter or Return key on your keyboard. Keys like F5 and F9 refer to those function keys (also sometimes labeled PF5 and PF9). "CTRL" refers to the Control key. A set of keystrokes joined with "+", like CTRL+F9, means you should press the first key (here, CTRL), hold it down, and press the second key (here, F9).

In listed source or command line code, lines indented from the previous ones should be entered on the same line when typed into the Editor window or console. For example, the following should all be typed as one line:

```
C:\jdk1.2.2\jre\bin\java -jar "C:\Program
      Files\Forte4J\modules\openfile.jar" -port 2121 "C:\My
      Development\com\mycom\Foo.java"
```

Look and feel of graphics

The pictures of Forte for Java's graphical user interface (GUI) that appear in this document were taken using a Microsoft Windows NT machine and the Java look and feel. The appearance of your GUI may differ from the pictures presented here, depending on which operating system and look and feel you are using.

Contact Information

For the latest news and information, check the NetBeans web site at http://www.netbeans.com/.

Chapter 1

Welcome to Forte for Java, Community Edition

Forte™ for Java™, Community Edition is a Java integrated development environment (IDE) written in Java. Forte for Java is a cross-platform tool that is fully functional for client and server side development. Forte for Java takes advantage of the strengths of Java to provide you with a dynamic and responsive environment.

Forte for Java is intuitive. The user interface provides several ways to perform most tasks and helps users with well-placed contextual menus and tool tips.

Forte for Java is customizable. The GUI can be modified to become a reflection of your own development style. You can easily customize the menus, toolbars, Component Palette, workspaces, and other settings.

Forte for Java is modular. This means that the IDE functionality for editing, debugging, GUI generation, EJB support, and so on is represented in modules that you can download and update dynamically. Instead of waiting months for a new major release, you can upgrade the latest modules from Sun Microsystems and our partners as soon as they are available.

Forte for Java is extensible. Forte for Java has a complete set of Open APIs that are

available to our users and partners – the same set of APIs that our own developers use to build Forte for Java. Complete new Java tools such as visual builders for industrial application domains can take advantage of a mature tool-construction platform!

Product features

Forte for Java, Community Edition has a wide range of features.

Editor

The Editor provides all standard editing functions and has many other useful features such as:

- Java code completion
- customizable syntax coloring for Java, HTML, and JSPfiles
- advanced search capabilities
- the ability to jump to the source for the class or method the insertion point is in, the variable declaration, or the Javadoc documentation for the source directly from the Editor window
- "jump list" capability, enabling you to easily jump back and forth through the various points where you have been working in your files
- a wide selection of other customizable keyboard shortcuts, including ones for features such as indenting text, commenting out text, and Editor bookmarks
- customizable Editor abbreviations

Form Editor

You can design graphical user interfaces for your applications visually with the Form Editor. The Form Editor augments the Editor with two more windows: the Form Editor window and the Component Inspector. The Form Editor window provides a design-time representation of your form as you are building it. You can easily add components to a form by clicking on a button in a special toolbar, the Component Palette, and then clicking on the part of the form where you want to place the component. Clean and readable code for the components is then generated automatically in the Editor. The Component Inspector window provides makes it easy to modify component properties.

The Form Editor includes advanced support for Java layout managers, including the **GridBag layout customizer** which makes the most complex standard Java layout manager

easier to use. The GridBag customizer dialog box provides you with a dynamic visual representation of the components in their respective "grid bags" and gives you different ways of adjusting the constraints (such as dragging, direct entry of values, and tool icons).

The Form Editor enables you to immediately to use any JavaBeans component for visual development. Because they use standard Abstract Window Toolkit (AWT) and Java Foundation Classes (JFC) components, generated forms do not depend on any proprietary code or components. No classes need to be bundled with your forms as the Form Editor generates code that is entirely independent of the IDE (unless you use Absolute Layout, a special Forte for Java feature which is an improvement on null layouts).

Explorer and Properties window

The Explorer provides a unified hierarchical view of Java classes and their elements, Javadoc documentation, and running and debugging processes. The Properties window enables you to view and edit the properties for the object represented by the node selected in the Explorer.

Object Browser

With the Object Browser, all of your sources are logically presented in one streamlined view. You can view your applications by package, object, and member (or bean patterns), as well as apply filters to customize what is displayed. Using context menus in the Object Browser, you can open files, access their property sheets, compile and execute applications, and more.

Multi-threaded debugger

Forte for Java provides full support for multi-threaded debugging and standard debugging features such as breakpoints, watches, trace into, trace over, and trace out.

Project functionality

It is possible for you to organize your work into projects with different IDE configurations for each.

CVS support

You can easily integrate CVS (Concurrent Versions System) with Forte for Java to make it easier to apply version control to your files and work on team projects.

Javadoc support

Forte for Java provides you easy access to Java API documents from within the IDE. You can also have Javadoc comments automatically generated for your files and produce your own Javadoc documentation.

Update Center

The Update Center feature enables you to connect to Forte for Java's web site straight from your IDE and download and automatically install new and updated modules.

Special support for JavaBeans components

Forte for Java provides special support for creating JavaBeans components, including generation of properties, event sets, and bean info. You can also customize them and add them to the IDE so that they can be easily dropped into applications.

Source synchronization

The Java source synchronization feature can save you time by automatically generating all of the implementation methods used by your source code.

Chapter 2

IDE Essentials

This chapter provides a brief overview of the parts of the Forte for Java graphical user interface (GUI) that you will first encounter when you start using the IDE. Once you have familiarized yourself with the concepts outlined here, you will have a clearer understanding of how to work in the IDE, where to look for commands for various tasks, and how to access help documentation if you do not immediately find what you are looking for.

User interface

The core of the user interface consists of the Main Window, the Explorer, the Editor, the Project Settings window, and the Global Options window. In this section, we outline the function of these windows as well as that of the Form Editor, which is a tool that simplifies the building of visual applications.

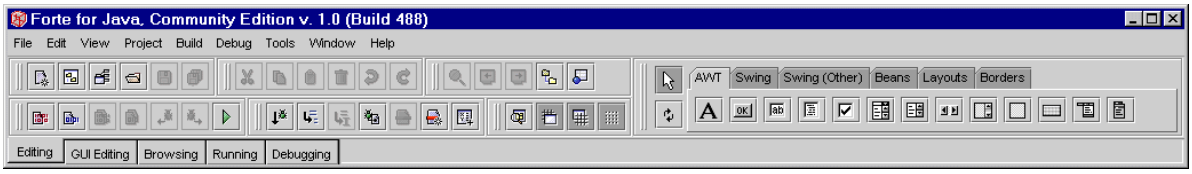
Other tools that work closely with core IDE such as the HTML browser, the Component Inspector, the Object Browser, the Debugger, the Execution View, and the Output Window will be discussed in later chapters.

For information on how to adjust the default set of windows, workspaces, and tools to match

your preferences, see "Configuring and customizing the IDE" on page 177.

Main Window

The Main Window opens when Forte for Java is launched and remains open as long as Forte for Java is running. The Main Window can be viewed as the control center of the IDE. Most important operations and commands are accessible from this window. The Main Window can be broken into four separate groups of controls: the menus, the toolbars, the workspace tabs, and the status line.



Menus and toolbars

A listing of all menu items and toolbar operations is given in "Main Window Menus and Toolbars" on page 246, and specific operations are mentioned in pertinent sections. Menu items and toolbar tools are context sensitive: they may be dimmed, depending on which window is currently active or which object is selected. Menu entries and toolbars can be re-ordered and customized. Keyboard shortcuts are shown in their corresponding menu items. See "Customizing menus and toolbars" on page 215 for more information.

Component Palette

The Component Palette (shown in the right half of the figure above) is a special toolbar used in conjunction with the Form Editor to visually build forms. It consists of several tabs, each housing standard components and layouts.

Workspaces

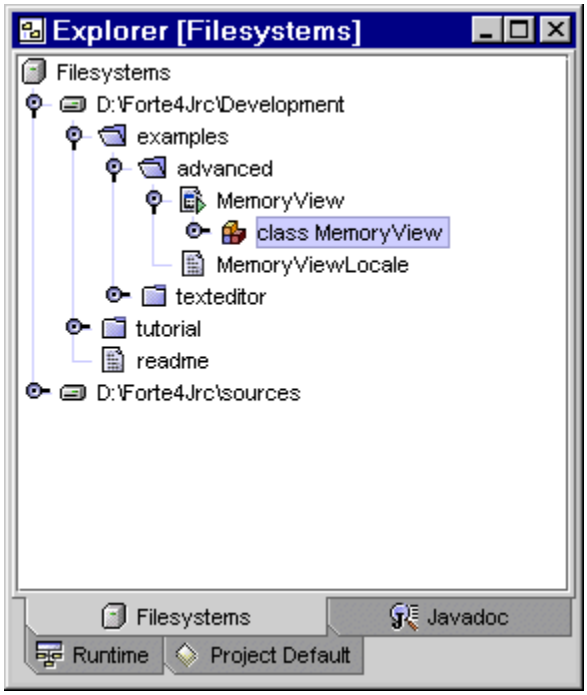
Forte for Java uses the concept of **workspaces** to manage windows and tools. On the lower left corner of the Main Window are five workspace tabs: **Editing**, **GUI Editing**, **Browsing** (unless the Object Browser is not installed), **Running**, and **Debugging** (unless there is no Debugger installed). Each workspace contains a set of opened windows appropriate to specific stages of the development cycle. Clicking on each of these tabs "flips" between each workspace. By default, the IDE automatically switches to the GUI Editing Workspace when you open up a visual form, to the Running Workspace when you execute an application, and to the Debugging Workspace when you initiate a debugging session. For more information on managing workspaces, see "Workspaces" on page 188.

Contextual menus

Many of the commands available in Forte for Java are available through contextual menus.

The Explorer

The Explorer in Forte for Java, Community Edition gives a unified view of all objects and files and provides a starting point for many programming functions. Here, you can work with objects, organize your work, modify object properties, and connect various data sources.



Navigation

When you first start Forte for Java and open the Explorer, you will see a multi-tab window with tabs labeled **Filesystems**, **Project**, **Runtime**, and **Javadoc**. Click on a tab to go to that part of the Explorer.

Navigating the hierarchy within each category is simple. Click on the expand button (for Metal, for CDE Motif, and for Windows) next to each item to expand the tree. Each tree node represents an object, and the object types are visually distinguished with icons. See "Object Types" on page 184 for an overview of available object types.

Right-click on any item to access its contextual menu, which contains the context-sensitive set of operations (as well as access to the property sheet) available for that item.

Default operations

In the Explorer window, double-clicking on an item or selecting the item and pressing ENTER performs the default operation on that object.

The default operation varies for each object type. The most common operation is opening the selected object in an Editor window. Double-clicking on a Java object opens the source in the Editor. Double-clicking on a method or variable of a Java class, as displayed under a parent, opens the Editor window and positions the insertion point on the line where that method or variable is declared. Double-clicking on an HTML file opens it in the internal web browser. Double-clicking on a folder object, such as a top-level node, simply expands or collapses the sub-tree.

Property sheets

Property sheets display (and, in the case of writable properties, enables you to edit) the properties of items such as class files, elements of classes, JavaBeans properties, visual components, HTML files, running processes, and debugging processes, and IDE settings. The property sheets for the Project Settings and Global Options windows can be used to configure the IDE.

When multiple items are selected, the property sheet displays the properties that are common to all of the selected items. If the items have multiple sets of properties, each set of properties is displayed on a separate tab.

Accessing the property sheet

The properties for objects listed in the Explorer, Object Browser, and Search Results can be viewed in the standard Properties Window. This window appears by default in the Editing workspace and can be displayed by selecting **View | Properties** from the main menu or using the ALT+1 shortcut. This window displays the properties of the item selected in the Explorer, Object Browser, and Search Results, depending on which of these is or was most recently the active window.

You can also display a separate Properties window for an object by right-clicking on it in the Explorer, Object Browser, or Search Results window and selecting **Properties** from the contextual menu.

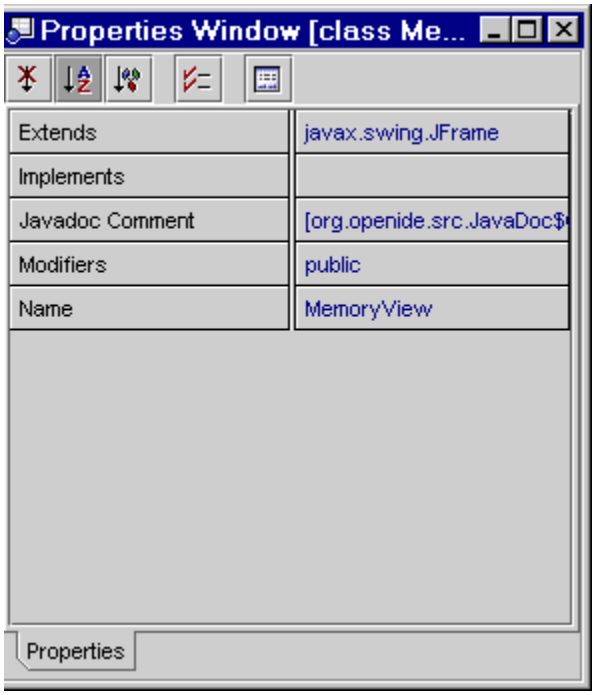
Note: If you open a property sheet by using the contextual menu of a node, the properties for that node will remain displayed in this window, even if you later select a different node.

The Component Inspector, Debugger Window, Project Settings window, and Global Options window each have their own property sheet pane which displays the property sheet for the item selected in the top pane of that window.

Contents of the property sheet

The property sheet consists of paired names and values. Each row of the property sheet represents a single property of one or more (if multiple objects are selected) items. A name/value pair may be dimmed if it represents a read-only property that is not connected with a custom property editor (a dialog box tailored to the specific property enabling editing or input of more complex values).

Each property name has a tool tip label which gives a brief description of the property. To access the tool tip, rest the mouse over the property name for a moment until the tool tip appears.



For JavaBeans properties, the name of the property is derived from the display name of the bean property. The tool tip for each property name displays its accessibility information and a short description for that property. Accessibility details are shown in parentheses (`r/w`, `r/-`, `-/w` or `No property editor`) and depend on the property and its property editor (see "Custom property editors" below).

The value field either shows a property value as text or paints a representation of it (like the color preview for a color property). The tool tip for the value of the property displays the type of property, such as `java.awt.Color`. Clicking on this area switches the value field into input mode, enabling the value of the property to be changed. Double-clicking a name switches the value if there are just two possibilities (for instance, changing `True` to `False`)—or, if multiple values are possible, the value advances to the next possible (for instance, `black` might change to `blue`, then to `red` and so on for each double-click). There are several ways to change the value, depending on the type of property. You may edit the value as text, choose from a pull-down list, open a custom property editor (by selecting the

property and then pressing the **...** button that appears), or customize a painted preview directly in the painted area.

Property sheet toolbar

The first three icons on the property sheet toolbar are toggles for sorting (to leave unsorted, sort by name, and sort by type, respectively). The next toolbar button enables filtering of properties by accessibility (read/write) so that only writable properties are displayed. The right-most toolbar icon brings up the Customizer. The Customizer is a dialog box which can be used for advanced customization of the whole object at once. This icon is context sensitive and is only available for certain objects.

Custom property editors

A **custom property editor** is a dialog box, brought up by pressing the **...** button on a property value in a property sheet, specially designed for editing the value of that property. Custom property editors range from the simple (for example, editors for strings) to complex (for example, for setting colors in the Editor).

All changes made in custom property editors are applied throughout the environment immediately – for example, changing the background color of the Editor window changes not only any new Editor windows you open but also any that are currently open.

All Property Editor dialog boxes contain these three buttons:

- The **OK** button closes the dialog box and is shown only if the property is writable.
- The **Cancel** button reverts to the setting before the property editor was called.
- For properties that have default values, the **Default** button sets the property to its default value.

For more information on the various property editors, see "Custom Property Editors" on page 288.

Form Editor

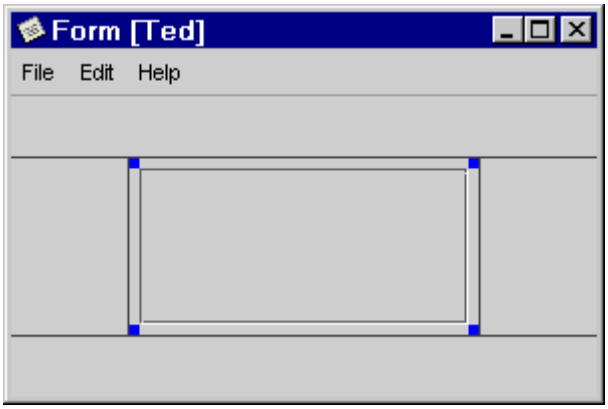
The Form Editor enables you to design user interfaces visually. You can select items such as panels, scroll bars, menus, and buttons in the Component Palette and then place them directly on the Form Editor window. As you do, Forte for Java automatically generates the Java code to implement the design and behavior of the application. (The code is visible in the Editor window.) The Form Editor also uses your chosen Java layout manager, which controls the appearance of the components in a window, to control the layout in the Form Editor. If you choose a different layout manager or change its parameters, the Form Editor displays your

changes immediately.

The Form Editor consists of three parts: the Form Editor window, the Component Inspector, and the Editor window.

Form Editor window

The Form Editor window shows the design-time view of the form you are creating. You can add components to the form by selecting them in the Component Palette and then clicking on the Form Editor window. You can also manipulate the components within the Form Editor window by dragging them or by right-clicking on them and choosing commands from the contextual menu to do things such as change their order, delete them, rename them, and access their property sheets.

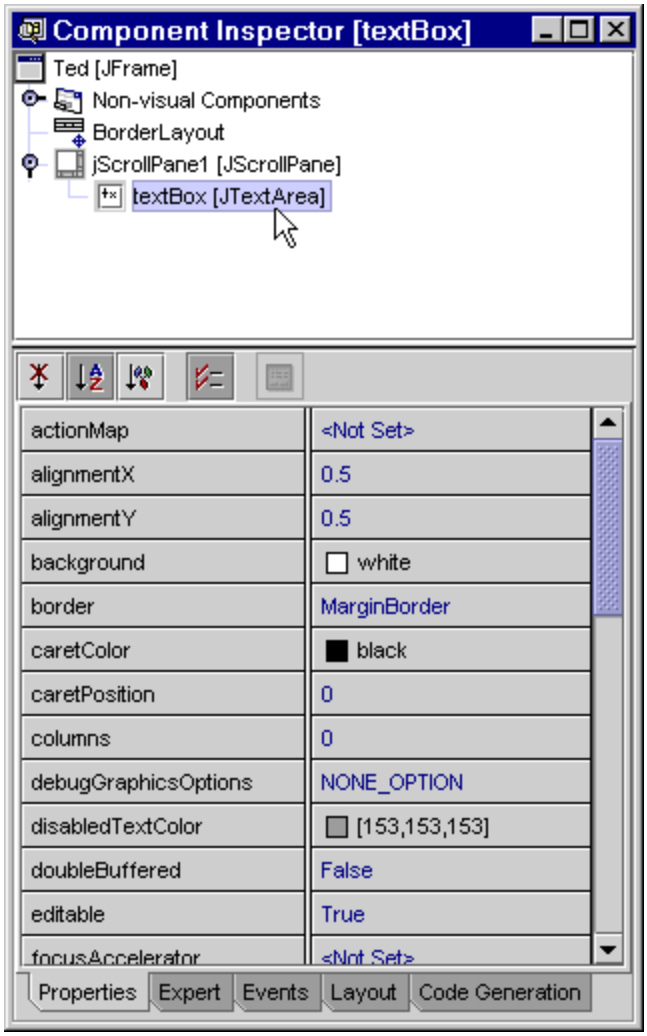


For a comprehensive guide to using the Form Editor, see "Designing Visual Classes with the Form Editor" on page 111.

Component Inspector

The Component Inspector is the part of the Form Editor that resembles the Explorer, except that it shows only information specific to the active form. It lists all of the form's components, including non-visual ones such as layout managers, and highlights the currently selected component, whether that component was selected in the Component Inspector or the Form Editor window. Tabs on the property sheet in the bottom panel display the general (the tab marked **Properties**), expert, code generation, and layout properties as well as the events of the selected component. See "Property Sheet pane in the Component Inspector"

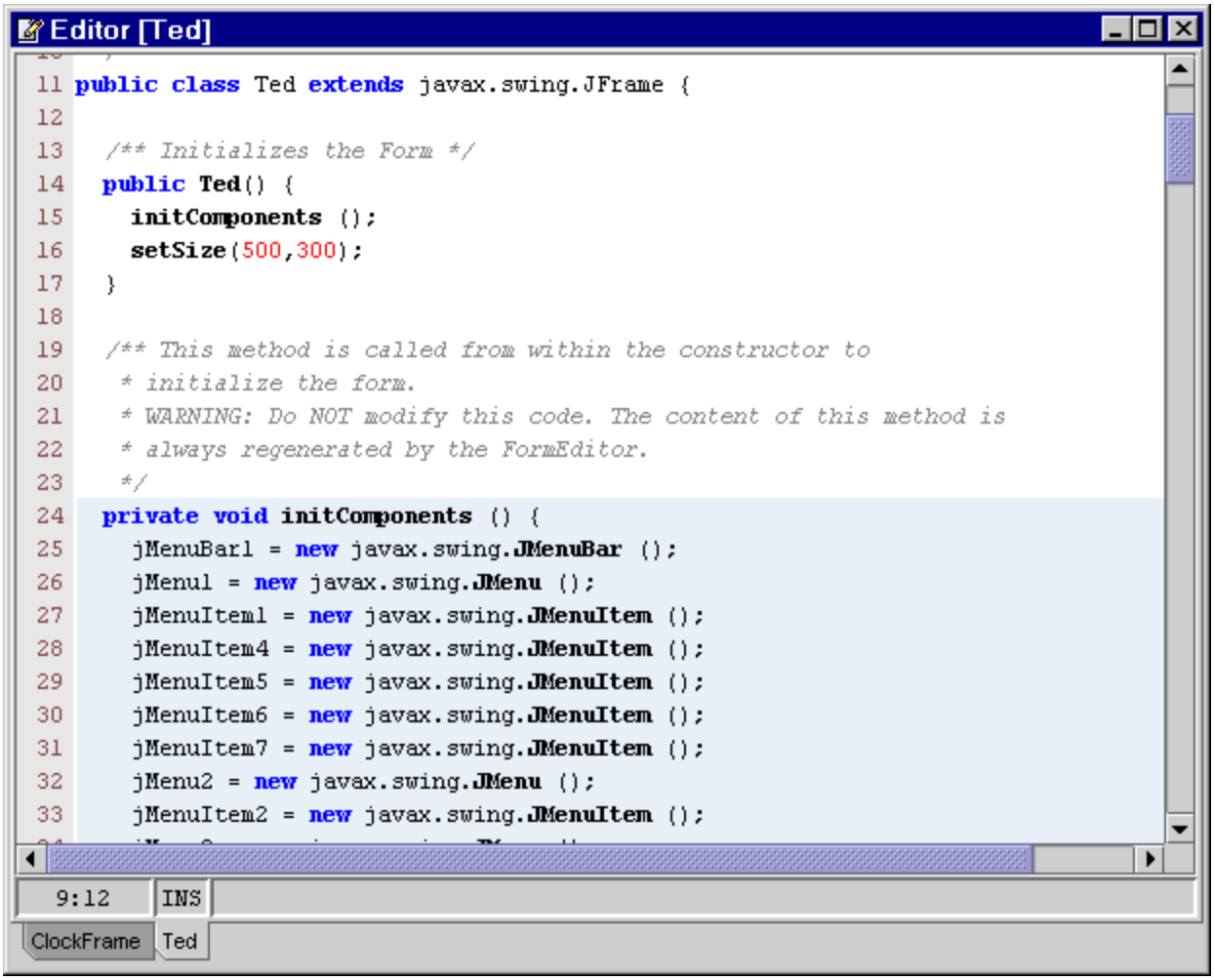
on page 117



Editor window

Besides acting as a text editor, the Editor window also displays code generated by the IDE as a result of additions and changes that you make in the Form Editor window and the Component Inspector. Such IDE-generated code appears in guarded blocks (denoted by background shading) meaning that you can not edit them directly in the Editor window. However, it is possible to affect the way the code is generated – see "Using the Form

Connection property editor" on page 139 for more information.



Project Settings and Global Options windows

These two windows hold the IDE's configurable settings. The Project Settings window (which can be opened by choosing **Project | Settings...** from the main menu) holds settings which can be configured separately for individual projects. Examples of Project Settings include Compiler Types, Executor Types, and Java Sources. The Global Options window (available by choosing **Tools | Global Options...** from the main menu) enables you to configure settings which apply to IDE as a whole. These settings include Menus, Toolbars, Component Palette, and proxy settings (found under System Settings). See "Project Settings" on page 177 and "Global Options" on page 182 for detailed descriptions of the items in each window.

JavaHelp, context help, and tool tips

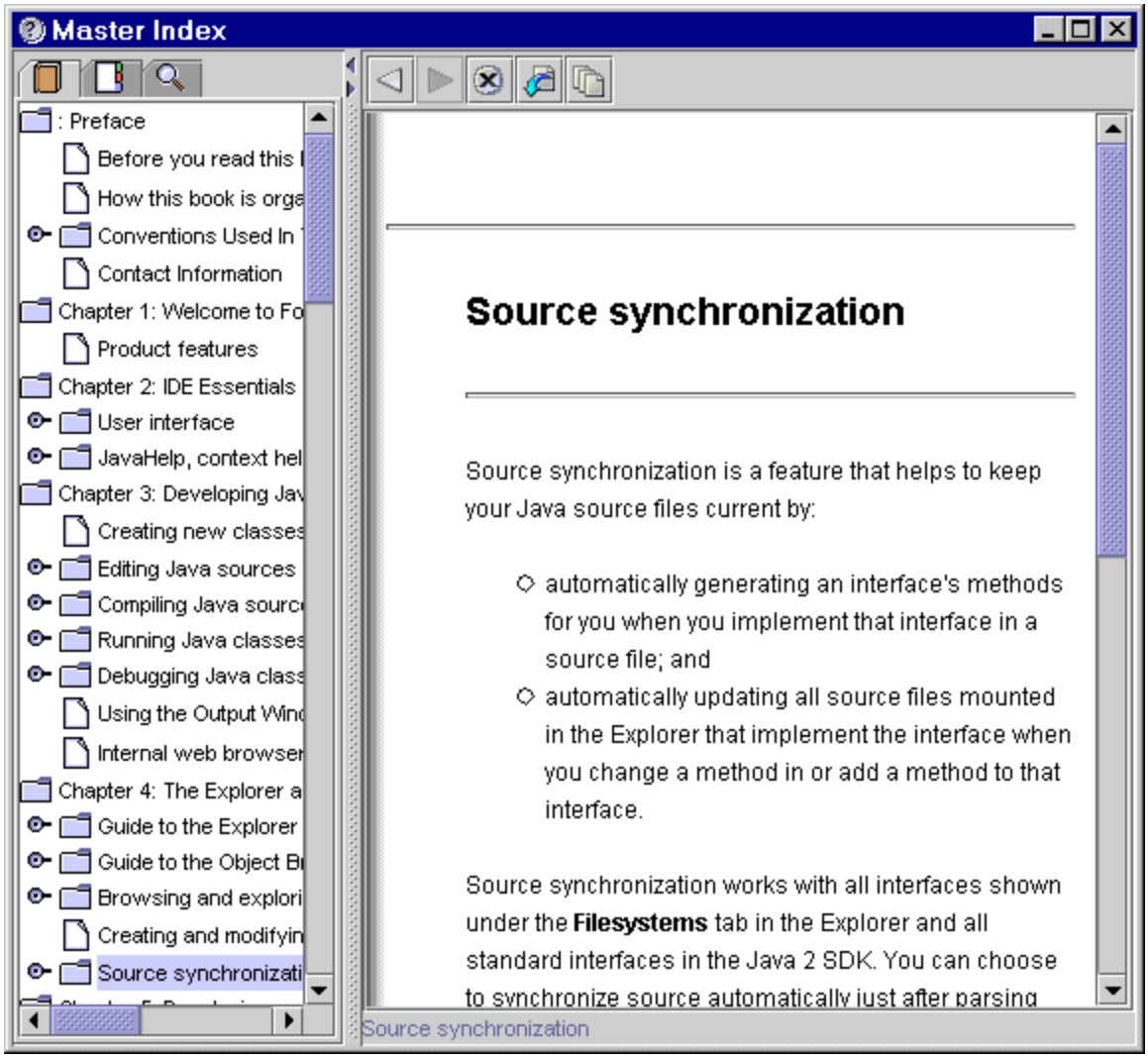
As you are getting to know Forte for Java, you may have questions about certain parts of the IDE. Forte for Java's implementation of JavaHelp (with search capability) as well as its context help and tool tips can provide answers to many of your questions.

JavaHelp

You can view the User's Guide through a JavaHelp browser by choosing **Help | Documentation | User's Guide** from the main menu. The JavaHelp viewer is divided into two panes. The left pane lists topics and the right pane provides the content of the topic selected in the left pane.

The toolbar at the top of the left pane has three icons which enable you choose what is shown in the topics list. Clicking the first button displays the User's Guide's table of contents, and clicking the second icon displays the index. The table of contents appears as a tree with

expandable nodes and sub-nodes.



To search the documentation, click the third icon and enter your search string in the **Find** field that appears just under the icons. JavaHelp will search the entire User's Guide for that string and then present you with a list of topics pertaining to that string. You can then select one of those topics to have its contents displayed.

Note: The JavaHelp search engine is context-sensitive.

Context Help

You can also obtain help for specific features of the IDE by pointing to a specific window, dialog box, or icon with the mouse cursor and pressing F1. You can obtain context help for menu items and nodes (such as nodes in the Explorer and Debugger windows) by selecting the node and pressing F1. For menu items, hold the cursor over the menu item so that it is

selected, but do not click or release the mouse button to select the command.

Most dialog boxes have a **Help** button which you can press to access context help.

Tool tips

Tool tips are panels with short text descriptions that appear when you briefly hold the mouse cursor over a part of the IDE. They are particularly useful for explaining the use of tree nodes and individual properties listed in the property sheet.

QuickStart and Tutorials

You can view the GuickStart Guide as well as tutorials for Forte for Java, Community Edition in the internal web browser. From the main menu, choose **Help | Getting Started** or **Help | Tutorials**.

Installation Guide

Besides installation instructions, the Installation Guide provides an overview of directories and files included in the installation directory as well as a list of switches that you can use to do things such as add class paths, set the IDE font size, and change the look and feel.

Browsing the documentation with an external web browser

HTML copies of the User's Guide, Getting Started, and the Tutorials can be found in the `docs` folder of the installation directory. You can open these files in an external web browser.

Printing the user's documentation

Copies of the User's Guide, the QuickStart Guide, and the Tutorials for Forte for Java, Community Edition are included with the IDE in PDF format. You can find them in the `docs` subdirectory of the IDE's installation directory. Using Acrobat Reader, available free from Adobe Software, you can view and print these files.

Chapter 3

Developing Java Applications

This chapter explains how to use the Forte for Java, Community Edition IDE to create and run applications. Specifically, we'll show you how to generate and edit Java objects and code using the Editor, Object Browser, Explorer, and templates. You will also learn how to compile, run, and debug your applications as well as create JavaBeans components and add Javadoc comments to your code. A separate chapter is devoted to using the Form Editor to design visual applications – see "Designing Visual Classes with the Form Editor" on page 111.

Creating new classes

In Forte for Java, Community Edition, you create new classes (as well as other objects) with templates. The template serves as a skeleton for the class and includes basic source code for that class. If you prefer to write all of the code yourself, you can choose the `Empty` template, and a file will be generated with the only code being the name of the package where you have created the template. A wide range of templates come with the IDE, and you can also create

your own. For more on templates, see "Using templates" on page 200.

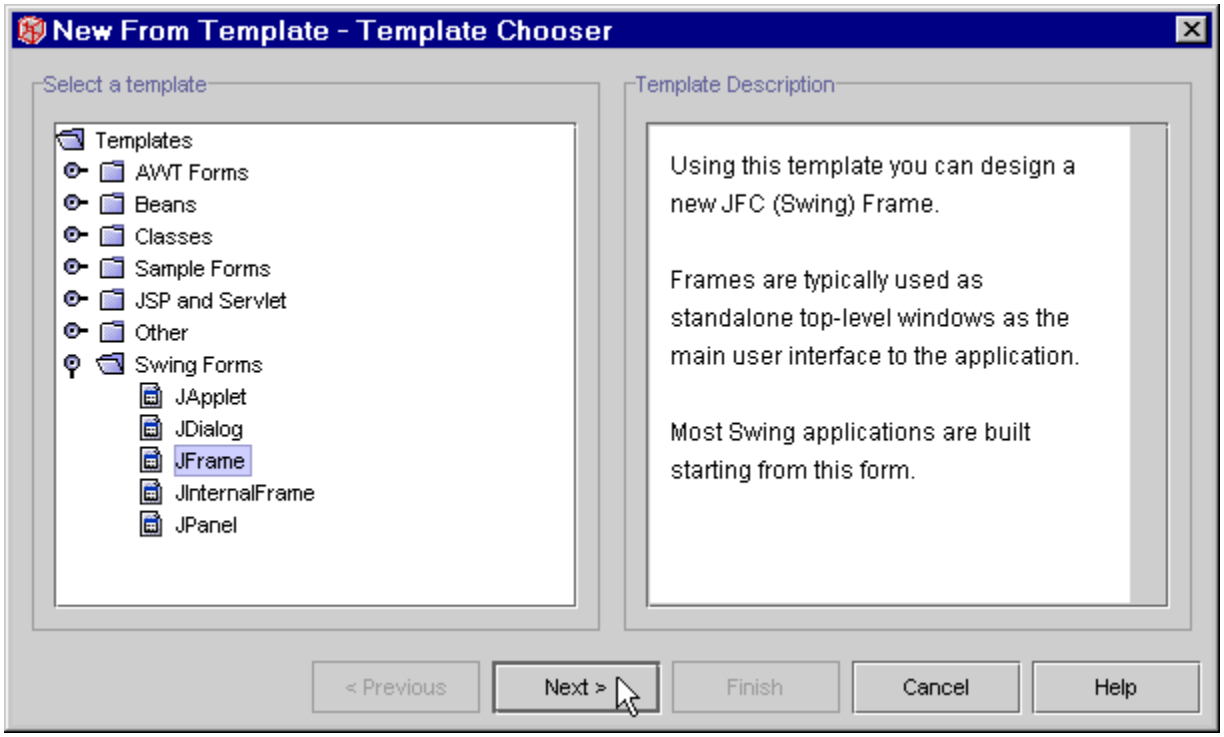
Classes are created using the New From Template wizard. You can access this wizard from the Main Window, the Explorer, and the Object Browser. When you access the wizard from the Main Window, you select the type of object you want to create from the first page of the wizard, which also provides short descriptions of each template. When you access the wizard from a contextual menu in the Explorer or the Object Browser, you skip straight to the second page of the wizard. In addition, the package information is automatically entered for you, based on where you right-clicked.

To create an object through the Main Window:

1 Choose **New From Template...** in one of the following ways:

- from the **File** menu
- from its toolbar icon
- using the CTRL+n keyboard shortcut

2 In the New From Template wizard that appears, expand the appropriate template category node (for example, **Classes**) and then select one of the templates. In the right panel of the dialog, a description of that template will appear.

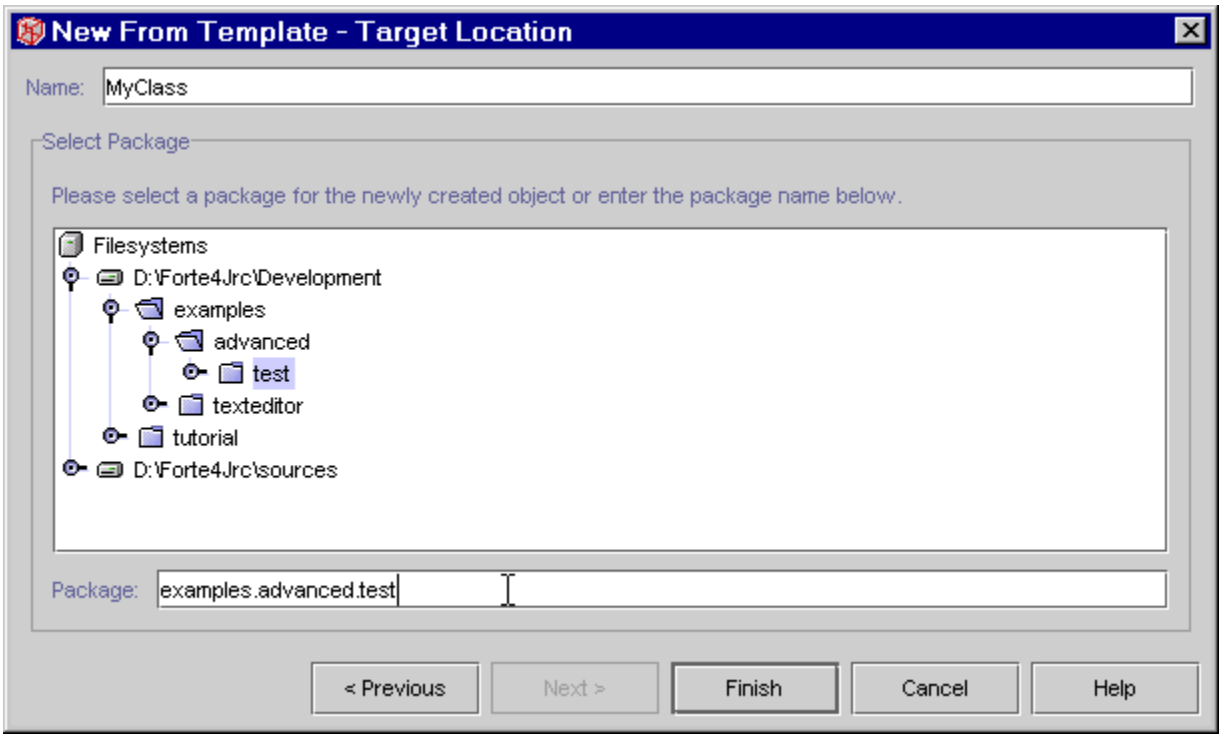


Press **Next** to move to the Target Location page of the wizard.

3 In the **Name** field, type the name you want to give the object. Do not type the extension for the file type. It will be added automatically.

4 In the same page of the wizard, choose a package for the template. You can do this either

by typing a package name in the **Package** field or by selecting one of the packages shown in the tree view just below it.



5 Press **Finish** to exit the wizard. The object will then be created.

Note: The first time you create an object from template, you will be prompted whether or not you want to add the file to the current project.

If you select **Yes**, the new file will be visible under both the **Filesystems** and **Project** tabs of the Explorer and will be among the classes affected if you compile by project.

If you select **No**, the new file will not be visible under **Project**.

If you select **Always**, this new class and all subsequent classes created from the wizard will be added to the current project. If you select **Never**, classes will not be automatically added to the current project unless you are adding them to a directory that is already included in the project.

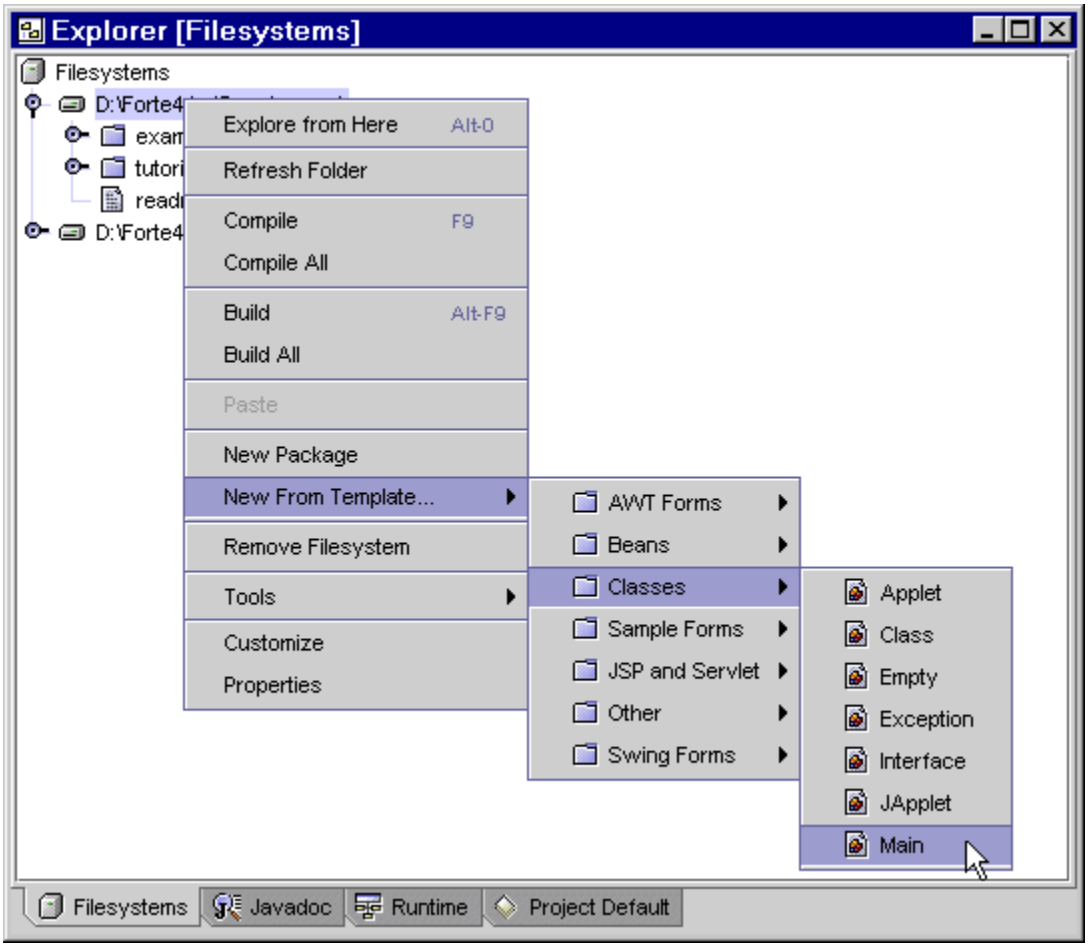
You can later add or remove files from projects and change whether files are automatically added to the current project. See "Organizing work into projects" on page 153 for a complete description of project functionality.

To create an object from the Explorer:

1 If the Explorer is not open, choose **Open Explorer** from the **File** menu or toolbar.

2 Find the package (marked with a folder icon) under the **Filesystems** tab in the Explorer where you want to place the class and right-click on it to bring up its contextual menu.

3 Choose **New From Template**, the template type from the first submenu, and then the template itself from the second submenu.



The Target Location page of the New From Template wizard will appear.

4 Type the name of the object in the **Name** field, verify that the correct package is selected in the **Package** panel, and click **Finish**.

To create an object from the Object Browser:

1 Open the Object Browser by selecting the **Browsing** tab in the Main Window to open up the Browsing workspace, or choose **File | Object Browser** from the main menu.

2 In the **Packages** pane of the Object Browser, right-click on the package where you want to place the class and right-click on it to bring up its contextual menu.

3 Choose **New From Template**, the template type from the first submenu, and then the template itself from the second submenu.

The Target Location page of the New From Template wizard will appear.

4 Type the name of the object in the **Name** field, verify that the correct package is selected in the **Package** panel, and click **Finish**.

Once you have created the class, the Editor window (or a tab in the Editor window if the Editor is already open) will open up and display the skeleton code for that class already generated. If the class you have created from the template is a visual form, the Form window and Component Inspector will also open. The new class will also be automatically added to the Explorer's tree and the Object Browser (see "Guide to the Object Browser" on page 86).

You can now edit your new class, either directly in the Editor window (see "Editing Java sources" below) or by using customizer dialog boxes to have the dialog generate various class elements, such as methods, constructors, and variables.

Note: Unlike other IDEs, the "project" is not the central paradigm for developing applications in Forte for Java. All objects developed in the IDE as well as any other file systems mounted in the IDE are all accessible from the Object Browser or from the **Filesystems** tab in the Explorer. However, it is also possible to organize files into projects to make it more convenient to compile or run them and to store different Project Settings for various applications. See "Organizing work into projects" on page 153.

Editing Java sources

The Editor is a full-featured text editor that is integrated with the Form Editor, Explorer, Compiler, and Debugger. It is the default viewer for all Java, HTML, and plain text files as well as any other types of files specified by installed extension modules. Each of these open files is displayed as a page in a multi-tab window – a single window with multiple tabs enabling you to choose which page to view in the window by clicking one of the tabs. Advanced features include customizable abbreviations and dynamic Java code completion.

Integration with other components

The Editor is integrated with the following parts of the IDE:

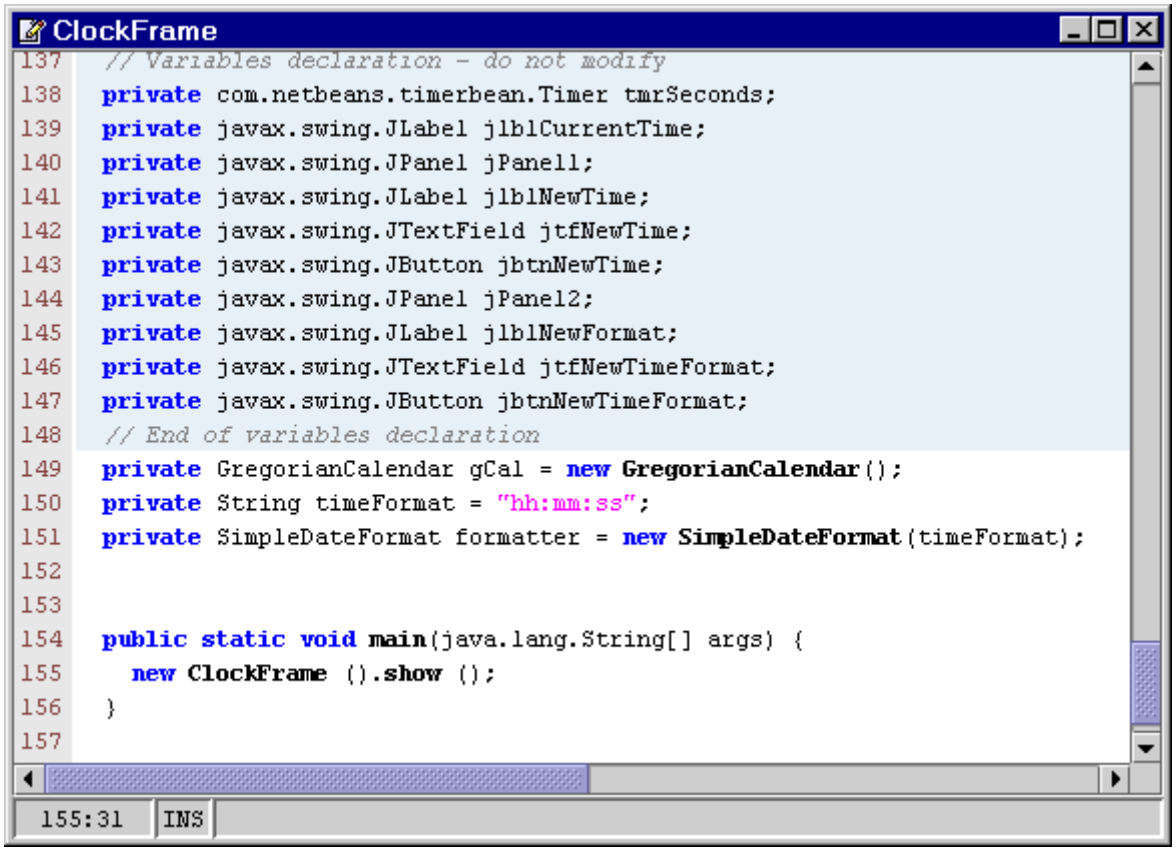
- **Form Editor** – All changes made in the Form Editor window are reflected in the source code in the Editor window.
- **Explorer, Object Browser, and Properties window** – Changes in properties and creation of new classes, methods, and variables, and so on done in these windows are reflected in the Editor window. You can also open files in the Editor from the Explorer and Object Browser.
- **Debugger** – When a program stops during execution, the Editor automatically jumps to the breakpoint where the code was interrupted. You can add or remove breakpoints with CTRL+F8.

- **Compiler** – If there is a compilation error, the cause of the error will be highlighted in red in the Output Window; press ENTER or double-click on that line to jump to the line with the error in the Editor.

The Editor window and its layout

The Editor contains source code that is syntactically colored. Different colors signify different text properties. For example, by default all keywords are shown in blue and all comments in light gray. **Guarded text** generated by the Form Editor has a blue background by default and cannot be edited.

A faint vertical line in the Editor marks a suggested right margin for your text. By default, it is set at 80 characters from the left margin. You can adjust this setting as well as its color in the Global Options window on the **Expert** tab of the `Editor Settings / Java Editor` property sheet. See "Editor settings by type of editor" on page 266.



The bottom of the Editor window has one or more tabs used to view different documents. From each of these tabs, the document can be saved, closed, docked or undocked, and cloned. For more details on managing windows, see "Window management" on page 187.

Opening files in the Editor

Double-click a Java or text object in the Explorer or the Object Browser to open the Editor (or select the node of the object and press ENTER). Any files that you subsequently open will also appear in the Editor with their own separate tabs in the bottom of the window. The tab of the currently visible file is highlighted. To flip between displayed files, simply click the tab of the file you want displayed or use the Alt+LEFT and Alt+RIGHT keyboard shortcuts.

Asterisks in Editor window tabs

A modified file in the Editor window is marked with an asterisk (*) after its name on its tab. (The asterisk also appears after the file name in the window title.) If there are any unsaved modifications when the Editor is closed, a prompt will appear to save or discard changes, or cancel the operation.

Editor window contextual menu

You can access many of the commands you need in the Editor window by right-clicking and choosing the command from the contextual menu. The following commands are available when you are editing a Java file:

- **Show Javadoc** – assuming that the proper Javadoc directory is mounted under the **Javadoc** tab of the Explorer, opens the web browser on the Javadoc documentation for the identifier that the insertion point is on (or immediately before or after).

- **Go to Source** – opens the source file for the the identifier that the insertion point is on (or immediately before or after) and moves the insertion point to the relevant section of code. For this command to work, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source. See "Adding your own classes to the parser database" on page 43.

- **Go to Declaration** – moves the insertion point to the declaration for the method or field that the insertion point is on (or immediately before or after). For this command to work, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source. See "Adding your own classes to the parser database" on page 43.

- **Reformat code** – regenerates the automatic code formatting for the current file (or for the selected text, if any text is selected).

- **Save** – saves the current source file.

- **Compile** – compiles the current source file.

- **Execute** – executes the current source file.
- **Add/Remove Breakpoint** – sets the current line as the breakpoint, or removes the breakpoint if the line is already a breakpoint.
- **Add Watch...** – opens the Add Watch dialog, enabling you to set a watch for debugging.
- **Cut** – removes the selected text from the file and pastes it to the clipboard.
- **Copy** – copies the selected text to the clipboard.
- **Paste** – pastes text from the clipboard to the insertion point in the file.
- **Delete** – deletes the selected text.
- **New | Initializer** – inserts a new initializer at the insertion point.
- **New | Field.../Constructor.../Method.../Inner Class.../Inner Interface...** – opens up a Customizer dialog enabling you to create a new class or element.
- **Tools | Auto Comment...** – opens the Javadoc Auto Comment Tool, which enables you to generate and modify Javadoc documentation for the classes in the source file.
- **Tools | Set As Project Main Class** – sets the current file as the project's main class.
- **Tools | Add to Project** – adds the current source file to the current project.
- **Tools | Create Group...** – creates a group object and adds the current source file to the group.
- **Tools | Add To Component Palette...** – adds the source file (preferably a JavaBeans component) to the Component Palette, from where you can use it in other applications.
- **Customize** – opens up the Customizer dialog box for the class or element that the insertion point is on. If the insertion point is not on an identifier that can be customized, the Customizer appears for the main class.
- **Properties** – opens a separate Properties window displaying the the property sheet for the source file or the class or class element that the insertion point is on.

Mouse and clipboard functions

The Editor uses standard mouse functions: click and drag the left mouse button to select a block of text, double-click the left button within a word to select that word, and triple-click to select an entire line. You can also select a block of text by clicking to place the insertion point where the block should begin, holding down SHIFT, and then clicking to determine the end of the selection.

You can select text and move it to and from the clipboard using the **Cut**, **Copy**, **Paste**, and **Delete** commands, which are available in the **Edit** menu, on the **Edit** toolbar, and in the contextual menu (accessed by right-clicking on the selected text). You can use **Undo** to reverse the previous command and **Redo** to reverse the reversal. These commands are available in the **Edit** menu, on the **Edit** toolbar, in various contextual menus, and by using keyboard shortcuts.

Editor abbreviations

To simplify editing of files, a set of customizable abbreviations is built into the Editor, which can be used to expand a few pre-defined characters into a full word or a phrase. Defining abbreviations is especially useful for long and frequently-used Java keywords. For example, if you type `pu` and press SPACE, the text will be expanded to `public`. To enter characters without expansion, type SHIFT+SPACE – this enters a space without checking for abbreviations. See "Default Editor Abbreviations" on page 241 for a complete list. See "Customizing Editor abbreviations" on page 211 for information on changing abbreviations and setting your own.

File navigation features

Forte for Java has many features that make it easier to navigate files and select text using only the keyboard.

Standard navigation and text selection shortcuts

When typing in the Editor, you have access to a wide variety of commonly-used shortcuts to help you navigate and select specific blocks of text in your files. These include:

- SHIFT+RIGHT – select the character to the right of the insertion point
- SHIFT+LEFT – select the character to the left of the insertion point
- CTRL+RIGHT – move the insertion point one word to the right
- CTRL+LEFT – move the insertion point one word to the left
- CTRL+SHIFT+RIGHT – select the word to the right of the insertion point
- CTRL+SHIFT+LEFT – select the word to the left of the insertion point
- SHIFT+PgDown – select the page-long block of text beginning with the insertion point
- SHIFT+PgUP – select the page-long block of text ending with the insertion point

See “Navigation shortcuts” on page 233 for a complete list

Text scrolling shortcuts

These shortcuts enable you to scroll the text without moving the insertion point:

- CTRL+UP – Scroll the text one line up while holding the insertion point in the same position.
- CTRL+Down – Scroll the text one line down while holding the insertion point in the same position.
- ALT+t – scrolls the text up so that the insertion point moves to the top of the window while remaining at the same point in the text.
- ALT+m – scrolls the text so that the insertion point moves to the middle of the window while remaining at the same point in the text.
- ALT+b – scrolls the text down so that the insertion point moves to the middle of the window while remaining at the same point in the text.

If you add SHIFT to any of the last three key combinations in this list, the insertion point moves to the top/middle/lower part of the window and the text does not scroll.

Navigating files with the jump list

You can move through a file or files by using keyboard shortcuts to jump straight to parts of the file where you have been previously working.

The Editor maintains a jump list where each entry represents a point in a file where you have added or deleted text or which you have navigated to with the find function. You can use the following keyboard shortcuts to navigate to points in your files according to the jump list:

- ALT+k – go to the previous entry in the jump list.
- ALT+l – go to the next entry in the jump list
- ALT+SHIFT+k – go to the next jump list entry *not* in the current file
- ALT+SHIFT+l – go to the next jump list entry *not* in the current file

For example, if you are editing line 7 of a file, then you scroll to line 100 and enter text, you can press ALT+k to go back to line 7 and then ALT+l to go back to line 100.

Using Editor bookmarks

You can set up Editor bookmarks (not be confused with bookmarks for web pages) to mark places in files that you return to often. You can then navigate to these bookmarks with a simple keyboard shortcut.

To bookmark a line in a file:

◊ Put the insertion point in the line you want to bookmark and press CTRL+F2.

The line will then be highlighted turquoise.

To jump to a bookmark:

◊ Press F2.

The insertion point will move to the next bookmark in the file. If the end of the file is reached without a bookmark being found, the search will continue from the top of the file.

To remove a bookmark:

◊ Put the insertion point in the bookmarked line and press CTRL+F2.

Find and replace

To find or replace text in a file open in the Editor window, press CTRL+f to bring up the Find dialog box, or press CTRL+r to bring up the Replace dialog box. The Find/Replace dialog box gives you checkboxes which enable you to choose any combination of the following options:

- **Highlight Search** – to highlight all occurrences of the search text in the file
- **Incremental Search** – for the search engine to try to find the text as you type (without having to press the **Find** button)
- **Match Case** – to limit the search to text that has the same capitalization
- **Smart Case** – to limit the search to text that has the same capitalization when at least one character of the search text is upper case
- **Match Whole Words Only** – to match the search text only to whole words in the file
- **Backward Search** – to search in reverse order in the file
- **Wrap Search** – to continue the search at the beginning (or end) of the file

You can also search using the following keyboard shortcuts:

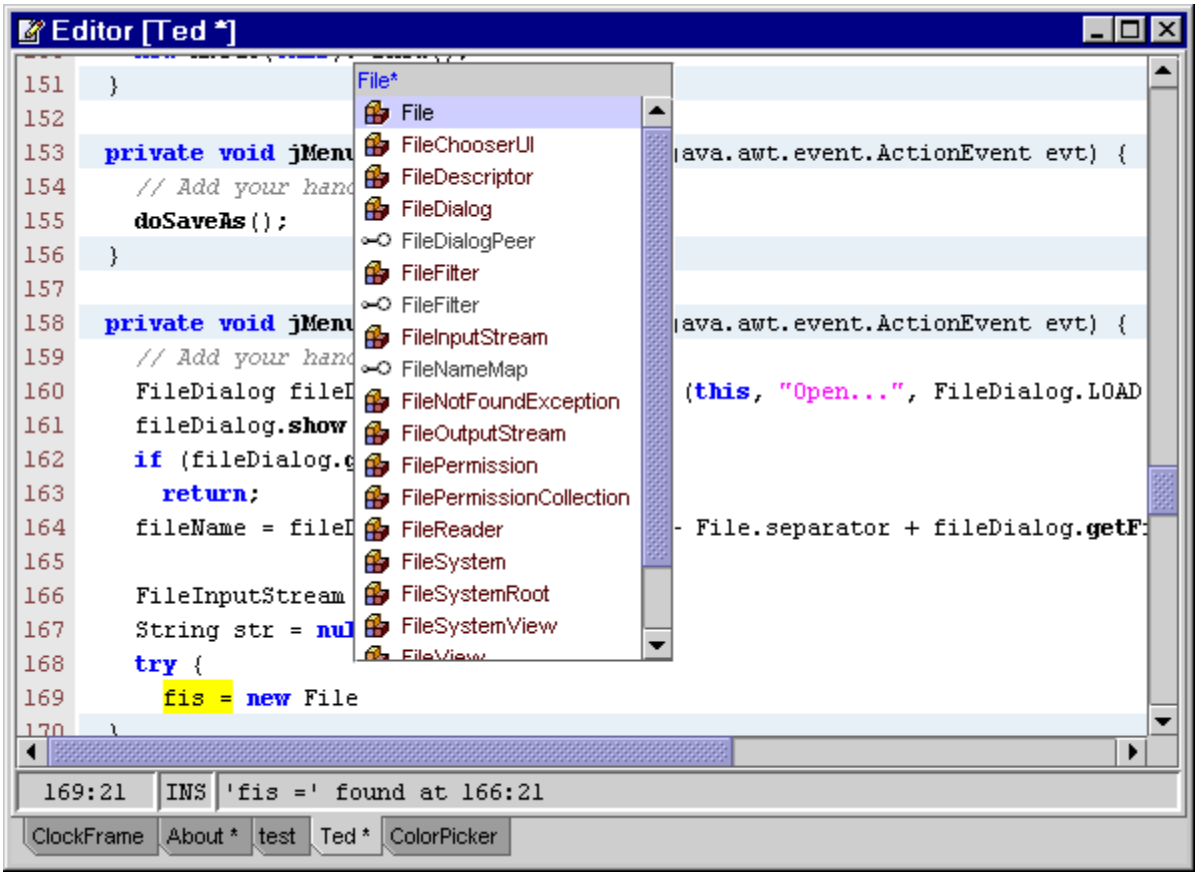
- F3 – to find the next occurrence of the search text
- SHIFT+F3 – to find the previous occurrence of the search text
- CTRL+F3 – to find the next occurrence of the selected string or, if nothing is selected, the word that the insertion point is in
- ALT+SHIFT+h – to switch the highlight search feature on or off

Java code completion

Forte for Java, Community Edition also has a dynamic code completion feature, where you can type a few characters and then bring up a list of possible classes, methods, variables, and so on that can be used to complete the expression.

To use Java code completion:

1. Type the first few characters of the expression (for example, `import javax.` or `someFile.getP`).
2. Press CTRL+SPACE (or pause after entering a period). The code completion box will then appear.



3. Then use the most convenient combination of the following steps:
   a. keep typing to narrow down the selection of items in the list.
   b. use the navigation keys (UP arrow, DOWN arrow, PAGE-UP, PAGE-DOWN, HOME, and END) or the mouse to scroll through the list and select an expression
   c. press ENTER to enter the selected method into your file and close the code completion box
   d. press TAB to select the longest common substring matching the text you have typed (Bash-style context-sensitive completion) and keep the list box open

If you press ENTER for a method with parameters, replaceable text is given for the first parameter which you can then fill in. If the method takes multiple parameters, you can bring the list box back by typing a comma after you fill in the parameter.

If the IDE recognizes what type of parameter is required, it's type is shown in the header of the completion window. If not, a question mark (?) is displayed.

If you enter a parameter that does not match any of the recognized parameter combinations for the method name, all the recognized methods of the given name and their parameter lists are displayed and an asterisk (*) appears instead of the parameter list in the header of the completion window.

When the code completion box is open, three other keyboard shortcuts are available:

- ALT+o – open the source file that the insertion point is on (or immediately before or after).
- ALT+g – go to the variable declaration for the identifier that the insertion point is on (or immediately before or after).
- ALT+F1 – open the web browser on the Javadoc file pertaining to the item that the insertion point is on.

For these shortcuts to work, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source. See "Adding your own classes to the parser database". In addition, for ALT+F1 to work, the Javadoc documentation for that source must be mounted under the **Javadoc** tab in the Explorer.

See "Customizing Java code completion" on page 212 for information on further options.

Adding your own classes to the parser database

The parser database can be found in the `System\ParserDB` folder in your installation directory. It consists of files with `.jcb` and `.jcs` extensions which are used whenever files are parsed in the IDE.

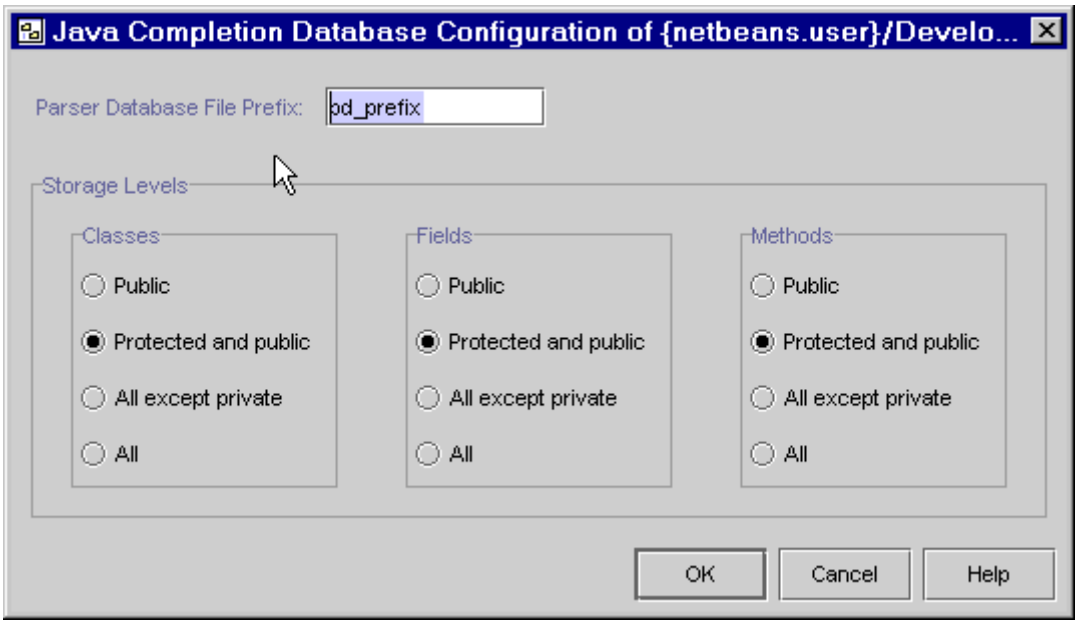
By default, the parser database contains files for the Java 2 SDK, v. 1.3. You can update the parser database so that your own classes are among the choices offered when using code completion and other special Java shortcuts in the Editor.

To update the parser database:

1. In the Explorer or Object Browser, right-click the package or file system containing the classes you want to add to the database and choose **Tools | Update Parser Database...** from the contextual menu.

   If you are updating the parser database for a package or file system that has already been entered into the database, the parser database will then be updated with any changes made to the classes.

2. If you are expanding the parser database with a package or file system that has not yet been entered in the parser database, the Java Completion Database Configuration dialog box will appear. Type a file name (with no extension) in the **Parser Database File Prefix** field. This will be used to create the `.jcb` and `.jcs` files.



3. In the **Storage Levels** panel of the same dialog box, select what level of code for classes, fields, and methods that you want included in the database file. By default, the **Protected and public** level is selected for all three categories.
4. Click **OK** to add the files to the parser database.

Other Java shortcuts

Besides Java code completion, there are few other shortcuts that are available only for Java files:

- ALT+o – open the source file that the insertion point is on (or immediately before or after). For this command to work, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source. See "Adding your own classes to the parser database" above.
- ALT+u then g – prefix the current identifier with `get`.
- ALT+u then s – prefix the current identifier with `set`.
- ALT+g – go to the variable declaration for the identifier that the insertion point is on (or immediately before or after). As with Alt+o, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source for this shortcut to work.
- CTRL+SHIFT+t – comment out the selected lines.
- CTRL+SHIFT+d – remove comment marks from the selected lines.
- CTRL+u – delete text in the following cycle (when using the shortcut successive times): first the text preceeding the insertion point on the same line, then the indentation on the line, then the line break, then the text on the previous line, and so on.
- ALT+F1 – open the web browser on the Javadoc file pertaining to the item that the insertion point is on. As with ALT+o and ALT+g, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source for this shortcut to work. In addition, the Javadoc documentation for that source must be mounted under the **Javadoc** tab in the Explorer.

When the insertion point is positioned after a brace, bracket, or parenthesis, you can also press CTRL+b to move to the matching bracket. If you press CTRL+SHIFT+b, the text within the braces is selected.

Word match

The word match feature enables you to type the beginning characters of a word used elsewhere in your code and then have the Editor generate the rest of the word.

To use the word match feature:

◊ Type the beginning of the word and press

a. CTRL+k to search backward through the text for a match, or

b. CTRL+l to search forward through the text for a match.

If a match is found, the rest of the matching word will be generated. You may type CTRL+k or CTRL+l multiple times to find additional matches in the document. The IDE searches for a match not only in the current file, but also in all other opened files (in the order that they

were last used).

Other Editing shortcuts

A wide range of keyboard shortcuts are available in Forte for Java to speed the editing process, including:

- CTRL+c (or CTRL+INSERT) – copy
- CTRL+x (or SHIFT+DELETE) – cut
- CTRL+v (or SHIFT+INSERT) – paste
- CTRL+w – delete previous word
- CTRL+e – delete current line
- ALT+j – select the current word (the word that the insertion point is on, immediately before, or immediately after) or deselect anything currently selected
- ALT+u then u – make the selection or the character after the insertion point uppercase.
- ALT+u then l – make the selection or the character after the insertion point lowercase.
- ALT+u then r – reverse the case of the selected text or the character after the insertion point.
- ALT+u then f – reverse the case of the first character of the identifier the insertion point is on.

For a complete list of shortcuts installed with Forte for Java, see "Editor Shortcuts" on page 233. See "Changing assignments for Editor keyboard shortcuts" on page 209 for information on setting your own key combinations for shortcuts.

Working with braces

In the Editor window, whenever the insertion point is immediately after a brace, bracket, or parenthesis (either opening or closing), the matching brace/bracket/parenthesis is highlighted.

When the insertion point is positioned after a brace, bracket, or parenthesis, you can also press CTRL+b to move to the matching brace. If you press CTRL+SHIFT+b, the text within the braces is selected.

Formatting code

The Editor provides several features that make it easier for you to create well-formatted and easy-to-read code.

When you type a line of code in the Editor and press ENTER, the next line is automatically indented according to its hierarchy in the code. However, you can easily modify the indents with keyboard shortcuts:

- CTRL+t – to increase line or block indentation.
- CTRL+d – to decrease line or block indentation

In both cases, if nothing is selected, the whole line is shifted one tab stop. If a block of text is selected, those lines are all shifted.

If you have made changes to the formatting of a file and would like to go back to the default formatting, use the Alt+f keyboard shortcut or right-click in the Editor window and choose **Reformat Code**. If no text is selected when you choose this command, the whole file will be formatted. Otherwise, the selected lines are reformatted.

There are several configurable options for formatting of automatically generated code. See "Java Editor formatting options" on page 212.

Editor Settings

The Editor has other configurable settings, such as for font and color settings, abbreviation tables, formatting of generated code, appearance of the insertion point, and so on. Many of these settings are separately customizable for Java, HTML, and plain text under their respective subnodes under `Editor Settings` in the Global Options window. See "Configuring the Editor" on page 209 for more information and "Editor Settings reference" on page 265 for a list of all configurable properties.

Compiling Java sources

Forte For Java, Community Edition offers a wide array of compilation options, from different ways to bring up the **Compile** command to the ability to use different compilers and set a specific compiler for each class.

Note: When you choose the Compile (or Compile All, Compile Project, Build, Build All, or Build Project) command for an object, the IDE (consistent with Java conventions) automatically compiles the first file it finds with the same name and package. Therefore, if

you have two files with the same file name and package hierarchy mounted in the Explorer, the file in the first package listed will be compiled automatically, even if you choose the Compile command with the second package selected.

After you initiate compilation of a class, the Output Window appears if there are any compilation errors. If you double-click on the error in the Output Window, the insertion point will jump to the line in the Editor with the error. See "Using the Output Window" on page 68 for more details.

Compilation progress is reported in the status line (next to the workspace tabs in the Main Window).

Compiling single classes

You can compile an object in the active Editor window tab or if selected in the Explorer by:

- choosing **Build | Compile** from the main menu; or
- clicking on the **Compile** icon on the main toolbar; or
- pressing F9; or
- right-clicking on the object in the Explorer and choosing **Compile** from the contextual menu.

Compiling packages

There are several options for compiling packages, all available from the **Build** menu and toolbar on the Main Window and the contextual menu for packages in the Explorer:

- Choosing **Compile** when a folder is selected compiles all sources in that folder which have been modified since they were last compiled or that have not been previously compiled.
- Choosing **Compile All** does this recursively on a folder and all its sub-folders.
- Choosing the **Build** command (also available with keyboard shortcut ALT+F9) is slightly different from compiling in that it forces re-compilation of all sources in a folder, whether they are current or not. Use this option when you wish to be sure that all of your code can compile together.
- Choosing **Build All** recursively builds a folder and all sub-folders.

Deleting .class files

The **Build** menu also has the commands **Clean** and **Clean All** which delete compiled classes. **Clean** deletes all `.class` files in the selected package and **Clean All** recursively deletes all `.class` files in the selected package and its sub-packages.

Built-in compiler support

The IDE comes with support for the Fastjavac and Javac compilers. The IDE uses Fastjavac by default.

Fastjavac is a native compiler, meaning that there are different versions of it for each platform, but each version compiles into the same Java bytecode. Fastjavac is available for the Solaris, Linux, and Windows platforms. See "Switching compilers by class" on page 50 for information on setting the compiler for your classes.

Javac is a cross-platform compiler written in Java.

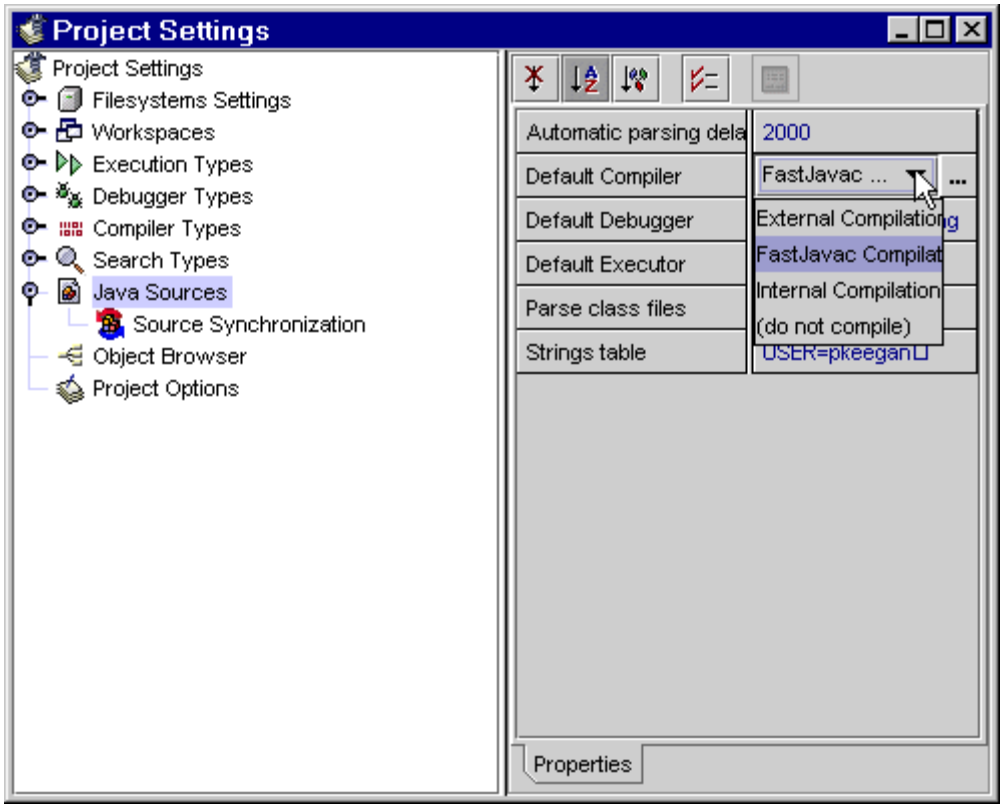
Switching the default compiler type

It is possible to switch the default compiler type. When you switch the default compiler type, this switch affects all classes and templates for which the user has not specifically assigned a compiler type

To switch the default compiler type:

1. Choose **Project | Settings...** from the main menu to open the Project Settings window.
2. Select the `Java Sources` node in the left pane of the Project Settings window.
3. Select the `Default compiler type` property and choose a new default compiler type

from the dropdown list.



Note: Once you change a class or template's compiler type, the IDE will never again recognize the class as using the default compiler type, even if you switch the compiler type for the class back to the one that is the IDE default. Therefore, if you change a class's compiler type and then change it back to the default compiler type, the class's compiler type will not be affected if you change the default compiler type.

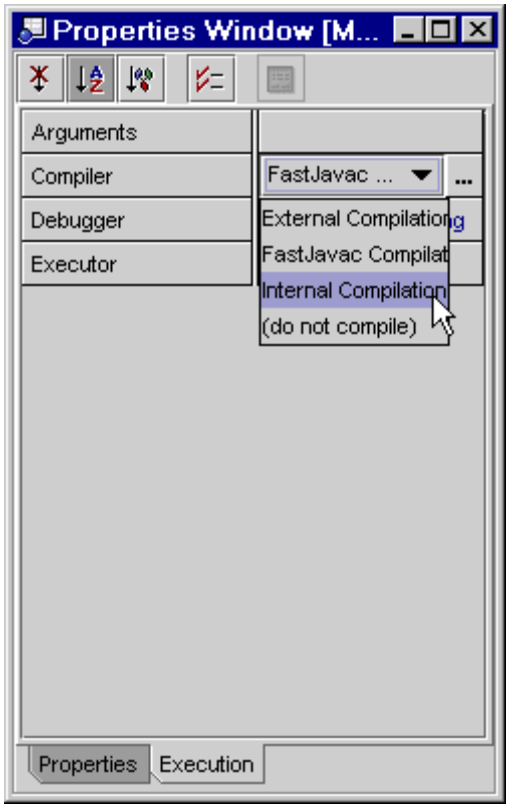
Switching compilers by class

A default compiler is set for each class. If you would like to use a different compiler (or a different configuration of a compiler) for that class, you can specify one on the class's property sheet.

To switch compilers for a class:

1. Select the object under the **Filesystems** tab in the Explorer.
2. Go to the object's property sheet (by choosing **View | Properties** from the main menu).
3. Click on the **Execution** tab in the Properties window.
4. Rotate through the compiler types by double-clicking on `Compiler`, or click on the

`Compiler` property's value and choose from the pull-down menu.



By default, there are three choices for Java sources: using Javac internally (in the same VM as the IDE), running Javac externally, and running Fastjavac. Other types of files have different choices.

Tip: To switch compilers for multiple classes simultaneously, select the classes while holding down the CTRL key (or the SHIFT key to select a range of classes) and then switch the compiler on the property sheet.

Disabling compilation for a class

If you have a source under the **Filesystems** tab in the Explorer that you specifically do not want to be subject to compilation, you can disable compilation for that class.

Important: Disabling compilation for a single class prevents the IDE from passing the name of the file to the compiler as a class to be compiled. However, if another class is dependent on that class, the compiler itself may include the "disabled" class in the compilation.

To disable compilation for a class:

1. Select the object under the **Filesystems** tab in the Explorer.
2. Go to the object's property sheet by choosing **View | Properties** from the main menu.

3 Click on the **Execution** tab in the Properties window.

4 Click on the `Compiler` property's value and choose `(do not compile)` from the pull-down menu.

Configuring compilers

It is also possible to customize the command-line template for the executable compiler, thus affecting the way the compiler is called. For more information, see "Adding and modifying service types" on page 204.

Setting compiler types in templates

If you use more than one compiler type in your work, you can create a set of templates with the different compiler types that you use. See "Creating your own templates" on page 201 and "Modifying existing templates" on page 202 for more information.

Running Java classes

Java applications may be run in several ways.

To run a Java application:

1 Make sure that the Java object is executable (that it either has a `main()` method or is a subclass of `Applet` or `JApplet`).

2 Right-click on it in the Explorer and choose **Execute** from the contextual menu.

Alternately, you can select the Java object in the Editor window and then run it one of the following ways:

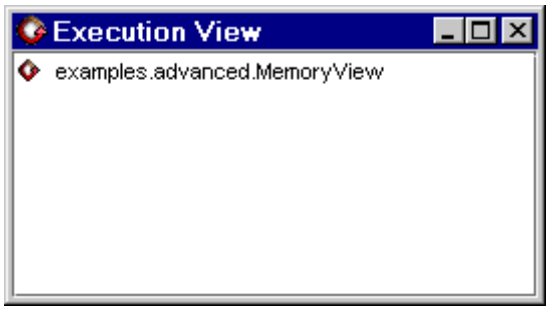
- Choose the **Execute** icon on the Main Window.
- Choose **Build | Execute** from the Main Window.
- Use the keyboard shortcut CTRL+F9.

When executing, the Java class is (by default) first compiled. After compilation completes successfully, the IDE switches to the Running Workspace (though you may configure it to do otherwise by opening the Project Settings window, selecting the `Execution Settings` node, and changing the `Workspace` property). You can return the IDE to the Editing or the GUI Editing workspace by clicking the tab for the workspace in the lower left part of the

Main Window. See "Workspaces" on page 188 and "Customizing workspaces" on page 222 for more information.

Execution View

The Execution View provides a view of all applications currently being run within the IDE. This window is actually a view of the Explorer hierarchy under `Runtime / Processes`.



By default, the Execution View is opened on the Running Workspace. When no applications are running, it simply displays `<No Processes Running>`. When an application or applet is running, it is listed by name. Each currently running process is listed. As for many objects in the IDE, a displayed process has a contextual menu. In this window, the menu contains just one item: **Terminate Process**. This enables you to force termination of the selected process.

Execution categories and executors

You can run typical Java applications using either internal or external execution, each of which have their advantages and disadvantages. Applets can be run using Applet Execution – see "Applet viewer settings" on page 57 for more information. Java Server Pages can be run using JSP Execution – see "Configuring JSP Execution Types" on page 151. Other execution categories can be installed by extension modules.

External Execution

Most applications use external execution, and the templates that come with the IDE use it by default.

A new virtual machine (VM) is used for executing the application. This enables you to run applications that require a special VM or need to do operations that are not possible with internal execution (see below). You can specify the VM executable (such as `java.exe`) and complete command-line parameters together with class path settings for the application. External execution also protects the IDE from application crashes and delays.

Internal (Thread) Execution

An application run using internal execution runs inside the Forte for Java IDE. This brings the advantages that the application can modify the IDE itself and be loaded faster. But it imposes at least two restrictions on the executed application. The application cannot install its own `URLStreamHandlerFactory` or `SecurityManager` (so you cannot run RMI applications, for example). In addition, if the executed application crashes, the IDE crashes with it. Go to the `Examples` directory under the **Filesystems** tab in the Explorer to look at some samples of internal execution applications.

Note: Some applications, such as startup classes (in the `Startup` folder which can be accessed by choosing **Tools | Global Options...** from the main menu) require the use of internal execution because they are intended to modify the IDE itself.

Other execution categories

Other execution categories tailored for specific types of applications are installed by various modules, such as RMI and JSP.

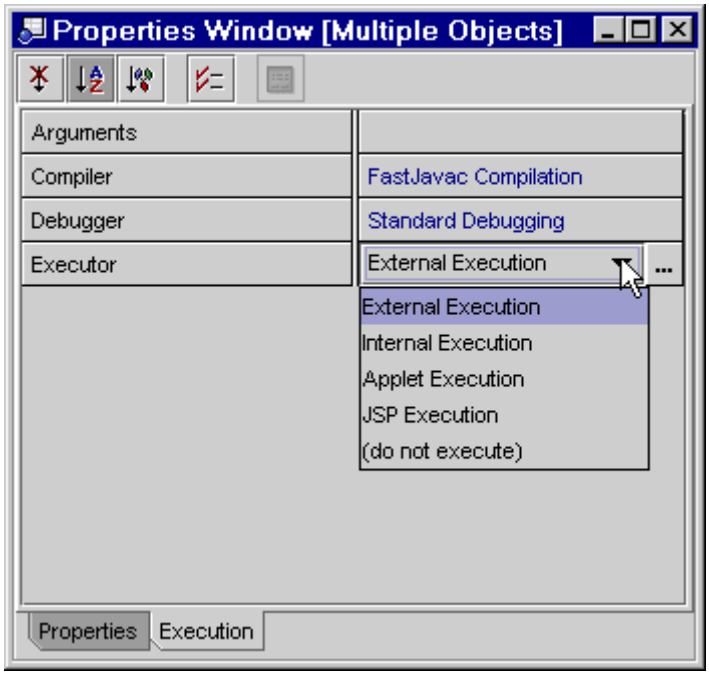
Setting execution

The execution category (for example, external, internal, or applet) is set for each separate object. When you set execution, you choose from a list of "executors", each of which represents a specific configuration (including the path to Java and arguments) of an execution category. There can be multiple executors for a given execution category, though the IDE comes with only one for most categories. See "Adding and modifying service types" on page 204 for more information.

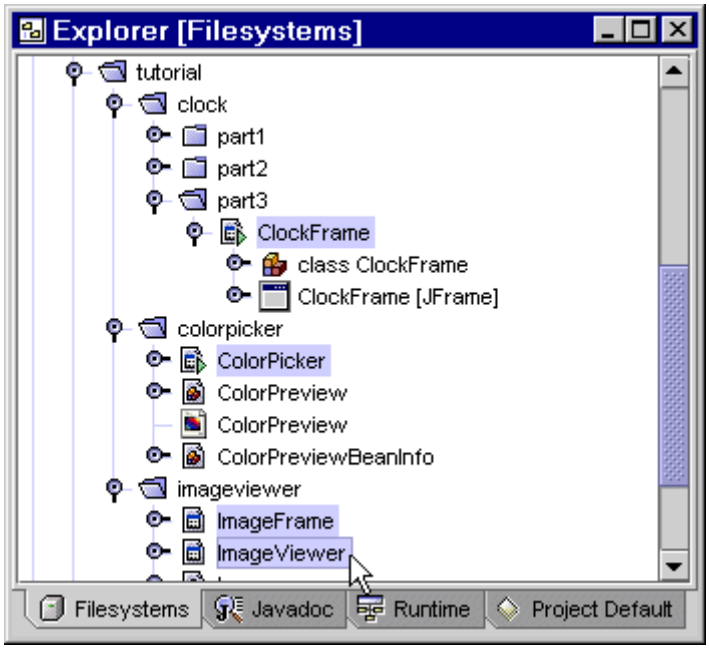
To switch an object's executor:

1. Select the object under the **Filesystems** tab in the Explorer.
2. Go to the object's property sheet (if it is not already open, choose **View | Properties** from the main menu) and click on the **Execution** tab.
3. Rotate through the executors by double-clicking on `Executor`, or click on the

`Executor` property's value and choose from the pull-down menu.



Tip: To switch executors for multiple classes simultaneously, select the classes while holding down the CTRL key and then switch the executor on the property sheet.



Switching the default executor

It is possible to switch the default executor. When you switch the default executor, this switch affects all classes and templates for which the user has not specifically assigned an executor

To switch the default executor:

1. Choose **Project | Settings...** from the main menu to open the Project Settings window.
2. Select the `Java Sources` node in the left pane of the Project Settings window.
3. Select the `Default executor` property and choose a new default executor from the dropdown list.

Note: Once you change a class or template's executor, the IDE will never again recognize the class as using the default executor, even if you switch the executor for the class back to the one that is the IDE default. Therefore, if you change a class's executor and then change it back to the default executor, the class's executor will not be affected if you change the default executor.

Configuring external executors

It is also possible to customize the command-line template for the executor, thus affecting the way the executor is called. For more information, see "Adding and modifying service types" on page 204.

Setting executors in templates

If you use more than one compiler type in your work, you can create a set of templates with the different execution types that you use. See "Creating your own templates" on page 201 and "Modifying existing templates" on page 202 for more information.

Passing command-line arguments to executed applications

To pass command-line arguments to executed Java applications:

1. Select the object in the Explorer.
2. Choose **Build | Set Arguments** from the main menu.
3. Enter the arguments in the dialog box, separated by spaces.

or

1. Select the object and open its property sheet (by choosing **View | Properties** from the main menu).
2. Click the **Execution** tab and type the argument in the `Arguments` property.

Note: This applies only to application arguments, not to Java virtual machine arguments (which must be configured on the object's executor – see "Adding and modifying service types" on page 204).

Execution Settings node

In the Global Options window is the `Execution Settings` node, where you can configure the IDE's behavior when running applications. The options include whether to automatically compile applications before execution and whether to create a new output tab each time you run a class. See "Execution Settings reference" on page 269 for more information.

Applet viewer settings

You can choose which viewer to use when running applets. You can use either:

- Sun's JDK utility AppletViewer, which is set by default.
- An external viewer such as Netscape Navigator or Microsoft Internet Explorer.

For security reasons, internal execution is not possible for applets.

To change the default viewer:

1. Choose **Project | Settings...** from the main menu.
2. In the Project Settings window, select `Execution Types / Applet Execution / Applet Execution`.
3. Click on the `External Viewer` property and type in the path and name of the browser or applet viewer.
4. In the same custom property editor, add any startup arguments that you require for the applet viewer.

To set up a viewer other than AppletViewer:

1. Right-click on `Execution Types / Applet Execution` in the Explorer.
2. Choose **New | Applet Execution Service** from the contextual menu.
3. A new node labelled `Applet Execution` (with a number in parentheses to give it a unique name) will appear. You can select the node and modify its `Identifying Name` property.
4. Follow steps 2 through 4 from the procedure for changing the default applet viewer.

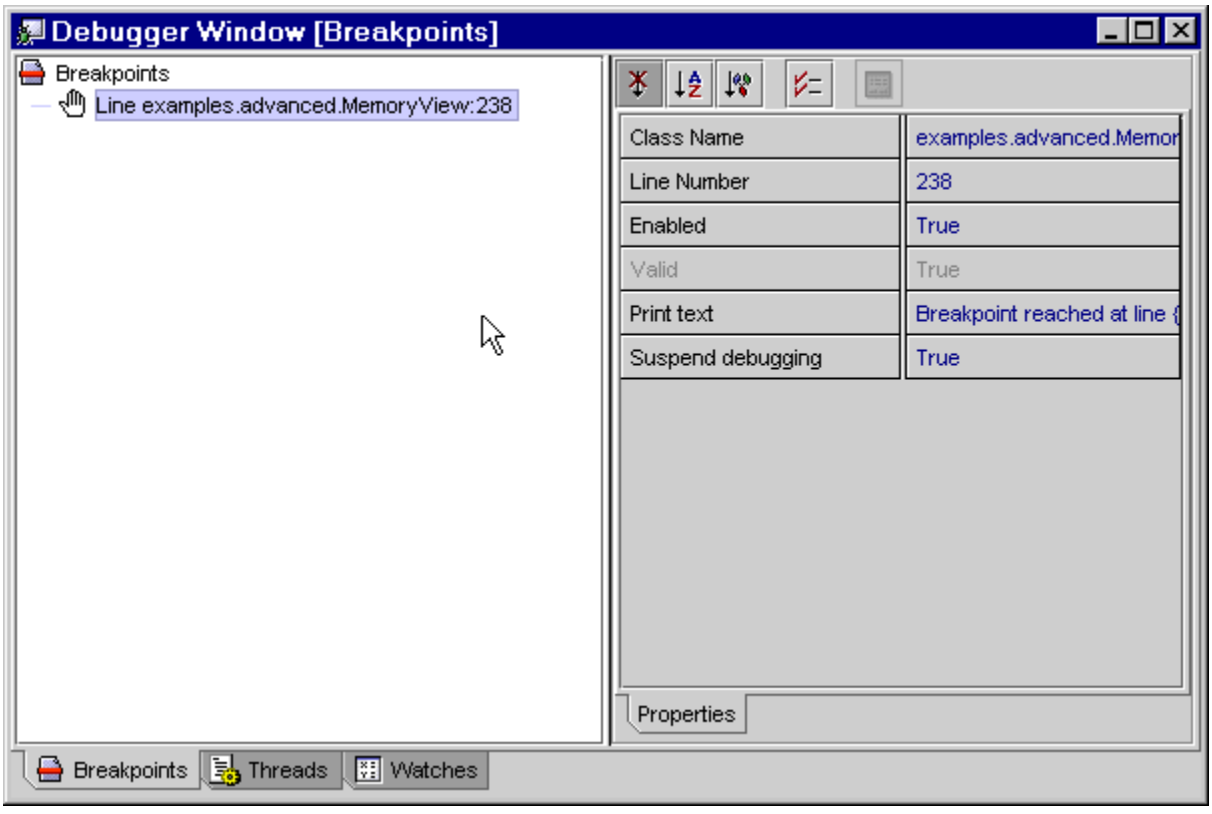
   You can now use this new executor for particular objects.

Debugging Java classes

The Debugger can be used to present "snapshots" of the system state during execution. By placing breakpoints at key positions throughout your source code, the Debugger can halt at these points and display details of the current environment at that point in the source. You can effectively step through your code, monitoring execution as it occurs. You can also connect the debugger to an already-running process.

Debugger Window

The Debugger Window is a three-tabbed display with tabs for **Breakpoints**, **Threads**, and **Watches**. In the right half of the window is the property sheet pane which displays the properties and their current values for the selected node in the left pane. By default, the Debugger Window opens (as part of the Debugging workspace) when you start debugging an application.



Note: This window is a separate view of debugging information available in the Explorer under `Runtime / Debugger` with the Properties window open.

Breakpoints

The **Breakpoints** tab simply lists the currently set breakpoints, showing the class name, and the line number or method on which the breakpoint has been set.

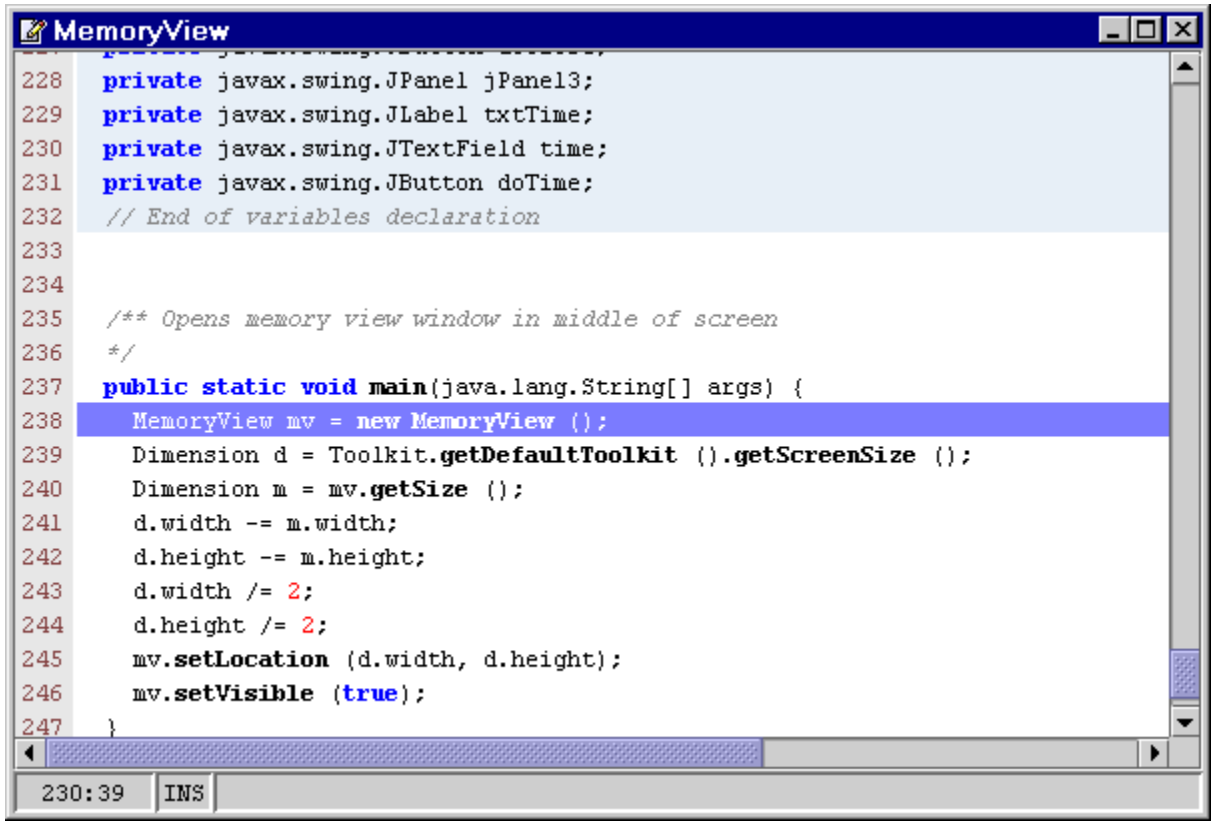
To add a new breakpoint to your code:

1. Position the insertion point at the desired line in the Editor window.
2. Choose **Add/Remove Breakpoint** from the **Debug** menu or toolbar in the Main Window, right-click on the line in the Editor and choose **Add/Remove Breakpoint** from the contextual menu, or use the keyboard shortcut CTRL+F8.

   The current line will be highlighted blue to indicate that the breakpoint has been set.

or

1. Choose **New Breakpoint** from the **Debug** menu or toolbar to bring up the Add Breakpoint dialog box.
2. Choose the type of breakpoint (either **exception**, **method**, or **line**) from the combo box.
3. Enter the settings (exception class name, class name and method name, or class name and line number).



Optional breakpoint settings

If you set the breakpoint using the **New Breakpoint** command, you have further options:

- If you want to be notified in the Output Window when the breakpoint is reached, check **Print text** in the Add Breakpoint dialog box. You can also set the text to be printed using a combination of plain text and these self-explanatory substitution codes: `{lineNumber}`, `{className}`, and `{threadName}`. In addition, you can use curly braces and a dollar sign to create a substitution code for a watch (for example, `{$mywatch}`).

- If you want to be notified in the Output Window when the breakpoint is reached, check **Print text** in the Add Breakpoint dialog box. You can also set the text to be printed using a combination of plain text and these self-explanatory substitution codes: `{lineNumber}`, `{className}`, and `{threadName}`. In addition, you can use curly braces and a dollar sign to create a substitution code for a watch (for example, `{$mywatch}`).

- Checking **Suspend Debugging** suspends the debugging session (all threads) when the breakpoint is reached.

You can also set these options (and later change them) in the property sheet for the breakpoint in the Debugger Window.

To remove a breakpoint:

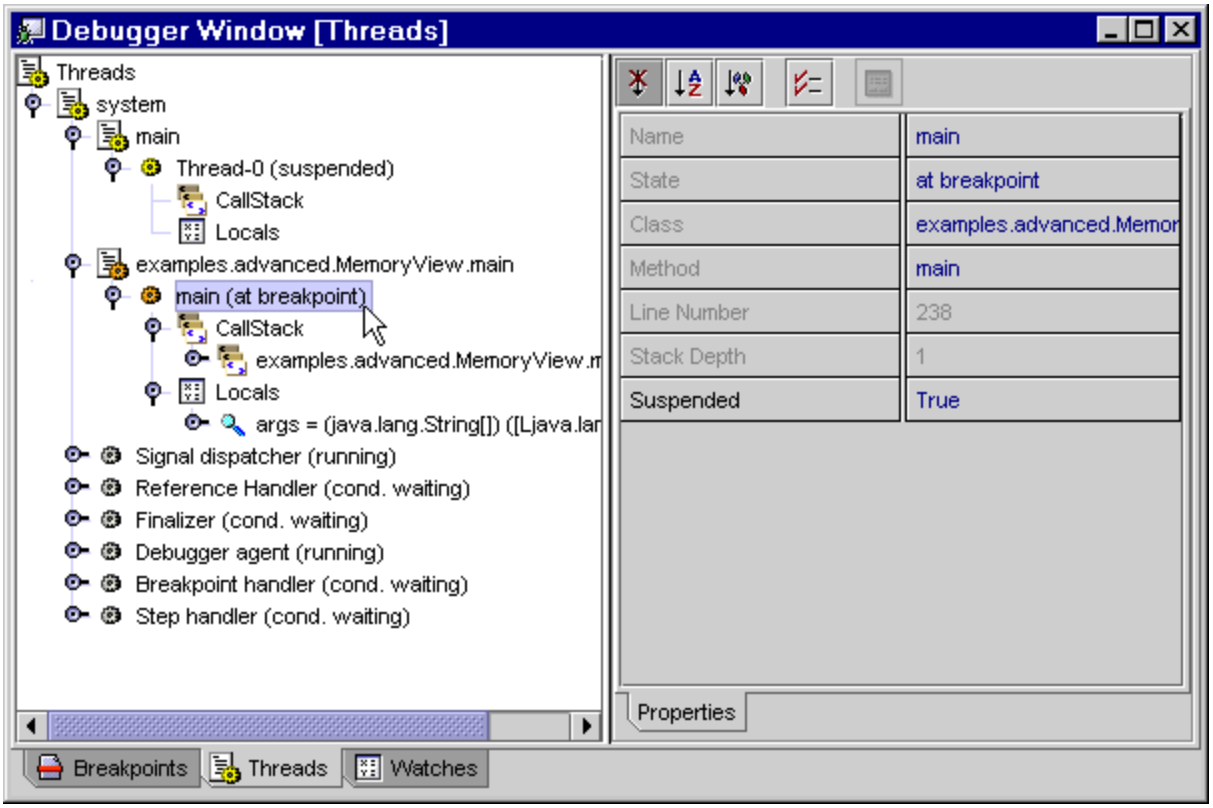
1. Position the insertion point on the line in the Editor window where the breakpoint has been set.
2. Choose **Debug | Add/Remove Breakpoint** from the Main Window or use the keyboard shortcut CTRL+F8.

   Breakpoints can also be removed directly from the Debugger Window by right-clicking on a listed breakpoint and choosing **Delete** from the contextual menu or by selecting the breakpoint and pressing DELETE on the keyboard.

Threads

The **Threads** tab displays all thread groups in the current debugging process. These thread groups are expandable hierarchies; each group containing other thread groups or single

threads, which in turn contain `CallStack` and `Locals` nodes.



When a thread is suspended:

- The `CallStack` node can be expanded to show the current hierarchy of method calls made during execution.
- The `Locals` node displays local variables and their current values in the context of the current thread. You can expand these nodes to see the object sub-structure.

If the process you are debugging has more than one thread, all threads and thread groups appear in the **Threads** tab showing a thread name and current status (such as "running", "at breakpoint", "cond. waiting"and "suspended"). Suspended threads and threads at breakpoint display all "current" system information.

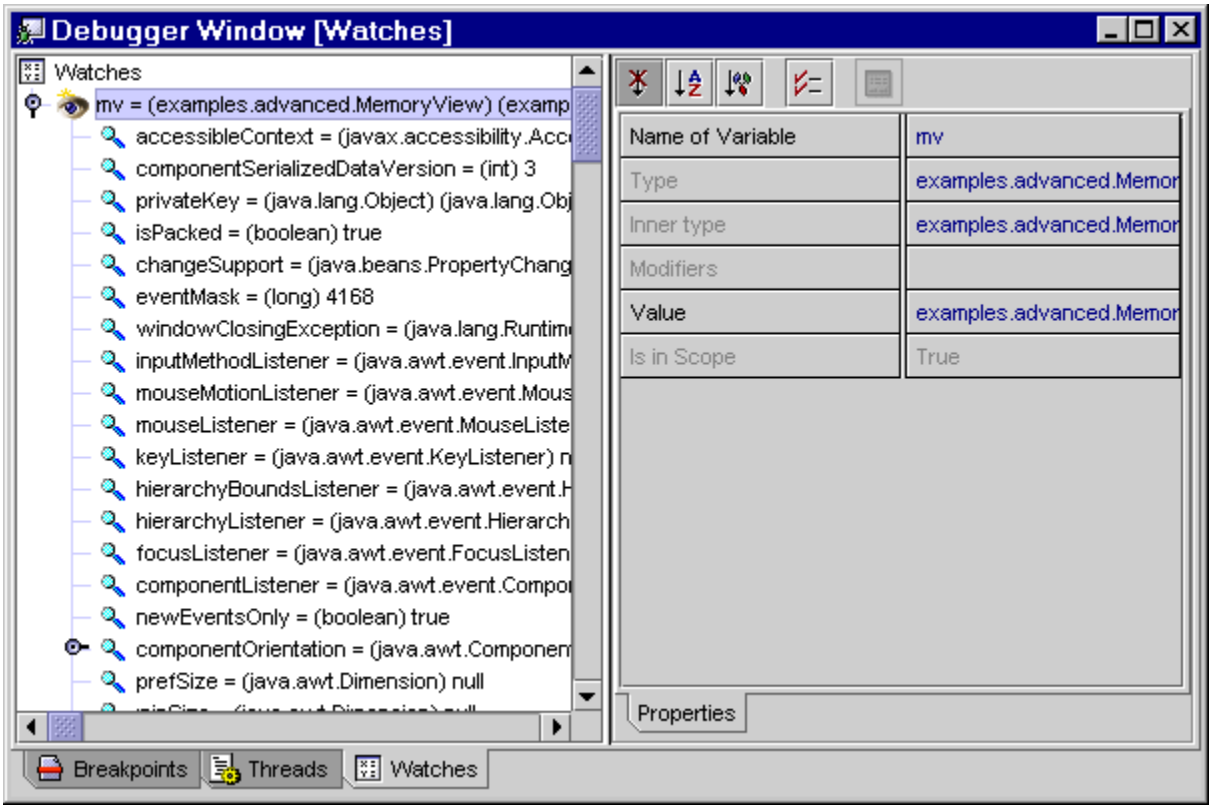
The Debugger Window displays the following properties for each running thread:

- `Name` – Thread name (according to the thread class).
- `State` – Status of the thread, such as `Running`, `Cond. waiting`, and so on.
- `Class` – Name of the class in which the thread is suspended.
- `Method` – Name of the method in which the thread is suspended.

- `Line Number` – Current line in the thread.
- `Stack Depth` – Number of methods in the call stack.
- `Suspended` – If `True`, the thread is suspended.

Watches

The **Watches** tab lists all currently set watches. A watch is a node you can create in the Debugger Window to display the current value of any variable of that name currently in scope. You can monitor the value of the variable as it changes during the execution of the program.



To set a watch:

1 Choose **Add Watch** from either

- The **Debug** menu on the Main Window.
- The contextual menu of the root `Watches` item on the **Watches** tab of the Debugger Window.
- From the contextual menu of a variable you have selected in the Editor (double-click the variable to select it, and then right-click to bring up the context menu).

A dialog box requesting the name of the variable to watch will open. Once you have entered the name of a variable in your source and clicked **OK**, it will be listed in the `Watches` tree.

2. Click on this item in the `Watches` tree to select it and display its property sheet.
3. Continue running the application and watch the variable change.

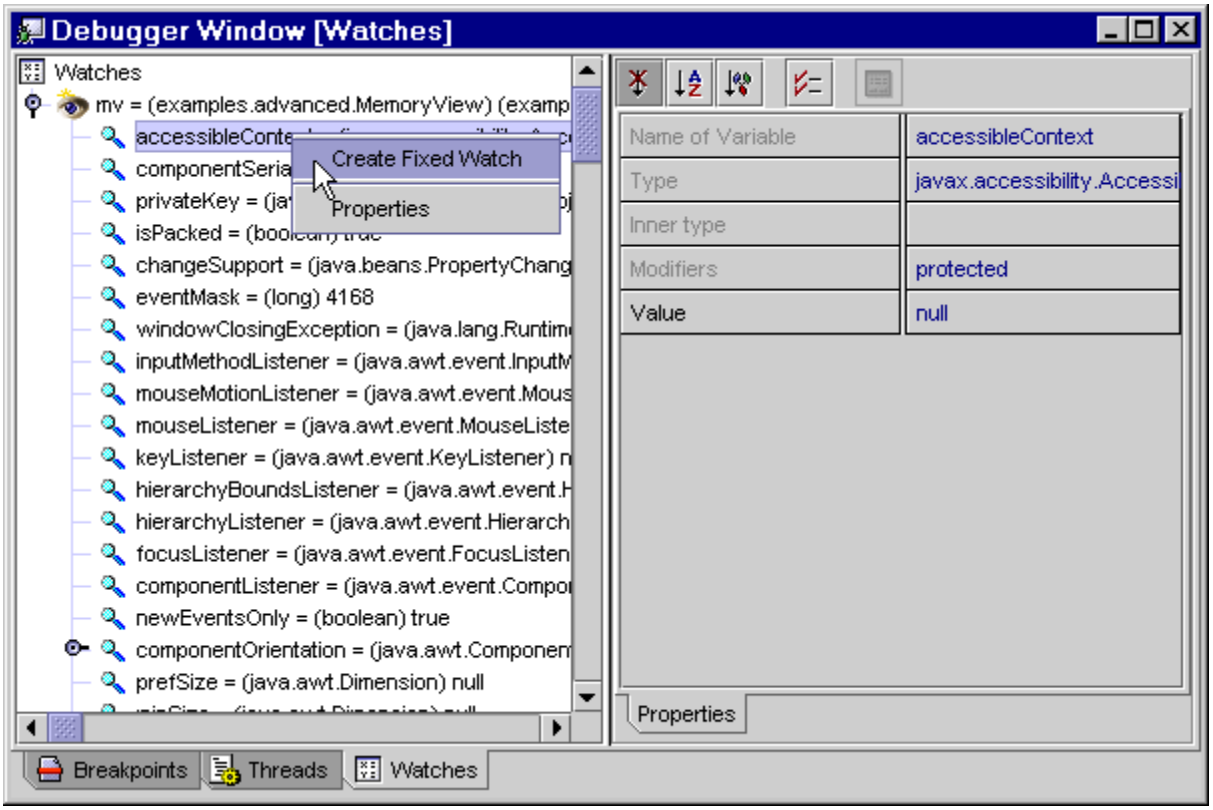
Fixed watches

Whereas a standard watch refers to the value of variable of that *name* currently in scope, it is possible to create a **fixed watch**, which always refers to the variable itself that it was created on.

To add a fixed watch:

◊ Right-click on the watch and choose **Create Fixed Watch** from the contextual menu.

A new node will appear in the Watches tree showing the value of that specific variable. If the context changes, and the new context contains a different variable of the same name, the watch will show the value of the variable currently in scope, and the fixed watch will show the value of the particular variable it was created on.



The debugging session

To initiate a debugging session:

1 Set a breakpoint and then choose **Debug | Start Debugging** from the Main Window (or press F5). (If you are debugging a GUI application or another looped application, it is not necessary to set a breakpoint before starting.)

By default, the IDE switches to the Debugging Workspace (to configure it to do otherwise, see "Customizing workspaces" on page 222), where the Debugger Window, the Editor with the source being debugged, and the Output Window all open up. The Output Window is split vertically, with Debugger output on the right and any output from the application being debugged on the left. When the Debugger reaches a breakpoint in your source, that breakpoint is highlighted pink. The pink line will move through your source as you code as you step through its execution.

Debugging can also be initiated by choosing the **Trace Into** command, which causes the Debugger to stop on the first line after the main method.

2 Once execution has halted (whether on a breakpoint or just after the main method), use the **Trace Into**, **Go To Cursor**, **Trace Over**, **Trace Out**, and/or **Continue** menu or toolbar items under the Main Window **Debug** menu (or the keyboard shortcuts F7, F8, CTRL+F7, and F5, respectively) to proceed.

Trace Into steps into the method at which the Debugger is currently stopped if there is a method call on that line and breaks at the start of the called method, enabling you to observe execution incrementally. If there is no method call on the current line, then it behaves like **Trace Over**.

Go To Cursor executes the current statement and all ensuing statements until it reaches the line that the insertion point is on.

Trace Over executes the current statement without breaking and stops at the next statement.

Trace Out halts execution after the current method finishes and control passes to the caller.

Continue resumes execution, which continues until it reaches the next breakpoint or the end of the application.

Finish Debugging ends the current debugging session.

By stepping through your code like this, you can monitor whatever parts of the system you choose during execution of the code.

Suspending and resuming debugging

The **Debug** menu and toolbar also have **Suspend All** and **Resume All** options, which enable you to "pause" execution at any time and then continue from the point execution was suspended.

To suspend selected threads or thread groups:

1. Under the **Threads** tab in the Debugger Window, select the nodes of those threads or thread groups (using SHIFT to select multiple consecutive nodes and CTRL to select various non-consecutive nodes).
2. Right-click on one of the selected nodes and choose **Suspend** from the contextual menu.

To suspend all threads, either:

- ◊ Choose **Suspend All** from the **Debug** menu or toolbar.
- ◊ Right-click on the root **Threads** node in the Debugger Window and choose **Suspend** from the contextual menu.

When a thread is suspended, the Debugger Window displays all current information for the thread.

Likewise, you can resume any or all of the suspended threads. Choose **Resume All** from the **Debug** menu to resume execution of all threads. To resume execution for threads individually, right-click on the thread or thread group and choose **Resume** from the contextual menu.

Changing the current thread

The current thread is set automatically when a breakpoint is reached. When you use the Trace In, Trace Out, and Trace Over commands, they affect only this thread. You can change the current thread manually.

To change the current thread:

- ◊ Under the **Threads** tab in the Debugger Window, right-click on the node of the thread you would like to switch to and choose **Switch to thread** from the contextual menu.

Connecting the Debugger to a running process

To connect the Debugger to an already-running virtual machine:

1. When launching the process, enter `-Xdebug` in the Java virtual machine's parameter list (after `-classic` when running on HotSpot) and note the agent password.
2. Choose **Connect** from the **Debug** menu or toolbar to bring up the Connect to Running VM dialog box.
3. Enter the host name and agent password in the dialog box.

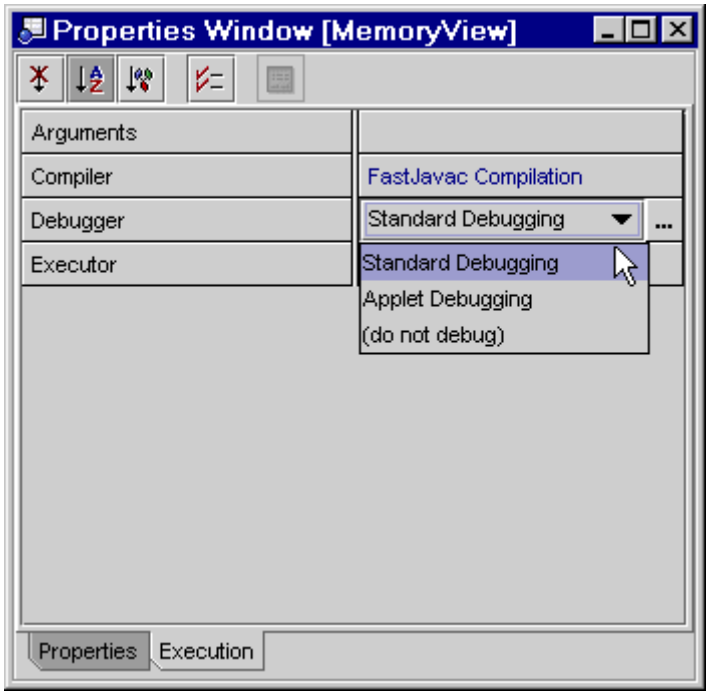
After clicking **OK**, the Debugger will connect to the running VM, and you will be able to see all threads as if you were debugging locally. If you have source code for the debugged application and you set a breakpoint in the source code, the Editor will be opened with the breakpoint line highlighted in the source.

Setting the debugger

The debugging category (for example, applet or standard) is set for each separate object in the IDE. When you set debugging, you choose from a list of "debugger types", each of which represents a specific configuration of a debugger (including the path to Java and arguments). There can be multiple debugging types for a given debugging category, though the IDE comes with only one for each category. See "Adding and modifying service types" on page 204 for more information.

To switch an object's debugging type:

1. Select the object under the **Filesystems** tab in the Explorer.
2. Go to the object's property sheet (by choosing **View | Properties** from the main menu).
3. Click on the **Execution** tab in the Properties window.
4. Rotate through the debugging types by double-clicking on `Debugger`, or click on the `Debugger` property's value and choose a type from the pull-down menu.

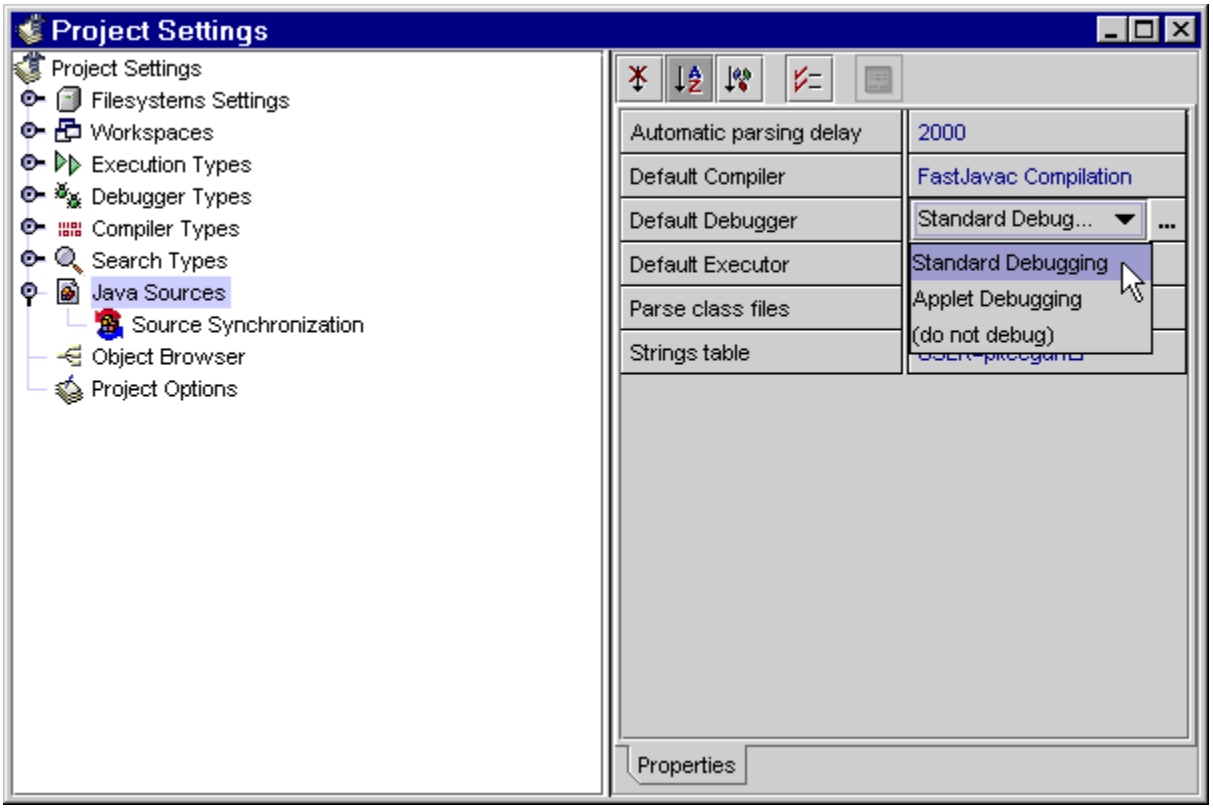


Switching the default debugger type

It is possible to switch the default debugger type. When you switch the default debugger type, this switch affects all classes and templates for which the user has not specifically assigned a debugger type.

To switch the default debugger type:

1. Choose **Project | Settings...** from the main menu to open the Project Settings window.
2. Select the `Java Sources` node in the left pane of the Project Settings window.
3. Select the `Default debugger type` property and choose a new default debugger type from the dropdown list.



Note: Once you change a class or template's debugger type, the IDE will never again recognize the class as using the default debugger type, even if you switch the debugger type for the class back to the one that is the IDE default. Therefore, if you change a class's debugger type and then change it back to the default debugger type, the class's debugger type will not be affected if you change the default debugger type.

Configuring debuggers

It is also possible to customize the command-line template for the debugger, thus affecting

the way the debugger is called. For more information, see "Adding and modifying service types" on page 204.

Setting debugger types in templates

If you use more than one debugger type in your work, you can create a set of templates with the different debugger types that you use. See "Creating your own templates" on page 201 and "Modifying existing templates" on page 202 for more information.

Using the Output Window

The Output Window is a multi-tab window displaying output from any component that produces output – such as the compiler or executed application or applet. Output from each component is presented on a separate tab: the **Compiler** tab, **Debugger** tab, and a tab for each executed process labeled with the name of the application being run.

By default, the Output Window is visible on the Running Workspace and is automatically displayed when you compile (Editing Workspace), execute (Running Workspace), or debug (Debugging Workspace) an application.

The **Compiler** tab is visible after compiling a Java source and displays compilation output and standard error. The output is color coded: errors are marked red, other text is blue. Double-clicking an error line on this tab brings forward the Editor window displaying the source, highlights the incorrect line in red, and positions the insertion point at the exact location of the compilation error. You can also jump to different compiler errors in the editor with keyboard shortcuts. Pressing Alt+F7 moves the insertion point to the previous error, and pressing Alt+F8 moves the insertion point to the next error in the file.

The Debugger tab is split vertically into two panes. The left pane displays the output of the application being debugged. The right pane displays useful debugging information such as details of the application's threads, thread groups and breakpoints as well as status messages from the debugger itself.

Any application currently being executed also has a separate tab in the Output Window. Its tab displays the standard output of the application. The standard input of the application (assuming the application tries to read anything) is also redirected here – more specifically, to the text field at the bottom of the window.

There are two properties which govern the use and re-use of application tabs: `Reuse Output Window Tab` and `Clear Output Window Tab`. These properties can be found on the property sheet of `Execution Settings` in the Global Options window (which can be opened by choosing **Tools | Global Options...** from the main menu). If the `Reuse Output`

`Window Tab` property is set to `True`, each individual application uses only a single output tab – that is, successive executions do not create new tabs. If the `Clear Output Window Tab` property is set to `True`, the tab is cleared before reuse. `Clear Output Window Tab` is useful only if `Reuse Output Window Tab` is set to `True`.

Contextual menus which provide window management options are available on the tabs – see "Window management" on page 187. You can also right-click in the body of the pane for the options **Copy to Clipboard** and **Clear Output**.

An application's output tab also provides a right-click contextual menu, with the options **Terminate Process** and **Close Output Window Tab**.

Internal web browser

Forte for Java, Community Edition includes the ICE Browser from ICEsoft, a built-in full-featured web browser that is useful in both testing and providing easy access to online help sources. It enables standard browsing capabilities from within the IDE and is useful in reaching online help sources.

To open the web browser, do one of the following:

- Choose **Web Browser** under the **View** menu on the Main Window.
- Press ALT+7, the default shortcut.
- Select a bookmark from **Help | Bookmarks** in the main menu.
- Open an HTML file or bookmark in the Explorer.

Once the browser is open, it operates like any simple browser. To load a different page, type the URL in the **Location**. The forward and back arrows cycle through previously-seen pages, the **Stop** icon stops the loading process, and the **Home** setting is set (by default) to the NetBeans web site, http://www.netbeans.com/. Clicking the **History** icon gives you a list of all URLs you have loaded in the session and enables you to double-click on any of them to bring that web page back. You can search for text strings in the web browser by choosing **Edit | Find** from the main menu or by using the CTRL+f keyboard shortcut.

To open an additional web browser window:

- Choose **Window | Clone View** from the main menu to open a new browser window.

  Another web browser window will open with the same web page. Type a new URL in the **Location** to load a different web page.

Note: You cannot open multiple web browser windows by other means. Choosing **Web Browser** or pressing ALT+7 causes the current window to revert to the default home page.

You can also set a different home page (the page that displays when you open the web browser or press the **Home** icon).

To set a different home page:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, select the `System Settings` node and enter the new home page under the `Home Page` property.

Chapter 4

The Explorer and Object Browser

Forte for Java provides two different tools for managing files in the IDE:

- The **Explorer** – which provides a hierarchical view of all files you use with the IDE, including applications you are developing and Javadoc files, and a view of running and debugging processes. In the Explorer, you can also **mount** directories that are on your local system or network so that they can be accessed from within the IDE.

- The **Object Browser** – which provides a Java-oriented view of your classes.

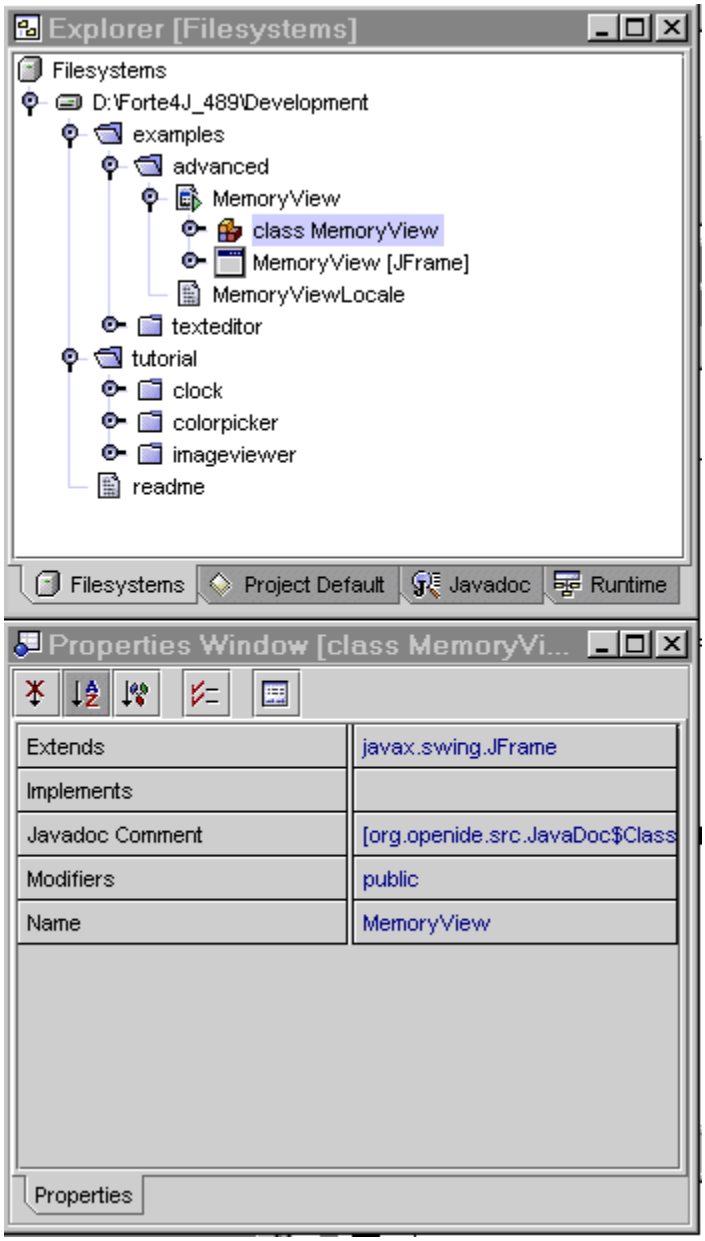
From these two windows, you can have code automatically generated for new classes and for specific elements of these classes. In addition, you can "synchronize" sources by having the methods for interfaces generated when you implement the interfaces in your code.

Through the Explorer, you can easily generate JavaBeans components. See "Developing and Customizing JavaBeans Components" on page 98.

Guide to the Explorer

The Explorer is not only a place where you can manage files, but also a place where you can create objects and manage their properties using:

- The contextual menu (available for each node by right-clicking on the node).
- The Properties Window, which is open by default in the Editing workspace (and otherwise accessible by choosing **View | Properties** from the main menu or **Properties** from the contextual menu of each item).



There are four tabs in the Explorer:

- **Filesystems** – contains work objects.
- **Projects** – shows the directories and files from Filesystems that you have added to the current project.
- **Runtime** – contains a list of currently running processes and debugger information.
- **Javadoc** – contains directories of API documentation in Javadoc format.

Filesystems

The **Filesystems** tab is the most important segment of the IDE's Explorer. It holds all the files that the IDE uses and all the files you create. Filesystems gives a unified view of files of all different types. You can access objects from different sources by mounting different file systems and JAR and ZIP archives in Filesystems.

File systems and the class path

A file system represents a hierarchy of files and folders. File systems can represent plain local files (or the network drive, depending on the operating system) or ZIP or JAR archives. Once mounted, the file system transparently hides any differences between these sources.

Some default items are added to Filesystems when the IDE is launched. However, there are additional file systems used by the IDE that are mounted at startup and hidden. All mounted file systems, including hidden ones, are visible in the Project Settings window under the `Filesystems Settings` node. Using the contextual menus and their property sheets, you can hide, unhide, or reorder them, and control their behavior.

Note: Once a new file system has been mounted, it is equivalent to having added the archive or directory to the class path accessible by the IDE. The mounted packages and classes are immediately available, and can be edited, compiled, and run.

Mounting file systems

To work with files in the IDE that you created outside of the IDE, you must **mount** them to the **Filesystems** in the Explorer.

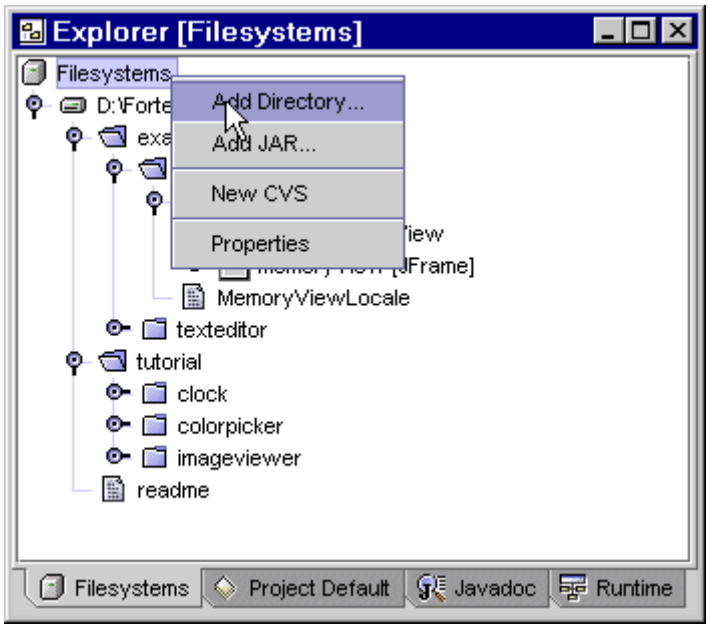
Mounted directories must be mounted at the default package, and Java sources in them must be in packages corresponding to their position relative to the mount point. If a file system is mounted at the wrong point, the IDE will be unable to compile its sources, and other problems will arise.

For example, mounting a file system from the point `C:\work` would not give the IDE correct access to the package `project1` stored as `c:\work\myprojects\project1`. In this case, this file system should be mounted at `C:\work\myprojects`, so that `project1` is the top-level package.

To mount a new file system:

◊ Right-click on the `Filesystems` node in the Explorer and choose **Add Directory...** from the contextual menu; or

◊ Choose **Tools | Add Directory...** from the main menu.



A dialog box will appear which enable you to choose the directory to mount.

To mount a new JAR or ZIP archive:

◊ Follow the same procedure as for adding directories, except choose **Add JAR** instead of **Add Directory**.

Note: Mounted JAR archives are read-only.

You can also mount a file system at the same time open one of the files in it.

To mount a file system and open one of its files:

◊ Choose **Open File** from the first toolbar or the **File** menu in the Main Window. A file chooser will appear which will enable you to browse your file system and select a file, which you can then add to the **Filesystems** tab of the Explorer. See "Adding a file to the IDE" on page 77.

To remove mounted directories or JAR or ZIP files from the Explorer:

◊ Right-click on the item and choose **Remove From Filesystems**.

Order of file systems

The order of file systems in the Filesystems is also significant. If files with the same name and hierarchy (for example, `com/MyPackage/MyClass.java`) exist in two different mounted directories, the first will be loaded during execution or debugging, even if you have selected the second one.

To change the order of file systems:

◊ Choose **Project | Settings...** from the main menu and, in the Project Settings window, proceed in one of the following ways

a. Right-click on the node of the file system under the `Filesystems Settings` node and choose **Move Up** or **Move Down** from the contextual menu.

b. Right-click on the `Filesystems Settings` node and choose **Change Order** from the contextual menu.

Adding a file to the IDE

The Forte for Java Open File feature provides the ability to open existing (meaning not created in the IDE) source files in the IDE straight from a file chooser without the need to first mount directories under the **Filesystems** tab in the Explorer or navigate to them in the Explorer.

To open a file in the IDE:

1 Choose **Open File** from the first toolbar or the **File** menu in the Main Window. A file chooser will appear asking you which file you wish to open.

2 Browse to the directory on disk where your file is (for example, `MyFirstClass.java`), select it, and choose **Open**.

If the file is already accessible in the Explorer, then the file will open (usually in the Editor window) immediately.

3 If the file is not already accessible in the Explorer, you must decide which package to put the file in. If you are trying to open a Java source file, a dialog box will appear suggesting the package to use and enable you to press **Accept** or **Select**.

a. If you press **Accept**, the file will be opened and the proper directory will be mounted.

b. If you press **Select**, a dialog box will appear with a list of possible packages. Select one and press **Mount**. The package you select will determine which directory will be mounted in the Explorer. The file will then be opened and the proper directory will be mounted.

If you are opening a Java source file, the IDE tries to determine the correct directory to mount by looking through the source file and trying to find a package declaration. For example, if you are opening `C:\mysources\com\mycom\MyClass.java`, and this file

begins with the declaration `package com.mycom;`, then `C:\mysources` will be selected as the default. If there is no package declaration, the directory directly containing the source file will be the default to mount.

You can override the default, but be sure that you choose the right mount point. If you do not choose the correct mount point, you will not be able to work with the file (your package declaration will be invalid, you will not be able to compile or execute the source, debugging will not work, and so on). When opening other types of files (such as GIF images, HTML, or resource property files) that do not have package declarations, you must be sure to mount the correct directory – look at the bottom of the dialog box to see what "package" will be used for the file. The choice will affect any Java sources that are in the same mounted directory. If you are using classloader-based resource loading (that is, based on an abstract resource name such as `/com/mycom/myImage.gif`), such as when creating a JAR file for distribution, then the resource names and JAR manifest entry names will be relative to the mounted directory as well.

Files that cannot be opened for editing will typically just be displayed in their own Explorer window. You can also select ZIP or JAR archives with Open File. When you do, they are immediately mounted in the Explorer and an Explorer window is opened on their contents to make it easy to browse archives.

Note: Mounted JAR archives are read-only.

Working with packages

All objects you create when writing an application in Forte for Java are displayed under the **Filesystems** tab in the Explorer, where they can be added, removed, and edited. You can organize these objects in **packages**, which are visually represented by folder icons.

When starting a new development project, identify the data path or file system (see above) to use for your work, right-click on that path, and choose **New Package** from the contextual menu. Once you enter a name for the new package, it appears in the Explorer under the path you have selected. You can also create a package at the same time as you create a new class from template in the New From Template wizard.

You can create several layers of packages (meaning packages within packages) in the same way. You can delete a package by right-clicking on its node and choosing **Delete** from the contextual menu or by selecting it and pressing the DELETE key on your keyboard.

Note: When you cut or copy source files and paste them to a different package, the sources' package declarations are automatically updated to reflect the new package. If you copy and paste a file to the same package, the pasted copy is automatically given a unique name (which you can change, either from the contextual menu, or by clicking on it in the Explorer to select it and then clicking again for an in-place rename).

The contextual menu commands available on nodes in the Explorer enable a wide range of

operations – from creating new packages to changing properties. The following is a list of menu items that appear in various contextual menus.

Table 1: Package contextual menu commands

Command	Description
Explore from Here	Opens a new Explorer window with the selected package as the root.
Find...	Enables you to conduct a search of files in directories mounted in the Explorer.
Refresh Folder	Updates the view, reflecting any changes to files in the folder made outside the IDE.
Compile	Compiles all uncompiled or modified objects in the selected package, at the selected level in the hierarchy.
Compile All	Compiles all uncompiled or modified objects in the selected package and recursively in all sub-packages.
Build	Compiles or re-compiles all objects (whether already compiled or not) in the selected package at the selected level in the hierarchy.
Build All	Builds all objects in the selected package and recursively in all subpackages.
Cut / **Copy** / **Delete** / **Rename**	Standard clipboard-style operations.
Paste | Copy	Pastes a copy of the object most recently copied under the selected node.
Paste | Create Link	Creates a link under the selected node to the most recently copied object. The object remains stored in the location where it was copied, but it can also be opened from the node where the link is pasted.
Paste | Instantiate	Creates a new instance of the copied template (only available when a template is on the clipboard).
Paste | Serialize	Serializes the instance of the copied JavaBeans object and places it in the selected package.
Paste | Default instance	Places the default instance of the copied JavaBeans object in the selected package, meaning that the name of the class is stored and the class name is provided as the default constructor in the pasted copy.

Command	Description
New Package	Creates a new, empty package as a sub-folder of the selected package.
New from Template	Creates a new object in this package, using one of the pre-built templates available under the **Templates** node in the Explorer.
Tools | Update Parser Database	Updates the Java parser database with the classes of the selected package, thus making those classes available in addition to the standard Java 2 Platform SDK classes when using the Java code completion feature in the Editor.
Tools | Generate Javadoc	Generates Javadoc documentation for the selected classes or elements and places it in the directory that you specify.
Properties	Opens a separate Properties window showing properties of the selected object(s).

Working with objects

While working in Forte for Java, Community Edition, you operate with objects rather than plain files. Each object represents one or more files on disk. Each object is shown with its own icon and properties. The following table shows the object types used in the IDE.

Table 2: Object Types

Icon	Object Type
	**Package** – A package (folder)—on disk or in a JAR or ZIP archive.
	**Java object** – Represents one Java source file (`.java`). Its children represent methods, variables, constructors, and inner classes acquired by parsing the Java source.
	**Form object** – Represents one Java source file that can be edited visually in the Form Editor in Forte for Java. The two types of subnodes are: 1) classes with methods, variables, constructors, and inner classes acquired from parsing Java source; and 2) items representing components on the form (visual hierarchy).
	**Class object** – Represents one Java class without source code. Children are methods, variables, constructors, and inner classes acquired from Java reflection.

Icon	Object Type
	**Serialized prototypes** – Files with a `.ser` extension, which are serialized objects.
	**HTML object** – Represents an HTML file.
	**Text object** – Represents a text file.
	**Image object** – Represents GIF or JPEG images. You can view these with the built-in image viewer.

There are several ways of creating new objects including choosing the **New From Template** command in:

- The Main Window toolbar
- The **File** menu
- An Explorer package's contextual menu

See "Creating new objects from templates" on page 200 for more information.

You can also copy objects from one package and paste them in another using **Copy** and **Paste** from the contextual menus. There are special paste options for JavaBeans objects.

Objects can be removed by pressing DELETE on the keyboard.

Three advanced operations can be done with JavaBeans and serialized prototypes. You can:

- Customize them (using the **Customize Bean** command) and make serialized prototypes from the customized object.
- Copy and paste them into the Component Palette.
- Copy and paste them directly into the Component Inspector (without the need to install them in the Component Palette first).

The following table shows some of the commands available in the contextual menus for the various types of objects.

Table 3: Common Object Commands

Command	Description
Open	Opens the default viewer for the object type – usually the Editor window. Also opens up the Form Editor window and Component Inspector for visual classes.
Customize Bean	Displays a window with the property sheet for the selected class as well as the option to serialize it.
View	Opens an HTML object in the default applet viewer.
Compile	Compiles selected object(s).
Execute	Runs the selected object.
Cut / **Copy** / **Paste** / **Delete** / **Rename**	Standard clipboard-style operations.
New | Method (or Constructor, Initializer, Variable, Inner Class, Inner Interface)	Creates a new element (of the type chosen in the submenu) in the selected class or source file. These commands are available on the contextual menu for the relevant category node (for example, `Fields`)
New | Property	Brings up the New Property Pattern dialog for creating a new JavaBeans property for the selected bean.
Tools | Create Group...	Creates a group (object composed of links to one or more files, enabling you to access them from the same place in the IDE). See "Group of Files" on page 201.
Tools | Auto Comment...	Enables you to comment your source code automatically and view all parts of your source code (methods, constructors, inner classes, variables, and such) and document them individually.
Tools | Generate JavaDoc	Generates JavaDoc documentation for the selected classes or elements and places it in the directory that you specify.

Command	Description
Tools | Add to Component Palette	Adds selected object to the Component Palette.
Tools | Synchronize	Forces synchronization of the selected source file with the interfaces it implements.
Tools | Set As Project Main Class	Sets the selected project as the class to be executed when you choose the Execute Project command.
Tools | Add to Project	Adds the selected class to the project, making it subject to compilation when you choose the Compile Project command.
Save as Template	Publishes the selected object as a template for future use.
Properties	Opens a separate Properties window showing properties of the selected object(s).

Searching for files in Filesystems

It is possible to do searches to find files in the mounted directories, based on modification date, file name, text in the file, or any combination of these three. You can also use regular expressions.

To initiate a search for files in Filesystems:

1. Choose **Tools | Search in Filesystems...** from the main menu, or right-click on a folder node in the Explorer or Object Browser and select **Find...** from the contextual menu.

   The Customize Criteria dialog box will appear.

2. Enter search criteria using any combination of the three tabs (**Date**, **Full Text**, and **Object Name**) and then press the **Search** button.

The Search Results window will then appear and give a tree view of the results of the search. You can use this window much as you use the Explorer. Contextual menus are available for each of the items listed, and you can use the Properties window to display the properties of the selected item. For every search that you initiate, a new tab will appear in the Search Results window with the results for that search.

If you press the **Customize Criteria...** button, the Customize Criteria dialog box will reappear with the criteria used in the last search. You can modify these criteria to refine your search.

Organization of the Customize Criteria dialog box

The Customize Criteria dialog box box is divided into two panels:

- In the first panel, labeled **Predefined** , you can choose from and save sets of search criteria.
- In the second panel, labeled **Modified** (for the **Date** tab), **Text Contains** (for the **Full Text** tab), and **Name Contains** (for the **Object Name** tab), you can adjust predefined criteria and create new search criteria.

The Predefined panel has a combo box that lists the options **Apply** and **Do not apply**, the empty search criteria for that type, and any sets of search criteria you have saved.

When **Do not apply** is selected, the criteria entered in the second panel are not applied in the search. When **Apply** is selected, the criteria in the second panel will be used in the search. When you select one of the predefined search criteria, that criterion will be appear in the second panel.

Doing simples searches

To do a simple search of file systems:

1. Select one of the tabs in the Customize Criteria dialog box based on whether you want to search by date of the file, text in a file, or text in the file's name.
2. If you select the **Date** tab, type a number in the **During last .... days** field or enter a range of dates in the **Between** and **and** fields. Date formats are determined by your locale. For the American English (en_us) locale, dates are entered in the format `MMM DD, YYYY`. If you use an invalid date format, your entry will appear in red.

   If you select **Full Text** or **Object Name**, enter a substring (remember that the search engine is case sensitive) or POSIX-style regular expression.
3. Press **Search**.

Searching specific nodes in Filesystems

If you want to search only in a specific part of the hierarchy in Filesystems, you can do so by selecting a node and selecting **Edit | Find...** from the main menu or using the CTRL+f keyboard shortcut. The Customize Criteria dialog box will appear. After you enter the search criteria, the search will be conducted recursively on all subnodes of the one selected.

Saving search criteria

Once you have entered a set of search criteria in the Customize Criteria dialog box, you can click **Save as...** and then enter a name for the criterion in the dialog box to save it for use in later searches. Search criteria are saved by search category.

Using saved search criteria

To search with a criterion that you have already defined, select the criterion in the combo box in the panel labeled Predefined and then press **Search**. You can also recall saved criteria and modify them. By pressing **Restore**, the newly-entered criteria are changed back to their saved values.

Searching with combinations of search criteria

You can search for files using a combination of the three types of search criteria. To do so, enter a search criterion under one of the three tabs and then switch to another tab and enter criteria.

Note: The predefined search criteria are stored by category. For example, predefined criteria created under the **Date** tab can not hold any Object Name criteria.

Predefining and modifying search criteria in the Project Settings window

Search criteria can be created and modified under the `Search Types` node in the Project Settings window.

To predefine search criteria:

1. Open the Project Settings window by choosing **Project | Settings...** from the main menu.
2. Expand the `Search Types` node and select the category you want to add a predefined criterion to and then expand that node.
3. Right-click the category node and select **New *SearchCategory* Service** from the contextual menu.

   A new criterion node will appear under the category node.
4. To rename the new criterion, select its node and use in-place renaming, right-click and select **Rename** from the contextual menu, or edit the `Identifying Name` property on its property sheet.

Runtime

The second major grouping in the Explorer hierarchy is `Runtime`. Nodes under `Runtime` display runtime information for execution (executed processes) and debugging (breakpoints, thread groups, and watches), as well as any external services as provided by extension modules and their connection to the IDE.

Processes node

The `Processes` node lists all processes currently being executed within the IDE. The contextual menu for each of these items contains just one element: **Terminate Process**. This enables you to force termination of a process.

This view is mirrored by the Execution View.

Debugger node

During a debugging session, the Debugger shows individual watches, threads and breakpoints. By right-clicking on Breakpoints or Watches, you can also add a new breakpoint or watch. This information is also available in the Debugger Window (part of the default Debugging workspace).

There is a detailed description of the Debugger in "Debugging Java classes" on page 58.

Javadoc

The **Javadoc** tab in the Explorer contains directories of API documentation in Javadoc format. You can mount Javadoc directories and navigate the hierarchies just like you would other directories under the **Filesystems** tab. Each node has its own property sheet and contextual menu, which you can access just as you would for nodes under **Filesystems**.

For more information on mounting and browsing Javadoc directories in Forte for Java, see "Searching and browsing Javadoc" on page 168

Project

The fourth tab from the left in the Explorer has the name of the current project. In its hierarchy are listed any files or directories that you have added to the project. The files are added as **shadows** of files mounted under the **Filesystems** node, meaning that any changes you make to the files from the **Projects** node will be reflected in in **Filesystems**.

For more information on how to work with projects, see "What is a "project" in Forte for Java?" on page 153 and "Organizing work into projects" on page 153.

Guide to the Object Browser

The Object Browser gives you a Java-oriented perspective on your classes, enabling you to

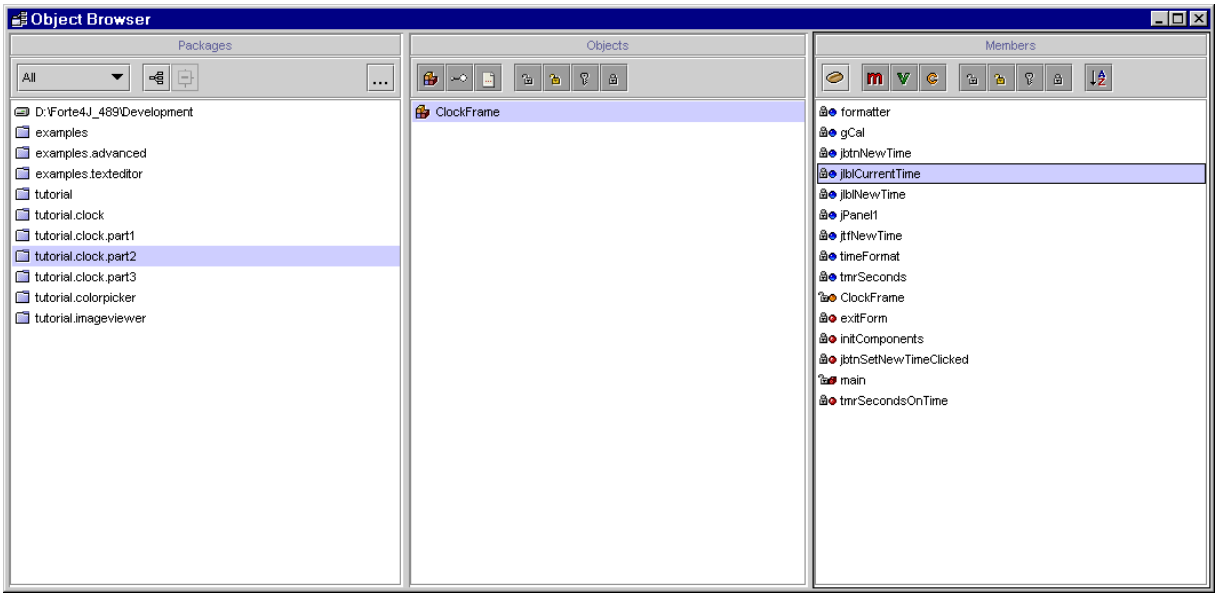
view a hierarchy of source files filtered in the way that you specify.

To open the Object Browser, do one of the following operations:

◊ Click on the **Browsing** workspace tab.

◊ Choose **Open Browser** from the **File** menu or toolbar.

The Object Browser window is divided into three panes – Packages, Objects, and Members.



Packages pane

The Packages pane lists all of the packages from all of the file systems visible under the **Filesystems** tab of the Explorer. The first icon on the top of the pane is the **Show as tree** icon, which gives you the choice whether to view the packages as a tree (when selected) or a list (when deselected). The second icon is for expanding the whole tree. A pull-down menu in the upper left corner enables you to choose different package filters.

Objects pane

The Objects pane displays objects of the package selected in the Package pane. Three types of objects (classes, interfaces, and sourceless files) can be filtered. The seven toggle icons in the top of the pane represent filters for these objects. The first three icons (**Class**, **Interface**, and **Sourceless Data Object** without source) represent general filters, meaning you can choose whether to display any objects of these types. The second group of icons (**Public**, **<default>**, **Protected**, and **Private**) are filters for the class or interface according to their access modifiers.

For example, if only the Class, Private, and Package filters are selected, all classes with private and package-private modifiers will be displayed.

No classes with public or protected access and no interfaces at all will be shown.

By default, all seven filters are selected, meaning that all objects are shown, including classes and interfaces with any access modifier.

Whereas the Explorer uses a tree structure, the Objects pane uses a list structure. Whereas the Explorer tree would should show class `Innerclass` as a subnode of `Outerclass`, the Object Browser would display them in a list as the separate items `Outerclass` and `Outerclass.Innerclass`.

Members pane

The Members pane displays members of the object selected in the Object pane.

The first icon in the Members pane is a toggle switch that enables you to view members as bean properties and events. When this icon is selected, the other filters in the pane are disabled.

The next three icons represent filters for general member types (methods, variables, and constructors). As with class and interface objects in the Objects pane, you can further filter method, variable, and constructor members according to their access modifiers using the second group of icons (identical to the last four icons in the Objects pane).

Clicking on the last icon in this pane enables you to sort the members alphabetically within member type.

Tip: You can use the tool tip feature to find out the names of the filters and more information about the items listed in the panes. To get an item's tool tip, place the mouse cursor over the item and wait about a second for the tool tip to appear.

Using the Object Browser

Much like with the Explorer, you can use the Object Browser as a base for many tasks in the development cycle. Contextual menus are available for every item listed in the Object Browser, enabling you to do the following operations:

- Open objects or their members in the Editor window.
- Open HTML files in the web browser.
- Cut, copy, and paste objects and their members.
- Delete, rename, and add packages, objects, and members.
- Compile or build an object or package.

- Open up a separate Explorer window on a package.

When you open a source file from the Object Browser, the Editor window by default opens up just below the Object Browser, enabling you to easily switch between these windows. If multiple source files are open at the same time, each file has its own tab on the bottom of the Editor window, enabling you to go back and forth between files.

Note: The Object Browser is not specifically designed for designing visual applications. When working on visual projects, you may find it easier to work in the GUI Editing workspace with the Explorer where you can easily edit properties for visual forms and double-click on a visual source node to open the Form Editor on that node. See "Designing Visual Classes with the Form Editor" on page 111 for more information.

Creating package filters

If you have a lot of packages in mounted in the IDE, you may want to create custom package filters in the Package pane of the Object Browser to define which of them you want to view.

To create a new package filter:

1. Make sure that the **Show as tree** icon is *not* selected.
2. Click the **...** button in the upper right corner of the Packages pane to bring up the Package Filter dialog box.
3. In the Package Filter dialog box, click **New**.
4. Now a Change Filter Name dialog box will appear. Enter a name for the new filter and click **OK**.

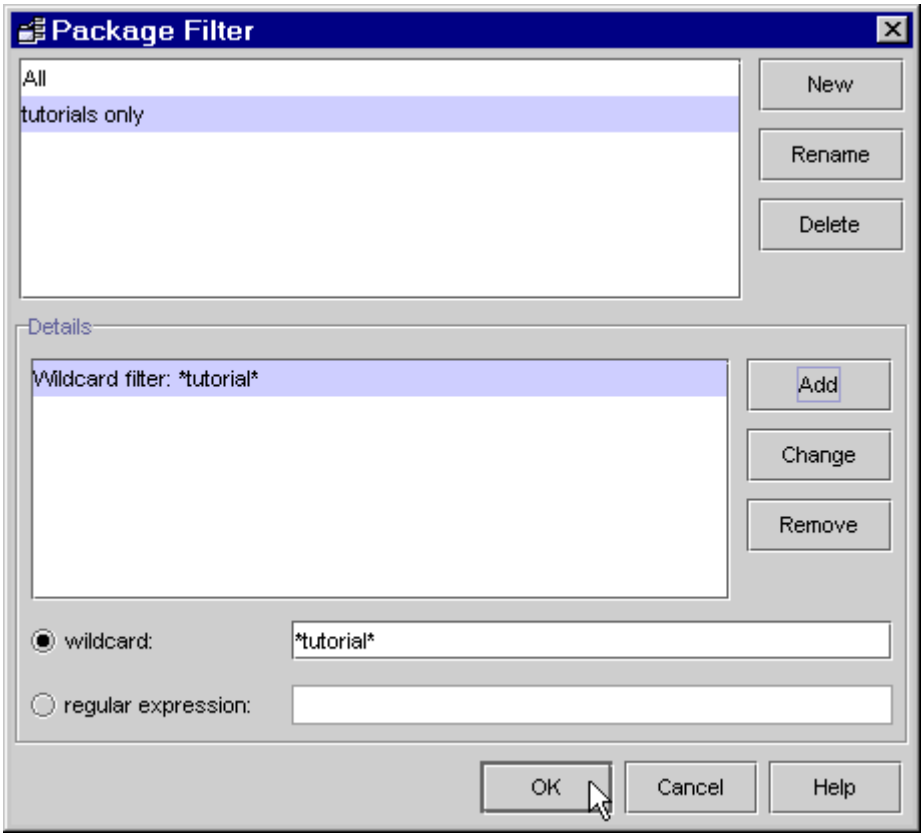
Next you will need to enter the specifications for your filter. In the details panel of the Package Filter dialog box is a list of filter expressions. By default, the expression `com.*` is created for all new filters.

There are two ways to create a filter:

- Using the **wildcard** option, you can enter prefix (for example, `com.netbeans*`), suffix (for example, `*debugger`), or infix (for example, `*debugger*`) expressions to narrow down the number of displayed packages.

- Using the **regular expression** option, you can create more complex filters using regular

expressions (in standard POSIX format).



To add an expression to a filter:

1. Select the filter and click **Add**.
2. Click the radio button for **wildcard** or **regular expression** and type the expression in that field.
3. Click **Change**.

To change an expression in a filter:

1. Select the filter in the list of filters in the Package Filter dialog box and select the expression to be changed in the Details list (for example, `com.*`).
2. Continue with steps 2 and 3 of the add filter procedure.

Other buttons

- Pressing **Remove** takes the selected filter expression out of the selected filter.
- Pressing **Delete** removes the entire filter from the pull-down menu.

Browsing and exploring objects and their elements

You can view members of classes in both the Object Browser and the Explorer. Source files in a file system are represented as Java objects. Each source file node contains at least one class node as well as code representing members such as methods, constructors, and variables. In the Object Browser, the top-level and inner classes are shown in the Objects pane with all of their members appearing in the Members pane. In the Explorer, classes, named inner classes, and members are all represented hierarchically in subtrees of the Java objects.

If you want to change the way the nodes for elements are labeled, you can choose **Tools | Global Options...** from the main menu, select the `Java Elements` node, and enter a format for each type of element. See "Java Elements reference" on page 272 for a guide to the substitution codes and formats available.

The tool tip for each object and element displays the full signature for each element, including access modifiers and field types.

It is also possible to create elements of classes using the contextual menus of classes in the Explorer and Object Browser.

Table 4: Source and form file icons

Icon	Description
	Java source file
	Java file without source code
	Runnable Java object
	Invalid Java source file (cannot be parsed)
	Form object
	Runnable form object
	Incorrect form object (cannot be parsed)

Invalid package declarations

Source files that have the wrong package named in their code are marked with `Invalid package declaration` in the tool tip. To rectify this, you can change the package name in the file or mount a different directory. As a shortcut, you can use the Open File feature to mount a new directory and open a file in that directory. See "Adding a file to the IDE" on page 77.

Elements of Java objects and member accessibility

Subtrees of each Java object represent the hierarchy of elements. Java objects typically contain at least one class that contains elements such as constructors, methods, variables and inner classes. You can set behavior properties for each of these elements.

Table 5: Java Object Elements

Icon	Elements	Properties
	Class, Inner class	Name, modifiers, extended class, implemented interfaces
	Interface, Inner interface	Name, modifiers, extended interfaces
	Constructor	Name, modifiers, arguments, exceptions
	Method (non-static)	Like constructor, plus return type
	Method (static)	Like constructor, plus return type
	Variable (non-static)	Name, modifiers, type, initial value
	Variable (static)	Name, modifiers, type, initial value
	Initializer (non-static)	
	Initializer (static)	

Member accessibility

The elements listed above can have several kinds of accessibility. The icons of the elements consist of icons in the previous table and their accessibility flag(s).

Table 6: Icons for accessibility

Icon	*Access*
	private
	package private (default)
	protected
	public

Creating and modifying elements using the customizer

You can create elements for classes from the class's contextual menu in the Object Browser or Explorer.

Creating elements

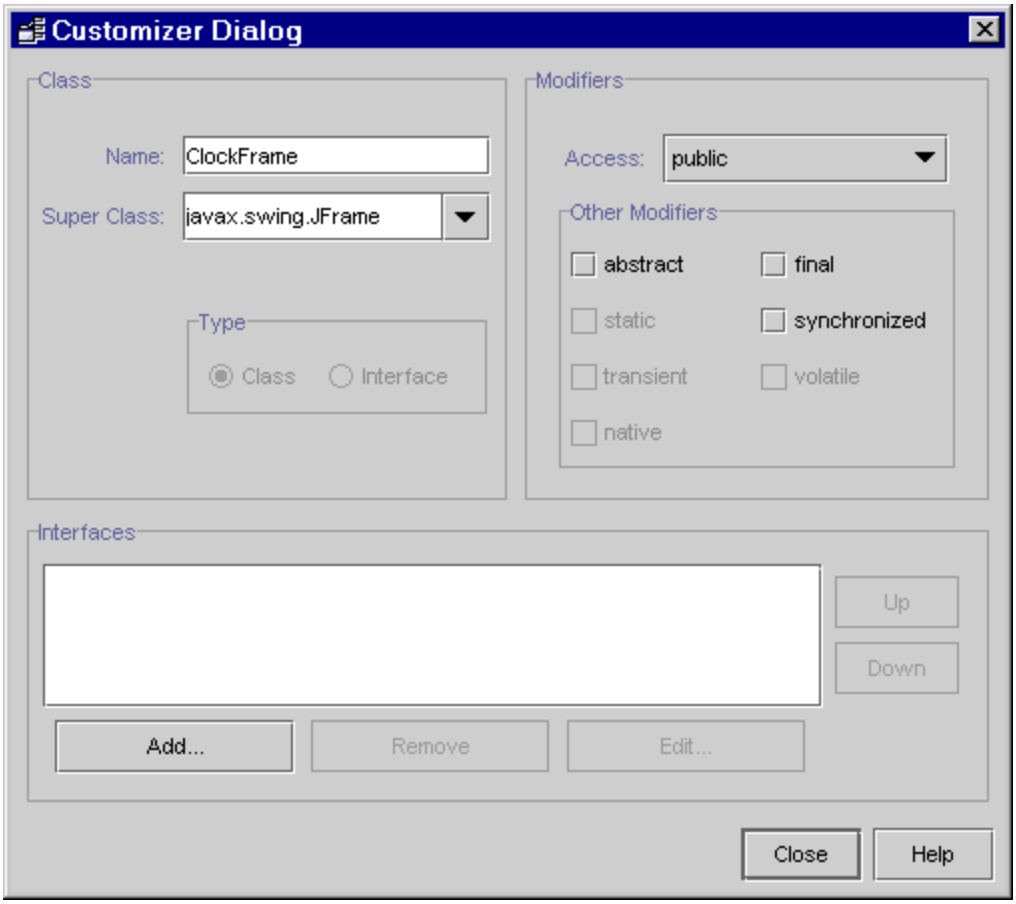
To create a new element:

1. In the Explorer or Object Browser, select the class (or interface) in which you want to create the element.
2. Choose **New** from the contextual menu.
3. In the expanded submenu, choose the element you want to create.

   If you select **New | Initializer**, a new initializer is immediately added to your code.

   If you select **New |** and then **Variable**, **Constructor**, **Method**, **Inner Class**, or **Inner Interface**, a customizer dialog box will appear. In the customizer, depending on the type of element you are creating, you can set member accessibility and other modifiers, method name and return type, class name and super class, method parameters and exceptions, and

interfaces.



Modifying classes and elements

You can modify classes and their members with the same type of customizer.

To customize an element:

1. In the Explorer or Object Browser, select the class or member that you would like to change.
2. Choose **Customize** from the contextual menu (or press the **Customize** icon, which is the last button on the toolbar in the Properties window).

   The Customizer Dialog will appear, enabling you, depending on the type of element you are creating, to set member accessibility and other modifiers, method name and return type, class name and super class, method parameters and exceptions, and interfaces.

Source synchronization

Source synchronization is a feature that helps to keep your Java source files current by:

- Automatically generating an interface's methods for you when you implement that interface in a source file; and
- Automatically updating all source files mounted in the Explorer that implement the interface when you change a method in or add a method to that interface.

Source synchronization works with all interfaces shown under the **Filesystems** tab in the Explorer and all standard interfaces in the Java 2 SDK. You can choose to synchronize source automatically just after parsing (which, by default, occurs after a two-second break in typing and insertion point movement) with or without confirmation, or explicitly by choosing **Tools | Synchronize** from the contextual menu of the implementing class in the Explorer.

All source files have a `Synchronization Mode` property on their property sheet (accessed by choosing **View | Properties** from the main menu). The property can be set to:

- **Do not synchronize**.
- **Confirm changes** – the default (and recommended) setting. The Confirm Changes dialog box appears enabling you to specify which methods are to be synchronized.
- **Without confirmation** – all interface methods are automatically generated for the implementing classes without a dialog box appearing to prompt you to confirm the changes.

Source Synchronization property sheet

You can adjust general source synchronization settings on the property sheet for the `Java Sources / Source Synchronization` node in the Project Settings window (**Project | Settings...** from the main menu). The properties are:

- **Return Generation Mode** – Generates either nothing, an exception, or the string "return null" when creating a new method declared to return a value.
- **Synchronization Enabled** – If `False`, all synchronization is turned off.

Synchronizing source

To synchronize (with confirmation) when implementing a new interface in a class:

1 In the Editor window, type `implements` *`InterfaceName`* in the class declaration line

of the source file.

2. Wait a few seconds for the source to be parsed, or save or compile the file. The Confirm Changes dialog box will then appear listing all of the methods to be added to the class. The dialog box gives you the following options.
   a. Choose **Process All** to add all of the interface's methods to the class.
   b. Select any combination of the methods in the **Changes List** and choose **Process** to add only those methods to the class.
   c. Choose **Close** to prevent any of the methods from being added to the class.

You can change the synchronization mode for the destination class by selecting one of the radio buttons at the bottom of the dialog box.

To synchronize (with confirmation) when changing or adding methods to an interface:

1. Add the method(s) to the interface code and then wait for parsing to occur and the Confirm Changes dialog box to appear. The dialog box will display the updated methods for the first class found that implements the interface.
2. Follow step 2 from the previous procedure.

If you have multiple source files (in directories mounted in Filesystems) that implement the changed interface, a Confirm Changes dialog box will appear for each one of them.

If you have automatic synchronization disabled on a specific source file or for all source files, you can still synchronize a class with its interface's methods by choosing **Synchronize** from the class's contextual menu.

Note: The **Synchronize** command only works when chosen on class objects (not interfaces).

Chapter 5

Developing and Customizing JavaBeans Components

Forte for Java, Community Edition provides tools to make generation and customization of JavaBeans components faster and easier. You can create the standard parts of your bean – such as properties, listener support, default constructor, and BeanInfo – without having to write any code manually. You can also customize existing beans and add beans to the IDE for use in your development work.

Developing JavaBeans components

In Forte for Java, creation of JavaBeans components begins with any class that has source code. In the Explorer, you will find `Bean Patterns` as a subnode of any such class. All of the operations for maintenance of property and event sets – thus all of the things that determine the characteristics of the bean – can be accessed and modified from this subnode.

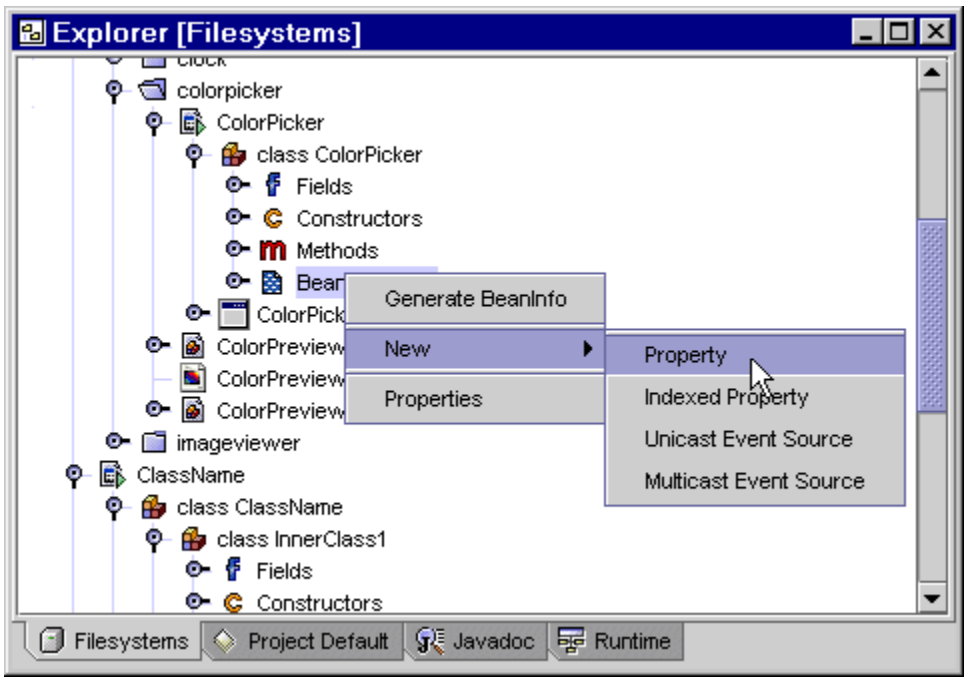
Note: `Bean Patterns` also includes non-public property and event mechanisms, which are not recognized as JavaBeans patterns in introspection. This is because `Bean Patterns` can be used to set property and event mechanisms in standard classes which are not true JavaBeans components.

For a more thorough review of JavaBeans components and concepts such as introspection, see http://java.sun.com/beans/.

Creating properties

To create a property in a class source:

1 Find the class in the Explorer, expand its node, and right-click on `Bean Patterns`.



2 In the contextual menu, choose **New | Property**. The New Property Pattern dialog box will appear, which will enable you to customize the code to be generated for the property.

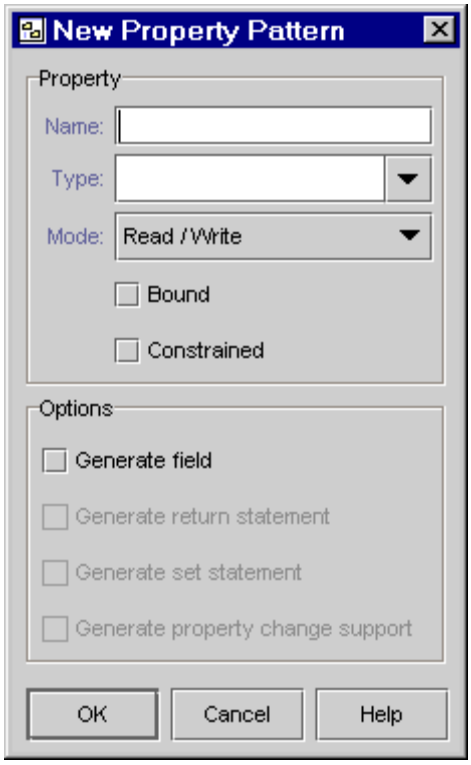
3 Type in a name for the property. (The name must be a valid Java identifier.)

4 In the **Type** combo box, choose the type of property from the list, or type in a class identifier.

5 In the **Mode** combo box, choose whether to have the getter method (`READ ONLY`), setter method (`WRITE ONLY`), or both methods (`READ/WRITE`) generated.

6 Check the **Bound** (meaning that property change events must be fired when the property changes) and/or **Constrained** (meaning that the property change can be vetoed) options, if applicable to the property. The usefulness of checking these options is enhanced if you

also check the **Generate Property Change Support** option – see below.



If you click **OK** after these steps, the definitions of the methods will be generated. However, you can also check all or any combination of the following options:

- If you check the **Generate field** option, a private field is generated. This field will have the same name and type as the property.
- If you check the **Generate return statement**, a statement that returns the field (for example, `return myProperty;`) is inserted in the body of the getter method.
- If you check the **Generate set statement** option, a statement setting the value of the property field to the value of the setter parameter will be inserted into the body of the setter method.
- If you check **Generate property change support**, all of the code needed for firing `PropertyChangeEvents` (for bound properties) and `VetoableChangeEvents` (for constrained properties) will be generated in the bodies of the setter methods. In addition, code to declare and initialize the property change support object is generated.

Creating indexed properties

You can also create indexed properties. Indexed properties are usually implemented as arrays or vectors and enable you to set and get values using an index. Indexed getter and setter

methods have a parameter which specifies the index of the element to be read and written.

To create an indexed property in a class source:

1. Find the class in the Explorer, expand its node, and right-click on `Bean Patterns`.
2. In the contextual menu, choose **New | Indexed Property**. The New Indexed Property Pattern dialog box will appear, which will enable you to customize the code to be generated for the property.
3. Proceed according to steps 3 through 6 above in the instructions for creating a simple (non-indexed) property.
4. Check the items in the Options section appropriate for the property. These four options are analogous to the options for a simple property.

   In addition, the indexed properties may have getter and setter methods which enable reading and writing all elements (the whole array). The checkboxes in the Non-Index Options panel enable you to add a getter with or without a `return` statement and a setter with or without a `set` statement.
5. Click **OK**.

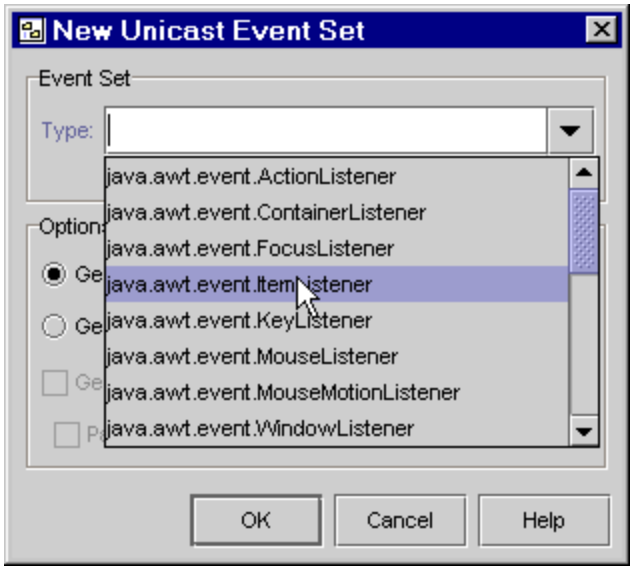
Creating event sets

You can also add event sets to your bean.

To add an event set deliverable to only one listener:

1. Right-click on the `Bean Patterns` subnode of your class and choose **New | Unicast Event Source** from the contextual menu. The New Unicast Event Set dialog box will appear on the screen.
2. Use the combo box in the **Type** field to select or type in any listener interface (event class

type) that extends `java.util.EventListener`.



3 Click one of the two radio buttons below the **Type** field to choose how you want the event set implemented. The choices are:

- **Generate empty** – generates an empty implementation
- **Generate implementation** – generates a simple implementation for one listener

4 Check the **Generate event firing methods** box if you want to generate a method corresponding to every method in the listener interface to fire the event to all listeners.

5 To specify how the event will be transferred to this method, check the option **Pass event as parameter**. This adds parameters to the firing events. The type of parameter (`EventObject` subclass) will be the same as the type of parameter of the corresponding method in the listener interface.

If you leave the **Pass event as parameter** option unchecked, the firing method will have the same parameters as the constructor of the event object class (subclass of `EventObject`), and this constructor will be called in the body of the firing method. If there are multiple constructors for the event class, the generator behaves as if the **Pass event as parameter** option is checked.

When you click **OK**, an `add`*`EventName`*`Listener` method and a `remove`*`EventName`*`Listener` method, along with firing methods if you specified them, will be added to your source.

For multicast event sources you can specify how to implement the adding of listeners and firing of events.

To add an event set deliverable to more than one listener:

1 Right-click on the `Bean Patterns` subnode of your class and choose **New | Multicast Event Source** from the contextual menu. The New Multicast Event Set dialog box will

appear on the screen.

2. In the **Type** field, specify any listener interface (event class type) that extends `java.util.EventListener`.
3. Click one of the three radio buttons below the **Type** field to choose how you want the event set implemented. The options are:

   - Generating an empty implementation
   - Generating a simple `ArrayList` implementation
   - Using the `EventListenerList` support class from the `javax.swing.event` package
4. Follow steps 4 and 5 in the procedure for adding unicast events.

Deleting properties and event sets

Properties and event sets can be easily deleted from classes using the Explorer or Object Browser.

To delete a property or event set using the Explorer:

1. Navigate to the property's node under the class's `Bean Patterns` node in the Explorer.
2. Right-click the property and choose **Delete** from the contextual menu.

To delete a property or event set using the Object Browser:

1. Open the Object Browser, select the package in the Packages pane and the object (Objects pane).
2. Select the **Bean** filter in the Members pane to display the members as bean properties and event sets.
3. In the Members pane, right-click the property or event set and choose **Delete** from the contextual menu.

Generating Bean Info

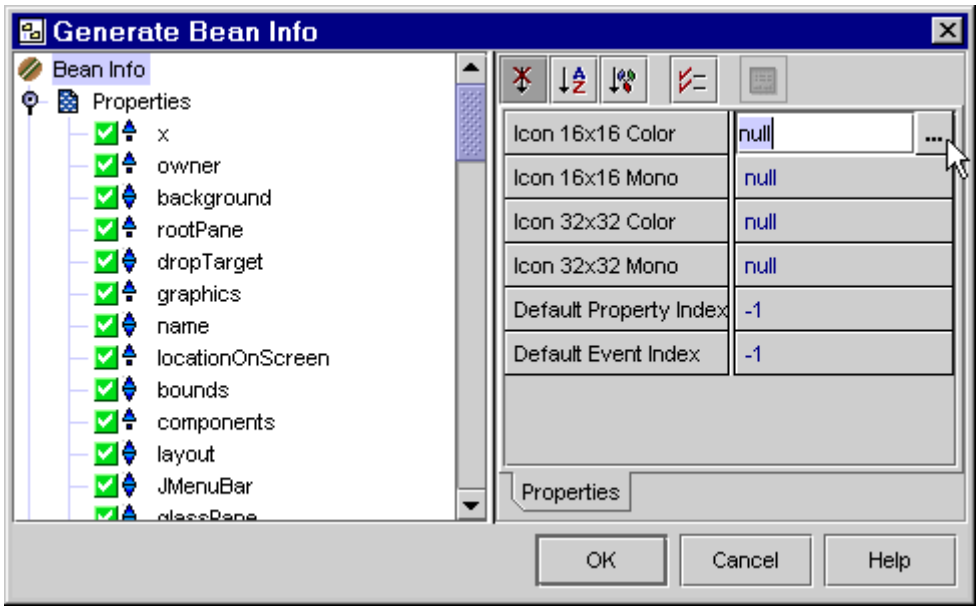
A BeanInfo class contains information about the JavaBeans component, such as properties, event sets, and methods. BeanInfo is an optional class, and you can choose what information will be included in it. In BeanInfo you can also specify additional properties of the JavaBeans properties and BeanInfo, including icons, display names, and so on.

By default, all superclass beaninfo, public properties and event sets are included in `BeanInfo` (permitting customization of various attributes), and all public methods are discovered by introspection.

To generate BeanInfo for a class:

1 Right-click on the `Bean Patterns` node in the class and choose **Generate BeanInfo** from the contextual menu.

The Generate Bean Info dialog box will appear. The left panel shows the three main nodes (`Bean Info`, `Properties`, and `Event Sources`) and the right panel shows the properties of the selected node.



2 Make any desired changes to the properties and then click **OK**.

Note: The JavaBeans component properties referred to in the `Properties` node should not be confused with the word "properties" as generically used in Forte for Java to refer to characteristics and settings (as displayed in each object's property sheet) of parts of the IDE and objects stored in it. Thus, in the Bean Info Generation dialog box, we refer to properties of JavaBeans properties.

Bean Info node

The first four properties (example, `Icon 16x16 Color`) enable you to designate icons for the bean by entering the relative path to and name of a graphic file (using a resource path).

The next two, `Default Property Index` and `Default Event Index`, apply to the whole BeanInfo class. In these two properties, you can specify a default property or event that may be declared in the BeanInfo to be customizable. The values are the indexes of the default property and default event set in the `PropertyDescriptor` and `EventSetDescriptor` arrays (respectively) and are set to -1 if there is no default property.

For more information on these and the other properties, see *BeanInfo* in the API documentation that comes with the Java 2 SDK or check Sun's web site,

http://java.sun.com/products/jdk/1.2/docs/api/java/beans/BeanInfo.html

Properties and Event Sources nodes

These nodes have only one property, `Get From Introspection`. By default, this property is set to `False`. If you set the value to `True`, then (when the bean is used) all information about the properties (or event sets) will be taken from introspection. Thus the generated `BeanInfo` will return a null value instead of an array of descriptors of properties (or event sets).

This feature is particularly useful if you want to customize either JavaBeans properties or event sources, but not both. Thus you can customize one (properties or event sources) and have the information about the other taken from introspection.

If you right-click on either of these nodes, you can choose from the contextual menu to either include or exclude all of the JavaBeans properties or event sets from the `BeanInfo`.

Subnodes of the Properties and Event Sources nodes

The subnodes are the JavaBeans properties and event sources themselves. Each of these subnodes has two groups of properties. The first group, under the **Properties** tab, are general properties and apply to both JavaBeans properties and event sets. The **Expert** tab holds settings specific to properties or event sets.

The most important property here is the `Include in BeanInfo` property, which is set to `True` by default. If you set this property to `False`, the JavaBeans property or event set will not be shown in the BeanInfo, either to users or to other classes. You can also change this value by right-clicking on the property or event source and choosing **Toggle Include**. When `True`, the icon has a green square with a white check mark; when `False`, it displays a red square with a white X.

In the **Expert** tab, you can change the mode of a JavaBeans property to make it more restrictive (that is, you can change a `READ/WRITE` to `READ ONLY` or `WRITE ONLY`).

Note: If you have any JavaBeans properties with non-standard names, source for these is not automatically generated in the `BeanInfo`. You can enter such code manually in the Editor.

In addition, if an indexed JavaBeans property has a non-indexed getter and setter, you can specify whether these methods are specified in the `BeanInfo`.

Tip: If you want to set a property sheet property to the same value for multiple JavaBeans properties or event sets, you can select several nodes at once and set the properties for all of them.

Editing BeanInfo source

After you have generated `BeanInfo` from the Generate Bean Info dialog box, you can manually edit the `BeanInfo` in the Editor window (except for the blue guarded blocks).

Regenerating BeanInfo

You can regenerate `BeanInfo` for classes that already have `BeanInfo`. If there is already a `BeanInfo` class for a bean, any changes you make in the Generate Bean Info dialog box or in the `BeanInfo` source code will be saved.

Customizing JavaBeans components

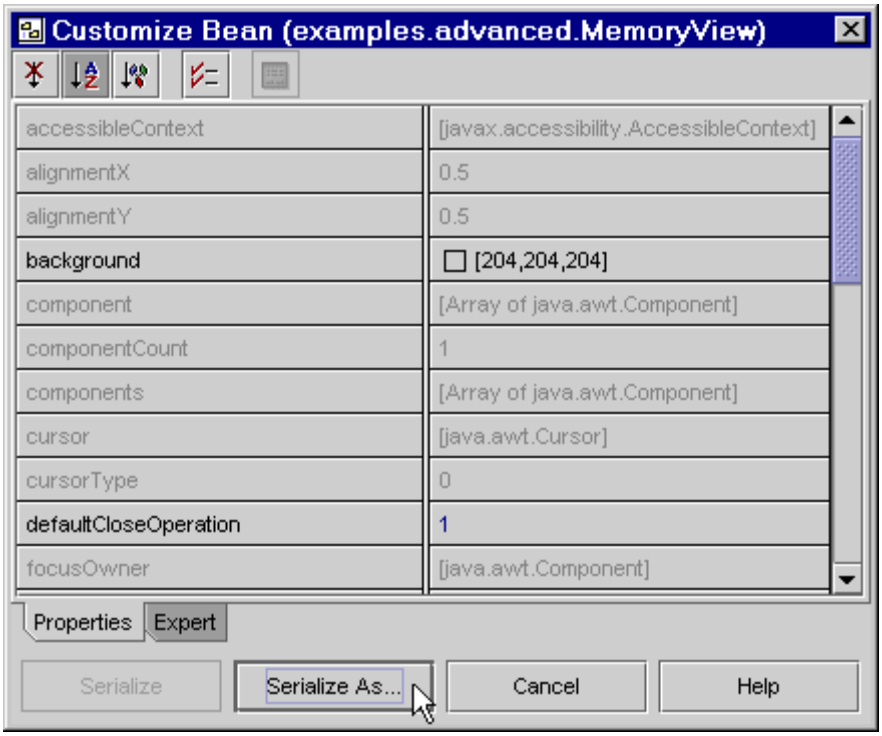
When writing in Java, it may be useful for you to take existing beans and alter them to better suit your purpose. The **Customize Bean** command enables you to create an instance of a JavaBean, customize its properties, and save it as a serialized prototype. This may be done on a Java class or on an already serialized prototype (`.ser` file). The class must be public and have a public default constructor, and it must also be serializable to make a serialized prototype from the customized settings.

To customize a JavaBeans component:

1. Right-click on the bean in the Explorer and choose **Customize Bean** from the contextual menu.

   The property sheet for the JavaBeans component will open up. If the bean is a visual component (meaning that it extends `java.awt.Component`), the component itself will

also appear on the screen.



2. Make the desired changes in the property sheet (or, if it is a visual component, on the component itself).
3. Click **Serialize** or **Serialize as** to create a serialized prototype (`.ser` file) of the bean with its customized properties.

   Serialize As enables you to select a name and location for the `.ser` file. **Serialize** (available for `.ser` files only) saves the bean into the same `.ser` file it originated from, overwriting the previous content.

   If you do not want to keep the customized settings, press the **Cancel** button to close the customization window.

Adding JavaBeans components to the IDE

You can expand the functionality of the IDE by adding your own visual and non-visual JavaBeans components to the Component Palette for use in visual design. Once the component is installed, you can easily select it in the Component Palette and add it to your forms. You can add class files (with the `.class` extension) or serialized prototypes (`.ser`).

To add a JavaBeans component to the Component Palette:

1. Choose **Tools | Install New JavaBean** from the main menu.

2. In the dialog box, select the path and JAR file of your new JavaBeans component and click **Open**.
3. A list will appear with available JavaBeans. Select the ones you want and click **Install**.

Note: If the beans you expect do not appear, the required attribute, `Java-Bean: True` is missing from the beans' entries in the JAR manifest. See the JavaBeans specification for details on proper packaging of JavaBeans.

4. In the Palette Category dialog box that appears next, choose where in the Component Palette (under which tab) you would like to place the bean(s) and click **OK**.

   You will then see the new bean(s) appear under the tab you indicated, ready to use in an application.

   Some JavaBeans components have their own icon. In cases where this is not true, the IDE assigns a default icon to the bean. To see the name of each installed JavaBean, position the cursor over the bean and a tool tip will appear.

Is Container property

A component's `Is Container` property enables you to choose whether or not to have the component treated as a container.

To access a component's Is Container property:

1. Choose **Tools | Global Options...** from the main menu to display the Global Options window.
2. Expand the `Component Palette` node and the component's palette category node.
3. Select the node for the component.

   The Is Container property for that component will appear in the property sheet pane.

If the `Is Container` property is set to `True`, you can use the component in your forms as a container. When you select such a component from the Component Palette, the Form Editor will assign the component a layout manager, and you will be able to add other components to it.

However, if the JavaBeans component you have created contains sub-components, these sub-components will not be recognized by the Form Editor. In addition, the container and layout manager code that the Form Editor generates could conflict with the original placement of your components.

To keep the Form Editor from creating container code which could conflict with your sub-components, you can set `Is Container` to `False`. Even though the Form Editor will not recognize the sub-components (they will not be listed in the Component Inspector), they

will appear in the Form Editor window and should work correctly in your program.

Adding JavaBeans from a JAR or local directory

You can also add beans from a JAR archive or from a local directory.

To add a bean from a JAR archive:

1 Either:

- Choose **Tools | Add JAR** from the main menu.
- Right-click the `Filesystems` node under the **Filesystems** tab in the Explorer and choose **Add JAR** from the contextual menu.
- Open the Project Settings window, right-click on the `Filesystems Settings` node and choose **New | Jar Archive**.

2 Select the path and directory, click on the `.jar` file, and click **Mount**.

The archive then appears in the Explorer, under the **Filesystems** tab, and in the Object Browser.

3 If you are using the Explorer, expand the node of the archive or directory that you have just added and locate the JavaBeans component you want to add.

4 Right-click on the JavaBeans component in the Explorer or Object Browser and choose **Tools | Add to Component Palette** from the contextual menu.

5 In the Palette Category dialog box, select the tab under which you would like the bean to appear in the Component Palette and click **OK**.

The new JavaBeans component on the Component Palette in the Main Window under the category you assigned.

To add a bean not packaged in a JAR or ZIP file:

◊ Follow the previous procedure, but use **Add Directory** instead of **Add JAR** in step 1.

To add a bean to the IDE from the Explorer:

◊ Follow steps 4 and 5 of the procedure for adding JavaBeans to the IDE.

Instead of using **Tools | Add to Component Palette**, you can copy and paste (using the **Edit** menu in the Main Window, contextual menus in the Explorer or Object Browser, or keyboard shortcuts) beans into the appropriate category under `Component Palette` in the Global Options window. If you copy and paste using the main menu or a contextual menu, you can choose from up to four options from the **Paste** submenu. For more on these options, see "Package contextual menu commands" on page 79.

Adding JavaBeans using the beans directory

You can have a JAR file installed automatically each time the IDE is started by putting it into the `beans` directory under the root of the installation. By default, each bean is put into a Palette category with the same name as the JAR file.

You can control which beans are loaded and what Component Palette categories they are loaded into by adding a `beans.properties` file.

Chapter 6

Designing Visual Classes with the Form Editor

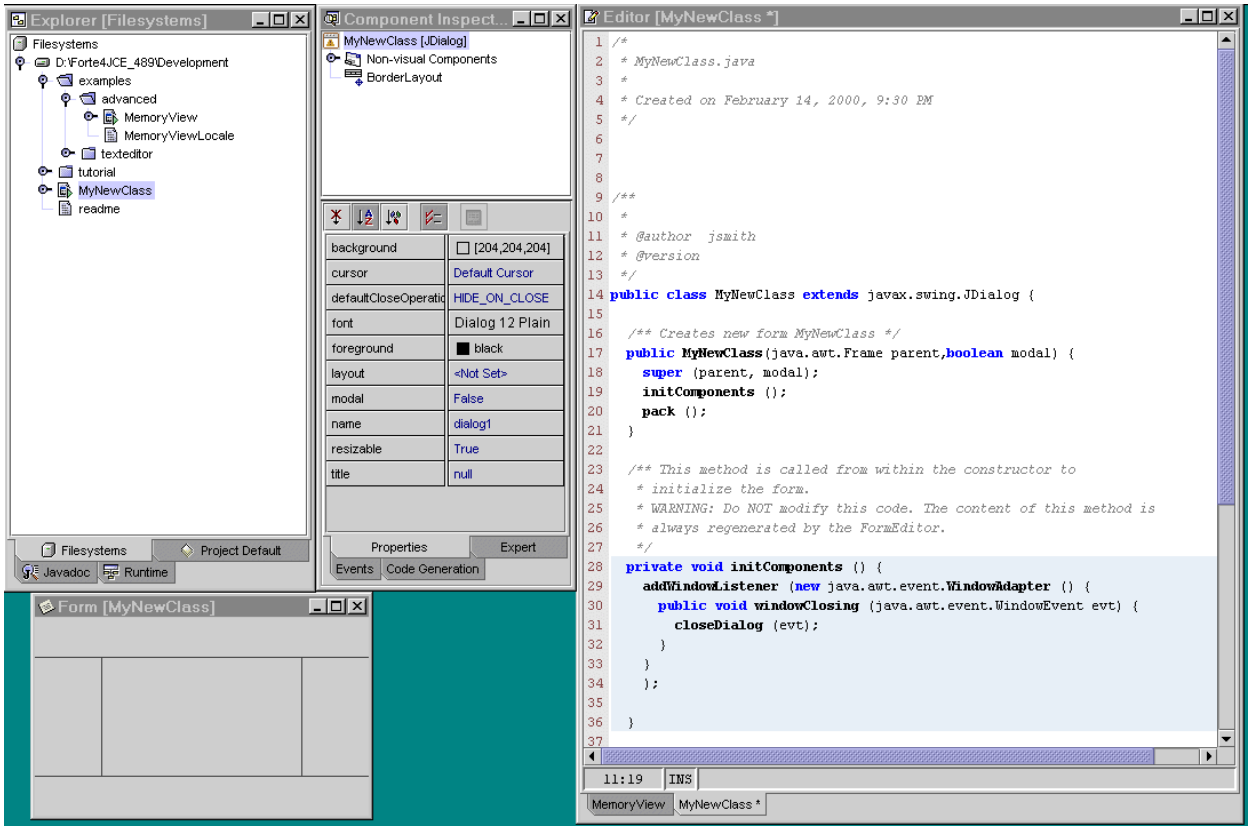
Forte for Java provides the **Form Editor**, which enables you to build forms for graphical user interfaces visually and have the code for them automatically generated. You can select items such as panels, scroll bars, menus, and buttons in the Component Palette and then place them directly on the Form Editor window. As you do, the IDE automatically generates the Java code to implement the design and behavior of the application. The Form Editor also uses your chosen layout manager to control the design-time layout of the form in the Form Editor window. If you change the layout manager or its parameters, the changes are displayed immediately. In addition, you can manage event handlers and customize the way code is generated for components.

Opening the Form Editor

To open the Form Editor for existing forms, double-click on the form object in the Explorer or choose **Open** from its contextual menu.

When a form is opened, the IDE (by default) switches to the GUI Editing workspace, and the following three windows are displayed:

- **Form Editor window** – the design-time view of the form.
- **Editor window** – contains the Java source for the form. If the Editor is already open, the form is opened on a new tab within that Editor window.
- **Component Inspector** – displays the hierarchy of components on the active form, including the layout managers, menus, buttons, and non-visual components such as timers. The current selection of components is highlighted. Tabs on the property sheet in the bottom panel display the general, expert, code generation, and layout properties as well as the events of the selected component. See "Property Sheet pane in the Component Inspector" on page 117.



If the Explorer is open, a form node (which is given the name of the class followed by the type of form in brackets) will appear in the source file's hierarchy.

If you have closed or minimized one of these windows and need to redisplay it, right-click on the form node in the Explorer or Component Inspector and select **Goto Form** (for the Form Editor window), **Goto Source** (Editor window), or **Goto Inspector** (Component Inspector).

Note: In Forte for Java, Community Edition, forms are stored in the XML format. When opening forms saved with NetBeans Developer 2.0, 2.1, X2 or early betas of 3.0, you will be

asked whether you want to convert the form to a new format. If you click **No**, the form editor will have the same features as in X2 and the saved form will be compatible with X2. However, some of the new features, such as multiple property editors per property and most code generation properties on components, will not be available. If you click **Yes**, the form will be saved in the new format, and it will not be possible to open it in these versions of NetBeans Developer.

Creating a new form

Use the **New From Template** command to create a new form from template and open it in the Form Editor.

Forte for Java supports nine basic form types from both the AWT and the newer JFC components. The table below lists them.

Table 7: Basic form types supported by Forte for Java

Name	*Description*
Frame	AWT Frame (top-level application window).
Dialog	AWT Dialog (modal or non-modal window for user feedback).
Applet	AWT Applet (embeddable program run by a web browser or other applet viewer).
Panel	AWT Panel (container for holding parts of an interface—can in turn be used in any other container, such as a Frame, Panel, Applet, or Dialog).
JFrame	Swing JFrame (top-level application window).
JDialog	Swing JDialog (modal or non-modal window for user feedback).
JApplet	Swing JApplet (embeddable program run by a web browser or other applet viewer).
JPanel	Swing JPanel (lightweight container for holding parts of an interface—within any container, such as a JFrame, JPanel, JApplet, or JDialog).
JInternal-Frame	Swing JInternalFrame (lightweight object with many of the features of a native Frame, used as a "window within a window" within a JDesktopPane).

Note that the table lists only the basic types which differ both in design-time look and in the code generated for the form's class. Customized forms (for example, a dialog box with **OK**

and **Cancel** buttons) and customized user classes (derived from one of the container classes above) can also be created and then saved as new templates. See "Creating your own templates" on page 201.

Working with components

Once you have created a new form, the first thing you probably want to do is to add components to it in order to create functionality. This section describes how to add and modify components in the Form Editor.

For more information on individual Swing components, see Sun's Swing tutorial at http://java.sun.com/docs/books/tutorial/uiswing/components/components.html.

Adding new components

The easiest way to add components is by using the Component Palette. The Component Palette is a toolbar on the Main Window, which holds commonly used visual components that you can add to forms. You can add a component merely by clicking on the component in the Palette and then clicking on the form in the Form Editor window. You can also add your own components to the Component Palette. See "Customizing the Component Palette" on page 221 for more information.

To add a component using the Component Palette:

1 Choose a component in the Component Palette by clicking on its icon. (The add/selection mode icon shown in the figure above will become unselected, showing that the Form Editor is in "add mode".)

Add mode with an AWT Label selected

2 Click on the Form Editor window to add the component to the form. You can add multiple components of the same type by holding down the SHIFT key and clicking multiple times.

The Palette automatically changes to selection mode (the add/selection icon becomes selected) after the component has been added to the form.

Depending on the layout used, you can add a component (or move it by dragging it) to a specific position (Border and Absolute layouts) and/or set the default size of the component by dragging the mouse on the selection handles (Absolute Layout).

To add a component using the Explorer:

1. Under the **Filesystems** tab in the Explorer, find the class of the bean you want to add.
2. Right-click on the class and choose **Copy** from the contextual menu.
3. Choose **Paste** from the contextual menu of either the Component Inspector or the Form Editor.

Selecting components

You can select components in the Form Editor either directly in the Form Editor window when in selection mode (the default mode, shown in the figure below) or by selecting component nodes in the Component Inspector's tree. To select multiple components when clicking on the Form Editor window, hold down the CTRL key as you select them.

Selection mode

The Component Inspector displays the current selection on the active Form Editor window. Its property sheet displays the properties of the selected component – or, if more than one component is selected, their common properties are displayed. To select a component in the Component Inspector's tree, click on it. To select multiple components, hold down the CTRL key and click each one. You can also select a consecutive group of components by clicking on the first one, holding the SHIFT key and then clicking on the last component in the group.

Connection mode

In connection mode, you can start the Connection Wizard, in which you can link two components together with an event.

Connection mode

See "Using the Connection Wizard" on page 135 for more information.

Moving components to a different container

If, when adding a component to your form, the component is added to the wrong container, the easiest way to put the component into the proper container is to cut and paste the component in the Component Inspector.

To move components:

1. Select the item in the Component Inspector or the Form Editor window (hold down the SHIFT or CTRL key to enable selecting of more than one item).
2. Right-click the item(s) and choose **Cut** from the contextual menu.
3. Choose the destination container in the Component Inspector or the Form Editor window and choose **Paste** on its contextual menu.

The components will be moved with all properties and events intact – although events are maintained only if pasting within the same form.

Copying components

If you want to have multiple copies of a component within a form, you can copy and paste the components in the Component Inspector.

To copy components:

1. Select the item in the Component Inspector or the Form Editor window (hold down the SHIFT or CTRL key to enable selecting of more than one item).
2. Right-click the item(s) and choose **Copy** from the contextual menu.
3. Choose the destination container in the Component Inspector or the Form Editor window and choose **Paste** on its contextual menu.

The components will be copied with all properties and events intact – although events are maintained only if copying within the same form.

Reordering components within the same container

A **container** is a Java visual component (for example, a Panel) that can contain other components. (Examples of containers are the AWT Dialog, Frame, and Panel, as well as the Swing JPanel, JDialog and JFrame.) The order of components within their parent container may determine the order in which they appear visually. For some layouts (such as Flow Layout), modifying the order is the only way to modify the way components are laid out.

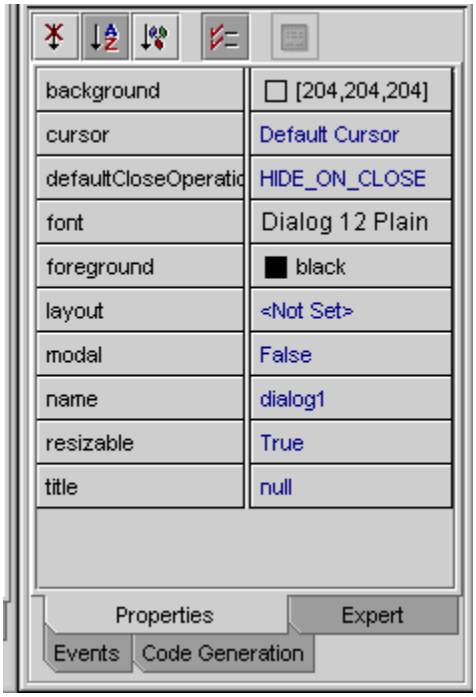
To change the order of components in the Component Inspector, either:

◊ Right-click on the component you want to move, and use the **Move Up** or **Move Down** items in its contextual menu.

◊ Right-click on the parent container and choose **Change Order** from its contextual menu. You will get a dialog box for setting the order of all subcomponents in the container.

Property Sheet pane in the Component Inspector

The lower pane of the Component Inspector displays the property sheet for the object selected in the upper pane. The property sheet contains up to five categories, represented by the tabs **Properties**, **Expert**, **Layout**, **Events**, and **Code Generation**.



Depending on the property, properties can be edited by:

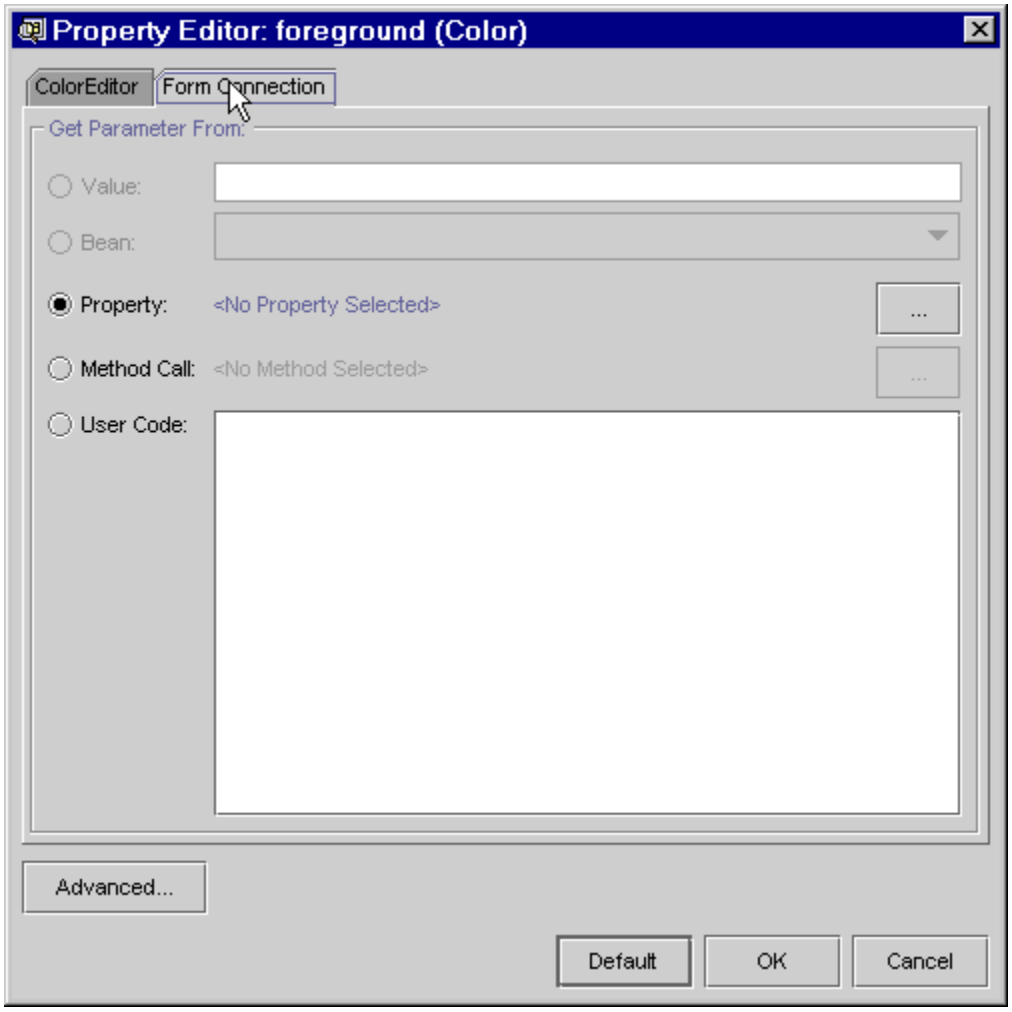
- Clicking on the current value, typing in a new one, and then pressing ENTER.
- Double-clicking on the property name to select from a fixed list of choices.
- Clicking on the value and then selecting from the drop-down list that appears.
- Clicking on the value and then clicking the **...** button that appears. Clicking on this button brings up a custom property editor dialog box.

See "Custom Property Editors" on page 288 for a description of the various custom property editors.

Custom property editors for component properties

When you bring up the custom property editor for a property of a form component, the Property Editor dialog box that appears may have more than one tab. For example, properties on the **Properties** tab have a Form Connection property editor associated with them (see

"Using the Form Connection property editor" on page 139).



Reverting to the default value for a property

If you have changed a value that you later decide you would like to change back to its default, right-click on the property name and choose **Set Default Value** from the contextual menu. If the value was changed in a custom property editor, you can press the **Default** button in the custom property editor to revert to the default.

Working with layouts

A layout manager is a special Java class that manages the positions and sizes of components in a container (Frame, Panel, or any other visual component that can contain other components).

The Form Editor in Forte for Java has advanced support for layout managers. In addition to absolute positioning, complex forms based on Flow Layout, Border Layout, Card Layout, Grid Layout, GridBag Layout or Swing's Box Layout are supported. (See "Standard layout managers" on page 122 for descriptions of these layout managers.)

Various containers come with different predefined layout managers:

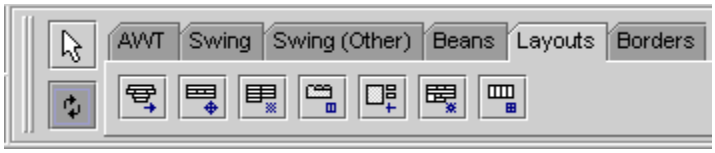
- Frame, JFrame, Dialog, JDialog, and JInternalFrame have the Border Layout by default.
- Panel and JPanel have Flow Layout as the default layout.

Layout managers do not apply to all containers. For example, ScrollPane, JScrollPane, JTabbedPane, JLayeredPane/JDesktopPane, and JSplitPane have their own special layouts which cannot be changed.

Setting and changing layout managers

To change the layout manager:

1 Select the desired layout in the **Layouts** tab of the Component Palette.



2 On the form, click inside the container whose layout you want to change.

Or:

1 Right-click on the target container – either an empty part of the form in the Form Editor window or the node for the container in the Component Inspector.

2 Change the layout using the **Set Layout** submenu.

Note: When you change layouts, the IDE remembers the constraints used on the discarded layout. Therefore, if you change the layout back, it looks the same as it last looked in that layout. The only exception to this is when you switch from Absolute Layout to GridBag Layout. When you switch from Absolute to GridBag, the GridBag constraints are created so that the GridBag Layout looks as close as possible to the previous Absolute Layout. See "Standard layout managers" on page 122 for more information on the various layouts.

Setting layout properties and constraints

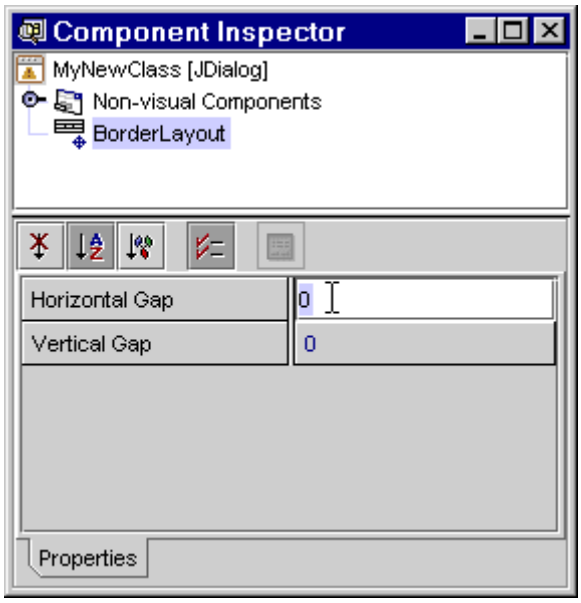
You can modify the appearance of a container on your form by setting general properties for the layout manager and constraints for the individual components in the layout.

Layout properties

The layout properties enable you to modify the layout manager's overall settings, such as the horizontal and vertical gap and alignment. The properties vary depending on the layout manager. Some layout managers do not have any properties.

To set layout properties:

◊ Select the layout manager's node in the Component Inspector. The layout properties will then appear in the property sheet pane.

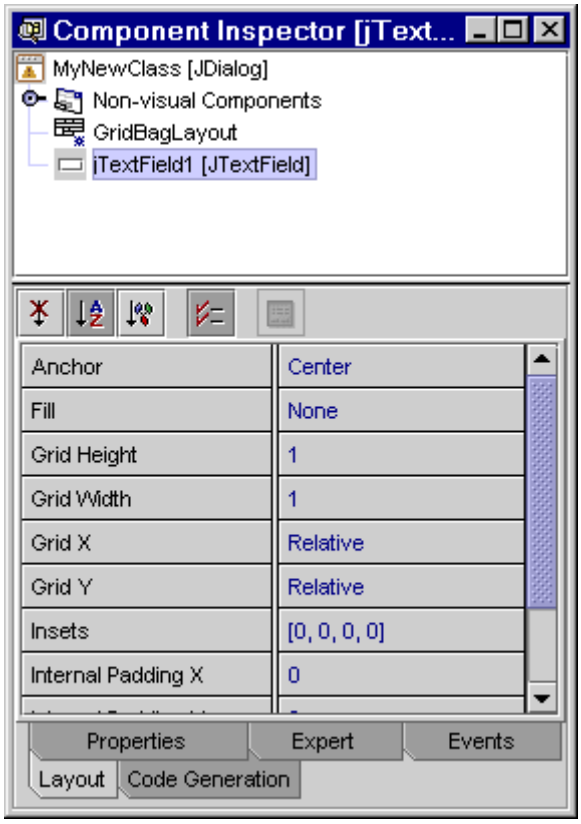


Constraints properties

These are the layout properties specific to each component managed by the layout manager. For example, when Border layout is used, the `Direction` property is set according to which of the five parts of the form the component resides in (the possible values being `North`, `South`, `West`, `East` and `Center`).

To view or set constraints properties:

◊ Select the component in the Component Inspector and choose the **Layout** tab.



Standard layout managers

Border Layout

A Border Layout enables components in a container to be arranged in five positions: `Center` (which expands to fill any empty space left in the other four parts of the component) and along the four sides `North`, `West`, `South`, and `East`. Components in the `North` and `South` positions automatically expand horizontally to take up the whole space, and the `West` and `East` components automatically expand vertically. The horizontal and vertical gap properties control spacing between the components.

Flow Layout

A Flow Layout arranges components in a container like words on a page: it fills the top line from left to right, then does the same with the lines below. You can specify the alignment of the rows (`left`, `center`, `right`). You can also set the horizontal spacing between components and the vertical spacing between rows.

Grid Layout

The Grid Layout divides its container into a configurable number of rows and columns. Like the Flow Layout, components are added to the grid from left to right and top to bottom. But unlike Flow Layout, the dimensions of the grid are fixed; individual components can not have different sizes.

Card Layout

You can think of a Card Layout as a deck of cards. During run time, only one component ("the top card") is visible at a time and each component ("card") is the same size. In design time in the Form Editor, the Card Layout is displayed as a JTabbedPane to make it easier to design the form. This enables you to switch between cards by clicking on their tabs when you are designing the form.

In the property sheet for the `CardLayout` node, you can set the `Current Card` property to `card1`, `card2`, and so on. In code, the methods `show`, `next`, `previous`, `first`, and `last` can be used to select cards. For example, the following code selects the next card in `panel1` when `button1` is clicked:

```
private void button1ActionPerformed(java.awt.event.ActionEvent
      evt) {
  // Add your handling code here:
  CardLayout cards = (CardLayout)panel1.getLayout();
  cards.next(panel1);
}
```

See the JTabbedPane component if you want to design a visual form that has tabbed panes in run time.

GridBag Layout

GridBag Layout (not to be confused with Grid Layout) lets you set almost any layout you want using a complex set of constraints. You can create a GridBag layout one of the following three ways:

- By using Forte for Java's **Customize Layout...** command on a form to bring up the customizer dialog box, which enables you to visually adjust the placement and constraints of the components. See "Using the GridBag customizer" on page 125 for more information.

- By setting the constraints yourself under the **Layout** tab of the property sheet of every component within the layout.

- By creating an Absolute Layout and then switching to GridBag to have the constraints code generated automatically. If you use this option, you might want to go back and

polish up the details using the two previous options.

GridBag Layout is particularly useful for multi-platform Java applications, because it enables you to create a free-form layout that maintains a consistent appearance, even when the platform and look and feel change.

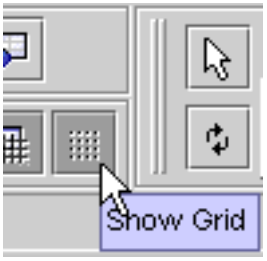
Box Layout

The Box Layout is part of the Swing API. It lets multiple components be arranged either vertically (along the Y axis) or horizontally (on the X axis). Components do not wrap – even when the container is resized. By default they are arranged from left to right or top to bottom in the order they are added to the container. See "Reordering components within the same container" on page 117 for changing the order of the elements. Box Layout is similar to Grid Layout, except that it is one-dimensional (that is, there can be multiple components on one axis but not both).

Absolute Layout

Absolute Layout is a design aid that enables you to place components exactly where you want to in the form and move them around by dragging. Ease of use makes this layout particularly useful for making prototypes. You do not have enter any property settings and there are no formal limitations within the layout. It is a substitute for "null" layout (for which there is no layout constraints object) and provides and places components in absolute positions more cleanly.

You can have a grid displayed on the Form Editor window when using Absolute Layout by pressing the **Show Grid** icon.



Important: Absolute Layout is not recommended for production applications. The fixed location of components in the form does not change, even when the environment changes. Therefore significant distortions in appearance can occur when such an application is run on a different platform or using a look and feel different than the one in which it was created. If you design a form using Absolute Layout, it is recommended that you switch the layout manager to GridBag Layout and then fine tune it before you distribute the application.

Note: Absolute Layout is not in the standard Java layout sets. Rather, it is provided with the IDE and can be found at `com.netbeans.developer.awt.AbsoluteLayout` and

`com.netbeans.developer.awt.AbsoluteConstraints`. If you do keep Absolute Layout in your application, these two classes must be distributed with it. They are ready for deployment in the `AbsoluteLayout.zip` file (in the `%FORTE4J_HOME%/lib/ext` directory). You can freely redistribute this archive with applications developed with the Forte for Java IDE. The source code is also redistributable and is available in `%FORTE4J__HOME%/sources/com/netbeans/developer/awt`.

Null Layout

You can also disable Absolute Layout and design forms with a null layout if you wish.

To use null layout:

1. Choose **Tools | Global Options...** from the main menu, select the `Form Objects` node and switch the value of its `Generate Null Layout` property to `True`.
2. When designing your form, select the Absolute Layout layout manager.

Using the GridBag customizer

Forte for Java, Community Edition provides a customizer to simplify the creation of GridBag layouts.

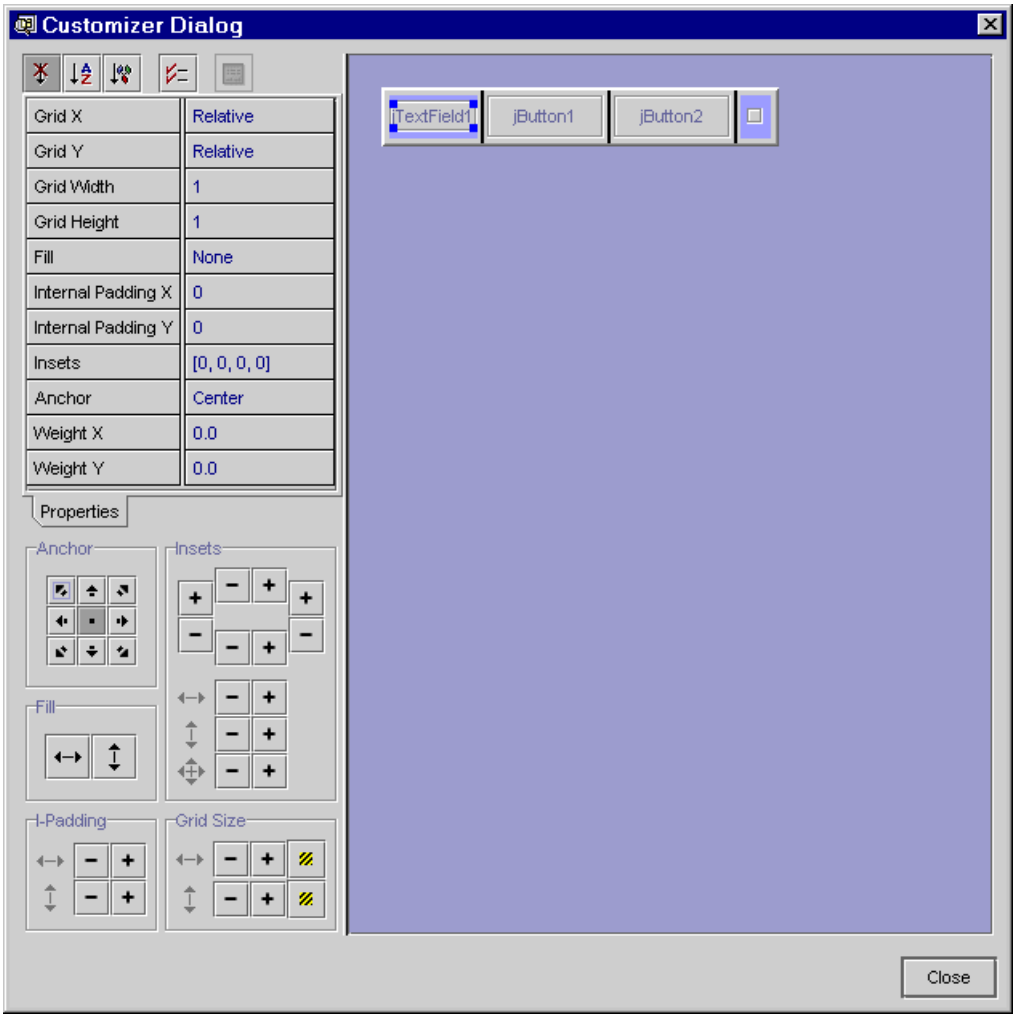
To use the visual GridBag customizer:

1. Add the components you want to use and make sure that you have GridBag Layout set.

Note: It is helpful to sketch out the way you want your layout to look before you open the customizer.

2. Right-click on the `GridBagLayout` node in the Component Inspector and choose **Customize Layout...** from the contextual menu, or choose the **Customize** button on the property sheet pane. The Customizer Dialog will open with a property sheet for GridBag constraints and buttons for adjusting the values of the constraints in its left pane and a

rough depiction of the placement of your components on the form in its right pane.



3 You can reposition the components in the right pane by dragging them. As you drag the component, its `Grid X` and `Grid Y` properties will change to reflect its new position. (Be aware, however, that this panel only serves as a rough guide and does not reflect the absolute positions of the components. The position of each component is largely governed by other constraints set in the left pane. The Form Editor window more closely reflects how the components will look.) You can also change these values manually in the left pane.

4 Once you have the approximate position of the components, you can adjust the other constraints of each component in the left pane. Follow these steps (in whatever order you prefer) to position each component:

   a. Select a component in the right pane.

   b. Further adjust its horizontal and vertical position if necessary by setting its X and Y grid positions.

   c. Adjust the `Grid Width` and `Grid Height` parameters to determine how many grid positions are allocated for the component in each direction. The values of

`Remainder` (allocates the rest of the space in the given row or column for the component) and `Relative` are also available.

You can also adjust these settings with the Grid Width buttons. Pushing the right-most button with yellow shading in each group sets the value to `Remainder`.

d. Adjust the weight settings of the component to determine how much space it should be given relative to that of the other components. For example, a component with a `Weight X` value of .5 has twice as much horizontal space allotted to it as a component with a value of .25 for this parameter. The sum of the values of the components in a given row or column should not add up to more than one. When a form is resized, these settings affect which components are resized (and by how much). Components with a value of 0 for one of these parameters retain their preferred size for that dimension.

e. Adjust the insets for the component. The insets determine the amount of free space on each of the four sides of the component.

You can enter numbers for these manually or use the inset buttons on the bottom part of the left pane. These buttons are divided into four sets. The top group enables you to increase and lower the inset for each side separately. The second group enables you to change the left and right insets simultaneously. The third group lets you change the top and bottom insets simultaneously. The fourth group enables you to change all insets simultaneously.

As you change the insets, you will see the inset area marked by yellow change in the right pane.

f. The internal padding settings enable you to increase the horizontal and vertical dimensions of the component. You can adjust these by directly entering numbers for the properties or by using the internal padding buttons.

As you adjust these constraints, you can see the selected component in the right pane expand or contract vertically and/or horizontally, according the changes you make.

g. The `Fill` constraint enables you to choose whether the component will use all of the vertical and/or horizontal space allocated to it. Any space allocated to a component that the component does not fill is marked with blue in the right pane.

h. The `Anchor` constraint enables you to place the component in one of nine positions within the space allocated to it (`Center`, `North`, `NorthWest`, and so on). This setting has no effect if there is no free space remaining for the component.

Support for custom layout managers

It is also possible to use custom layout managers in the Form Editor. Any layout manager with a default constructor and that does not use constraints for adding components can be used in the Form Editor when designing.

Important: When adding custom layout managers that do not follow these criteria, you must follow the specifications of the Forte for Java Layout Managers API in order to be able to use

the layout manager in the Form Editor. See the NetBeans web site for more information on obtaining and using the Forte for Java Layout Managers API.

To install a custom layout manager

1. Add the directory or JAR archive containing the layout manager's source or class to the IDE by choosing **Tools | Add Directory** or **Tools | Add JAR** from the main menu.
2. Install the custom layout manager in the Component Palette by right-clicking on the layout manager's class in the Explorer or Object Browser and choosing (**Add to Component Palette...** from the context menu).
3. In the Palette Category dialog box that appears, select the **Layouts** category and click **OK**.

Working with source code

The Editor window displays the code for the active opened form. The source code is always synchronized with the visual appearance of the form. Every change made in the Form Editor window or the Component Inspector is immediately reflected in the source code.

Guarded text

Some parts of the source code generated by the Form Editor are not editable directly in the Editor. The background of these guarded blocks is shaded. Guarded text includes:

- The block of variable declarations for the components on the form.
- The `initComponents()` method, in which all the form initialization is performed (this method is called from the form's constructor).
- The header (and trailing closing brace) of all event handlers.

Though the `initComponents()` method is not editable manually, you can affect the way it is generated by editing properties found on the **Code Generation** tab of the component's property sheet and by using the Form Connection custom property editors for the components in your form. See "Modifying generated code for components" below and "Using the Form Connection property editor" on page 139.

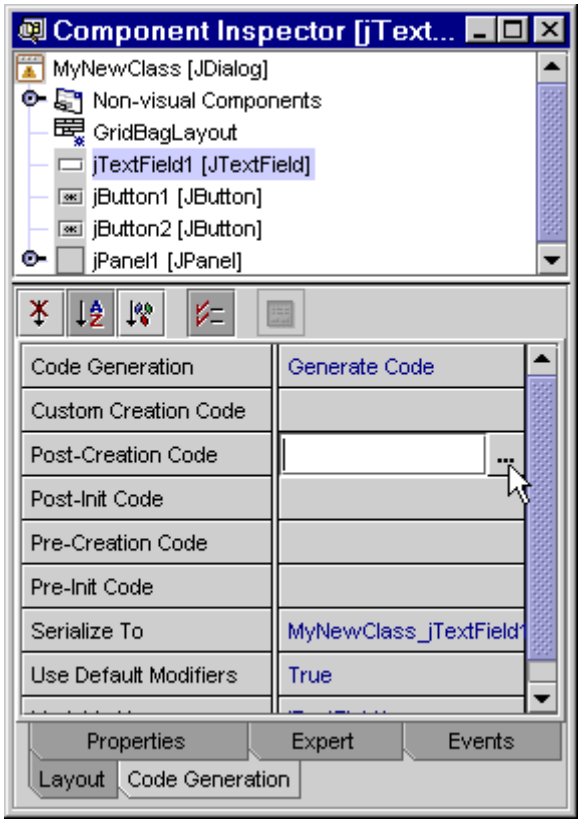
Modifying generated code for components

You can use a component's property sheet to affect the way the code is generated in the `initComponents()` method. In addition, you can write code and specify where it should

be placed within `initComponents()`.

To modify code generation for a component:

1. Select the component in the Component Inspector.
2. Click the **Code Generation** tab to access the Code Generation properties.



In the `Custom Creation Code` property, you can enter your own creation code for the component (not including the variable name and equal sign (=)). If this property is left blank, the IDE generates default creation code for the component.

You can change the variable name by modifying the `Variable Name` property.

You can also add code before and after a component's creation code and before and after its initialization code using the `Pre Creation Code`, `Post Creation Code`, `Pre Init Code`, and `Post Init Code` properties.

Note: The IDE always places creation code before initialization code in `initComponents()`.

For information on adding pre- and post-initialization code for a component's property, see "Using the Form Connection property editor" on page 139.

Tip: You can also perform additional initializations of a form's components in the constructor

after the call to `initComponents ()`. For example, here you might initialize a button group or set component labels from resource bundles.

Other code generation options

In addition to the custom code creation properties, the following options are available under the **Code Generation** tab of each component's property sheet in the Component Inspector:

- **Code Generation** – enables you to choose between generating standard or serialization code for the component.
- **Serialize To** – enables you to set the name of the file for the component to be serialized to (if it is serialized).
- **Use Default Modifiers** – If `True`, the component's variable modifiers will be generated according to the default modifiers set in the `Variables Modifier` property set in the property sheet for `Form Objects` in the Global Options window. If `False`, the `Variable Modifiers` property will appear on the property sheet enabling you to set the modifiers.

Code Generation properties for containers

A set of Code Generation properties is also available for containers. These properties affect the size, positioning, encoding, and active menu bar of the form.

- **Form Encoding** – The encoding to be used in the XML `.form` file.
- **Form Size Policy** – Overall policy for resize code generation. You can choose to have resize code generated, to not have resize code generated, or to have the `pack ()` method generated (in which case the form would be sized to fit the preferred size and layout of its subcomponents).
- **Form Position** – Position on the screen where the form will open (coordinates from top left corner). This is applied only when the `Generate Position` property is true and `Form Size Policy` is set to `Generate Resize Code`.
- **Form Size** – Initial size of the form. This is applied only when the `Generate Size` property is true and `Form Size Policy` is set to `Generate Resize Code`.
- **Generate Center** – If `True`, generates code that sets the position of the form to the center of screen (regardless of the value of the `Generate Position` property).
- **Generate Position** – If `True`, generates code that sets the position of the form to the coordinates given by the `Form Position` property (or to the center of screen if

`Generate Center` is set to `True`).

- **Generate Size** – Generates code that sets the size of form to the one given by the `Form Size` property.
- **Menu Bar** – Drop-down list enabling you to choose which menu bar is to be set in the form.

External modifications

Forms are stored in two files:

- A `.java` file, which contains the (partly) generated Java source.
- A `.form` file, which stores layout manager properties, the properties and layout constraints of JavaBeans components on the form, and other information. This file does not need to be distributed with your application. It is merely used to display the form in the Form Editor.

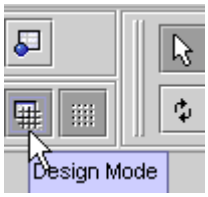
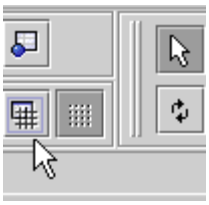
You can edit the `.java` files using external editors (not simultaneously while the form is being edited in the IDE), with the following exceptions:

- Do not modify the content of the `initComponents ()` method. The body of this method is always regenerated when the form is opened.
- Do not remove or modify any of the special comments placed in the source by the IDE's Form Editor (`// GEN-...`). They are required for the form to open correctly. (These comments are not visible inside Forte for Java's source editor.)
- Do not modify the headers or footers of event handlers.

Form Editor modes

The Form Editor can be in one of three modes:

Table 8: Form Editor modes

Toolbar	Description
	*Design mode.* This is the default Form Editor mode. By clicking on the form you can select, add, or drag components. Note that in this mode, depending on the layout, the layout won't necessarily look the same as it does during run-time. This is because design would be difficult if every layout manager worked exactly the same way at design time as it does during run time. For example, with some layout managers, when you drop two components next to each other, they could resize automatically and make it impossible to add a new component between the two. So, if you need to see the effect of a certain layout manager at run time, change the Form Editor from design mode to either real mode or test mode.
	*Real mode.* The Form Editor window displays the actual Java layout managers, so the forms look the way they will during run-time. You can select or add components, but dragging components is disabled.
	*Test mode.* Similar to real mode, the form behaves exactly the same as during run-time. When clicking on the form, the mouse events are delivered to the actual components. Thus, for example, a button looks "pressed" when you click on it. This mode is suitable for checking the look and feel of the form without the need to compile and run it.

To switch between design and real mode, use the **Design Mode** icon (the square with a pound sign inside) from the toolbar. If the icon is selected (as shown in the design mode example above), you are in design mode. Otherwise, you are in real mode.

Switching to test mode is done using the **Test Mode** icon on the toolbar; this is a toggle button, which means you click to enable it and click again to disable. Note that switching test mode on disables the **Design Mode** icon; to enable it again, test mode must be switched off.

These modes are kept separately for each form.

Note: When test mode is switched on, the form is resized to its preferred size – this is how it looks if it is not explicitly resized by calling `setSize()` or `setBounds()`.

Events

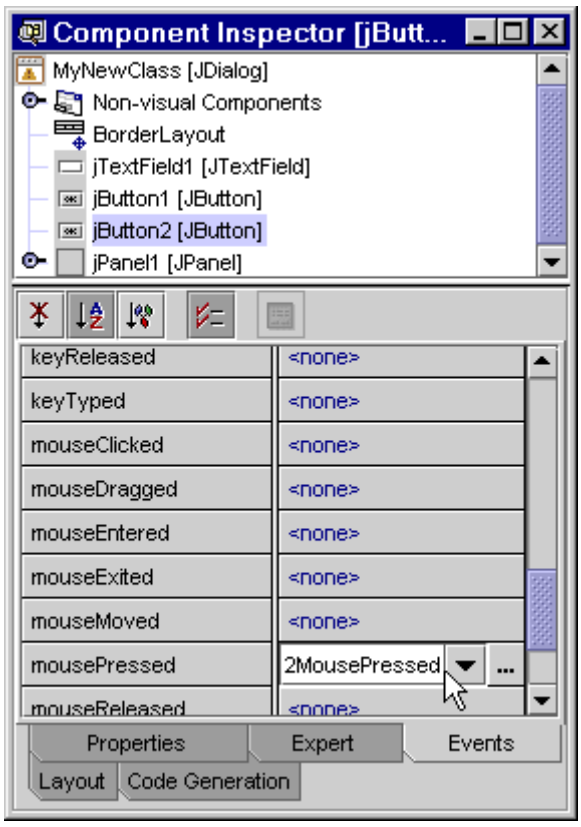
Events are managed through the component's **Events** tab in the Component Inspector, which gives a list that matches each event fired by an object with its corresponding handler method. There is also an **Events** entry on object contextual menus in both the Component Inspector and Explorer.

A new component does not have event handlers defined; the values are all <none>.

Defining event handlers

To define an event handler using the property sheet:

1. Select the **Events** tab in the form object's property sheet (either in the Component Inspector or the Properties window).
2. Click on the value (it should be <none>) of the one of the events in the list.



3. Choose a name for the event by:

   - Pressing ENTER to have the default name automatically generated.
   - Typing a new name over the default name and pressing ENTER.

After you press ENTER, the code for the listener and the (empty) body of the handler method will be generated. If you do not press ENTER, your changes will be ignored.

To define an event handler using the contextual menu:

1. Expand the source file's node in the Explorer and then right-click on its form object subnode.
2. Choose **Events** and then move through the two submenus to choose the event. The default name will be given to the event.

If multiple events are of the same type (for example, `focusGained` and `focusLost` are both of type `java.awt.event.FocusEvent`), then you may use the same handler for all of them. For example, you could set both `focusGained` and `focusLost` to use the `button1FocusChange` handler. You can also have the same event on several components share a handler.

When you enter a new name for an existing handler, the code block is automatically edited to use the new name. When you delete a name, the code block is deleted. If more than one handler uses the same name (and the same block of code), deleting a single reference to the code will not delete the code; only deleting all names will delete the corresponding code block, and a confirmation dialog box will be displayed first.

Note: If you delete an event but do not delete the event handler when prompted by the confirmation dialog box, an orphaned handler will remain in the guarded text. If you wish to delete the orphaned handler, you will have to add another event, attach it to the handler, delete that event, and then agree to delete the handler in the confirmation dialog box.

Removing event handlers

To remove an event handler:

1. Select the object in the Component Inspector and select the **Events** tab on the property sheet.
2. Select the event in the property sheet, delete all of the text of its name, and press ENTER.

   The value `<none>` will reappear in the value field for that event.

Renaming event handlers

To rename an event handler:

1. Select the object in the Component Inspector and select the **Events** tab on the property sheet.

2 Select the event in the property sheet and then press **...** button that appears to bring up the Handlers custom property editor.

3 In the Handlers For... dialog box, click **Rename** and then enter the new name for the handler in Rename Handler dialog that appears.

Adding multiple handlers for one event

The IDE also enables for the possibility of adding multiple handlers for one event with a custom property editor.

To add multiple handlers for an event:

1 Select the object in the Component Inspector and select the **Events** tab on the property sheet.

2 Select the event in the property sheet and then press **...** button that appears to bring up the Handlers custom property editor.

Using this editor, you can add and remove multiple handlers.

Using the Connection Wizard

The Connection Wizard enables you to set events between two components without having to write much (or any) code by hand. When you choose two components to be "connected", the first is the one that will fire the event, and the second one will have its state affected by the event.

To start the Connection Wizard:

1 Switch to connection mode on the Component Palette (see "Connection mode" on page 115).

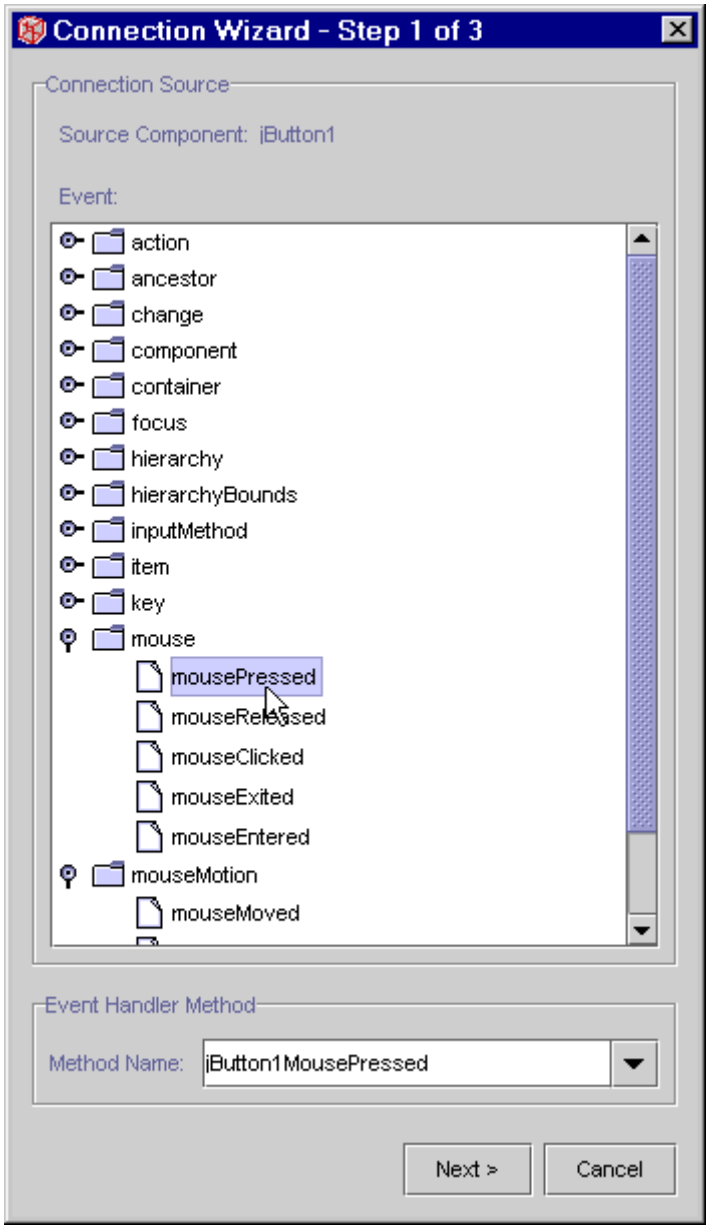
2 Next, click on the two components (first the component with the source event and then the target component). You can click on the form—or click in the Component Inspector (which enables you to also use non-visual components as parts of the connection).

The Connection Wizard opens to connect the chosen components.

The Connection Wizard enables you to set the connection parameters in two or three steps:

Connection Source

The first step enables you to choose the event of the source component on which the operation is to be performed and the name of the event handler method generated.

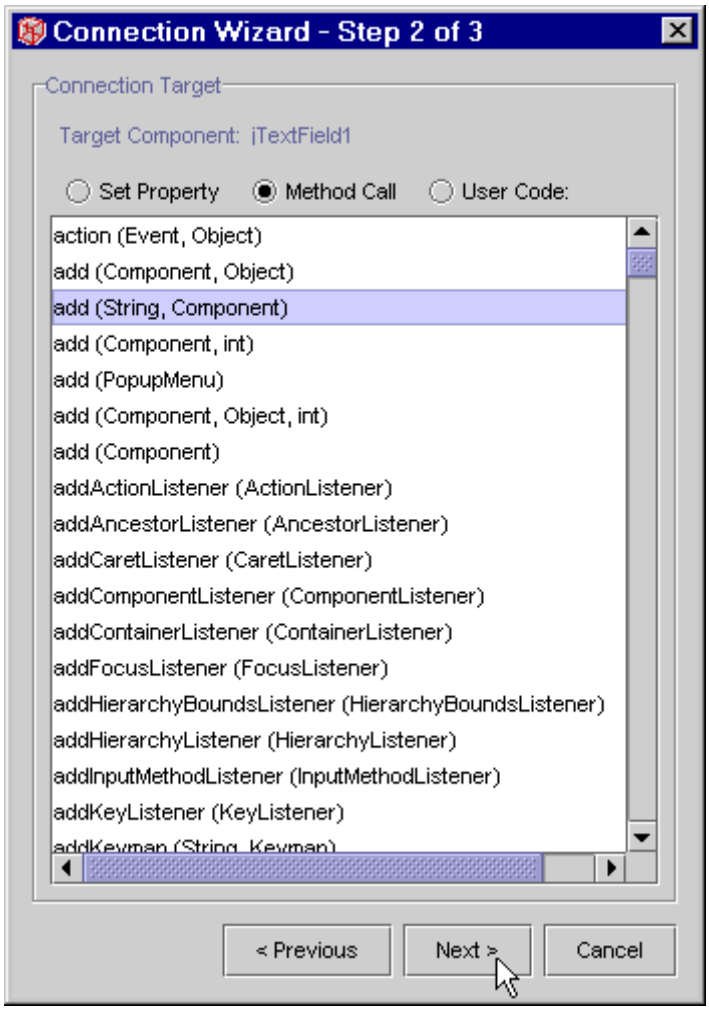


Note: If you choose an event that already has an event handler attached (the name of the event handler is visible in the tree of events after the event name – like `actionPerformed [button1ActionPerformed]`), when clicking **Next**, you will be asked if you want to delete the old event handler. If you answer yes, the old event handler is replaced with the connection code when the Connection Wizard is finished.

Connection target

The second step enables you to specify the operation that is to be performed on the target component. There are four options which can be changed using the radio buttons along the top of the dialog box:

- **Set Property** – enables you to set a property on the target component. In the next step you specify the property value.
- **Method Call** – enables you to call a method on the target component. In the next step you specify the parameters for the method call.
- **User Code** – enables you to write the code by hand. When you choose this option, the **Next>** button changes to **Finish**, meaning there are no settings remaining in the wizard and the target component is ignored by the wizard. The insertion point in the editor is then placed into the new event handler enable you to write the handling code by hand.



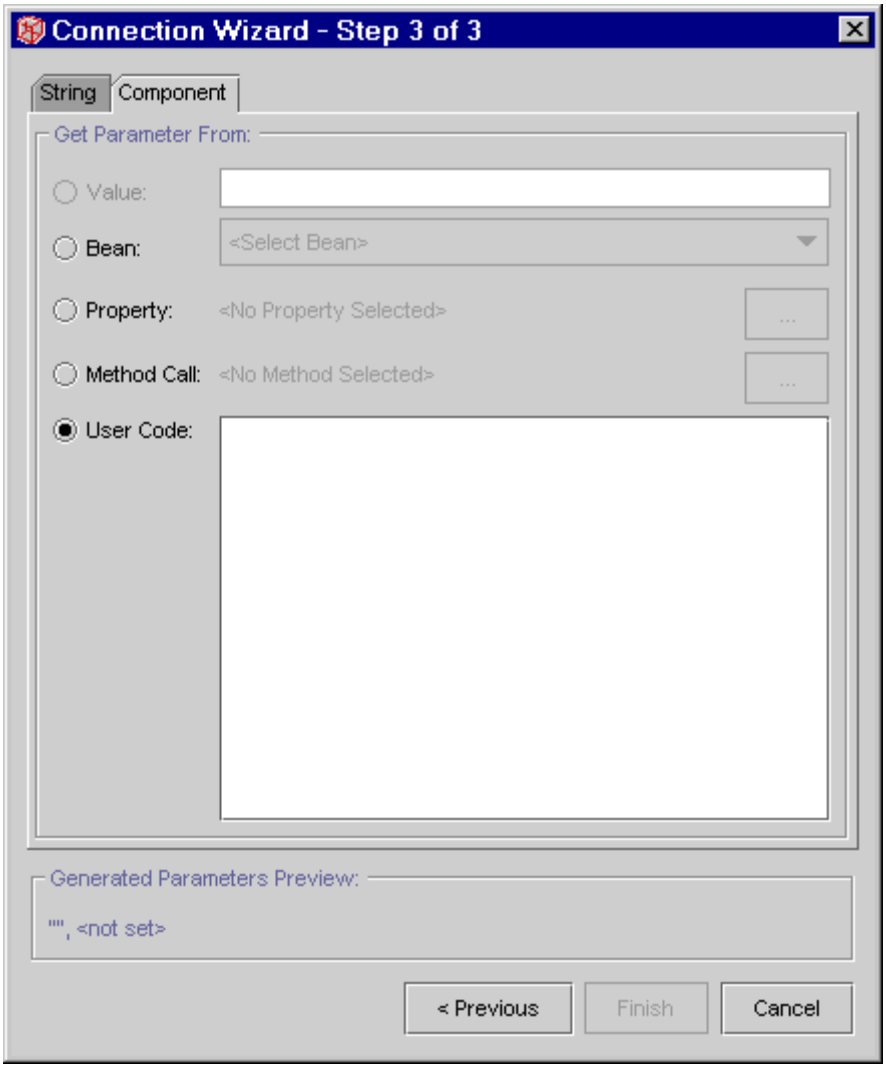
Choosing property value or method parameters

If you have chosen **Set Property**, **Method Call**, or **Bean** in the previous step, the third step will enable you to specify the values for the target property or parameters for calling the target method. If you have chosen **Method Call**, the dialog box will display a list of all parameters types as tabs, where each tab represents a single method parameter.

On each tab, you can set the source from which the value (of the specified type) is acquired:

- **Value** – This option is available only if the value is a primitive type (`int`, `float`, and so on) or a `String`. You can enter the value into the text field.
- **Bean** – This option enables you to acquire the value from a bean on the form. If you click on the **...** button, a dialog box will appear which will let you choose the bean.
- **Property** – This option enables you to acquire the value from a property of a component on the form. If you click on the **...** button, a dialog box will appear which will let you choose the component and its property. Note that only properties of the correct type are listed.
- **Method** – This option enables you to acquire the value from a method call of a component on the form. If you click on the **...** button, a dialog box will appear which will let you choose the component and its method. Note that only methods which return the correct type and which do not take any parameters are listed.
- **User Code** – This option enables you to write user code which is used as the parameter

for the setter of the chosen property or a parameter of the chosen method call.



If you click **Finish**, a new event handler will be generated with code reflecting the settings entered in the second and third step.

Using the Form Connection property editor

The Form Connection Property editor gives you greater control over the generated initialization code for your forms. You can have properties of components on your form initialized in a more flexible way than just setting static values. This editor enables you to initialize property values from:

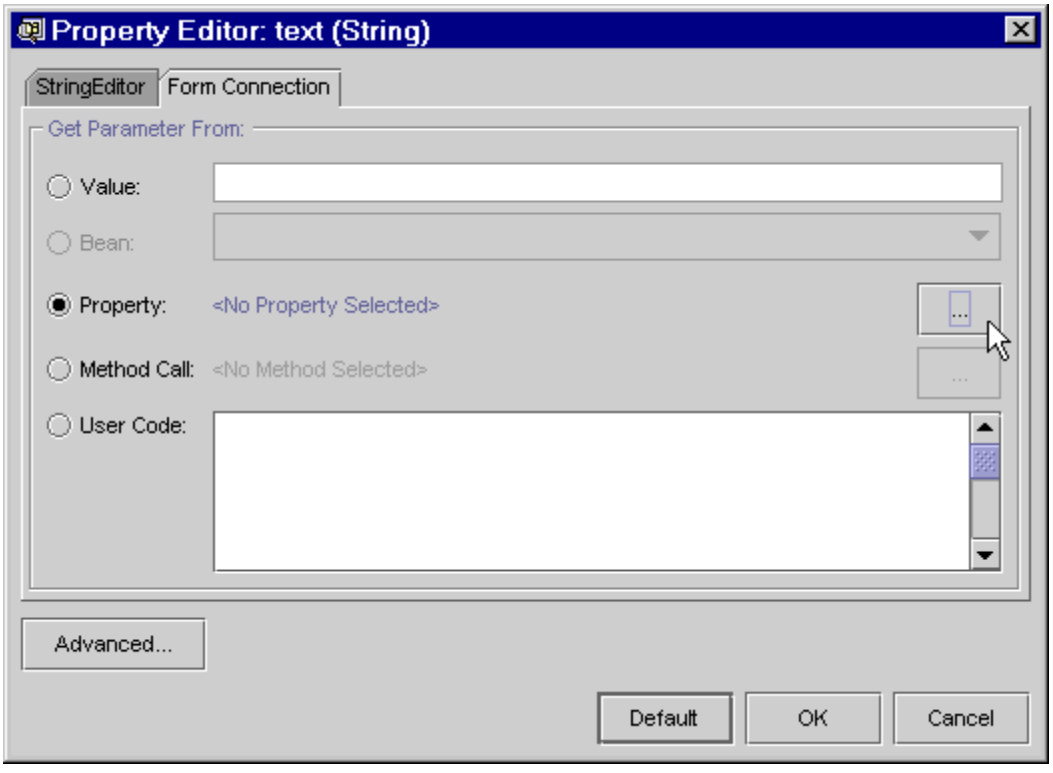
- A static value you define.

- A JavaBeans component.
- A property of another component on the form.
- A call to a method of the form or one of its components (you can choose from a list of methods that return the appropriate data type).
- Code you define, which will be included in the generated code.

To use the Form Connection property editor:

1. With the Component Inspector open on a form, select the property (under the **Properties** tab) that you would like to modify the initialization code for.
2. Press the **...** button to bring up the custom property editor multi-tab dialog box and select the **Form Connection** tab.
3. In the dialog box, choose the type of code you would like to add (**Value**, **Bean**, **Property**, **Method Call**, or **User Code**) and then add the code in the field provided.

   If you select **Property** or **Method Call**, press the **...** button to bring up a dialog box with a list of properties or methods available. If you select **Bean**, use the combo box to choose from the available beans.



4. Once you have made your choice, click **OK**.

Pre- and post-initialization code

In the Form Connection property editor, you can also write code to be placed before or after the initialization code for the modified property.

To place code before or after a property's initializer:

1. In the property's Form Connection property editor, click **Advanced...** to bring up the Advanced Initialization Code dialog box.
2. Click either or both of the checkboxes (for **Generate pre-initialization code** or **Generate post-initialization code** and enter the code you want to have generated in the appropriate text fields.

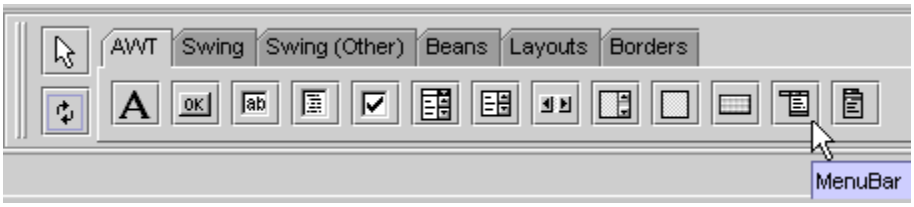
For information on adding pre- and post-initialization and pre- and post-creation code for components, see "Modifying generated code for components" on page 128.

Menu editor

The Form Editor can be used to create and modify menus. The AWT and Swing variants of both `MenuBar` and `PopupMenu` are supported.

Creating a menu bar

To create a new menu bar, click on an AWT or Swing menu bar component in the Component Palette, as shown below. Then click anywhere on the form on which you want to add the menu. The menu will appear under the `Non-visual Components` list in the Component Inspector (since it cannot be manipulated by the layout manager).



If this is the first menu bar you have added to this form, it will also appear visually on the form. Note that only the AWT `MenuBar` can be used as the main menu for AWT forms. The same is true for Swing forms and the `JMenuBar`. It is possible to add the AWT `MenuBar` to Swing forms and vice-versa, but you will not be able to use it as the displayed main menu for

the form.

You can add multiple menu bars to one form, but only one of these can be used as the current menu bar. (You can write code to switch the current menu bar when the form is running.) To change the current menu, select the form's node in the Component Inspector, select the **Code Generation** properties tab, and choose the menu you want active from the drop-down list on the `Menu Bar` property. You can also switch the menu bars in user code while the form is running.

Adding menus to the menu bar

Newly created menu bars come with one menu.

To add more menus:

1. Right-click on the menu bar under the `Non-Visual Components` node of the form in the Component Inspector.
2. Choose **New Menu** (or **New JMenu** if you are using Swing components) from the contextual menu.

Creating a popup menu

To add a popup menu to a form:

1. Click on the popup menu icon under the **Swing** or **AWT** tab in the Component Palette.
2. Click anywhere on the form to add it.

The new menu will appear under the `Non-visual Components` list in the Component Inspector (since it is not laid out by the layout manager). You can use the same editing features as with a menu bar. To use the popup menu visually, write code like this:

```
popupMenu1.show (component, 100, 100);
```

where you want to show the popup menu. The API documentation that comes with the Java 2 SDK has more details about displaying popup menus in its specifications for PopupMenu and JPopupMenu.

Adding menu items

The Component Inspector displays the hierarchy of menu bars and popup menus under `Non-visual Components`. Every menu (whether it is on a menu bar or it is a popup menu or submenu) starts with one menu item, displayed as a subnode.

To add new menu items, submenus, or separators to a menu bar or popup menu:

1. In the Component Inspector, select the menu you would like to add an item to.
2. Right-click on the menu and choose **New** and then the item you want from the submenu.

Menu item events

You can use the same mechanism for assigning event handlers to menu items as with any other component. For menu items, you can also add the `actionPerformed` event handler by actually selecting the item from the menu in the Form Editor window.

To add an event handler to a menu item:

1. Select the component in the Component Inspector and either use the popup menu or event properties to attach the event handler.
2. Use the Connection Wizard by clicking on the **Connect** icon and then (in the Component Inspector) clicking on the connection source and target. See "Using the Connection Wizard" on page 135.

Components with special support

Forte for Java, Community Edition's Java-based architecture enables it to support interactive design with components including the AWT ScrollPane and the Swing JScrollPane, JTabbedPane, JTable, JList, JDesktopPane, and JSplitPane.

In general, these components are used the same way as other components. Forte for Java provides special support in the form of features such as custom property editors which simplify some of the more complex aspects of designing forms.

JScrollPane, ScrollPane

In many cases, AWT components pop up their own scroll bars if their content needs to scroll. An AWT ScrollPane is available, though, to add scrolling where it is needed. Swing components, however, must be added to a JScrollPane if the content is to scroll. This

document will not cover other differences between ScrollPane and JScrollPane. See the Java reference materials and tutorials for these particular classes. We will discuss JScrollPane as it is more frequently needed (and it is generally preferable to use Swing components).

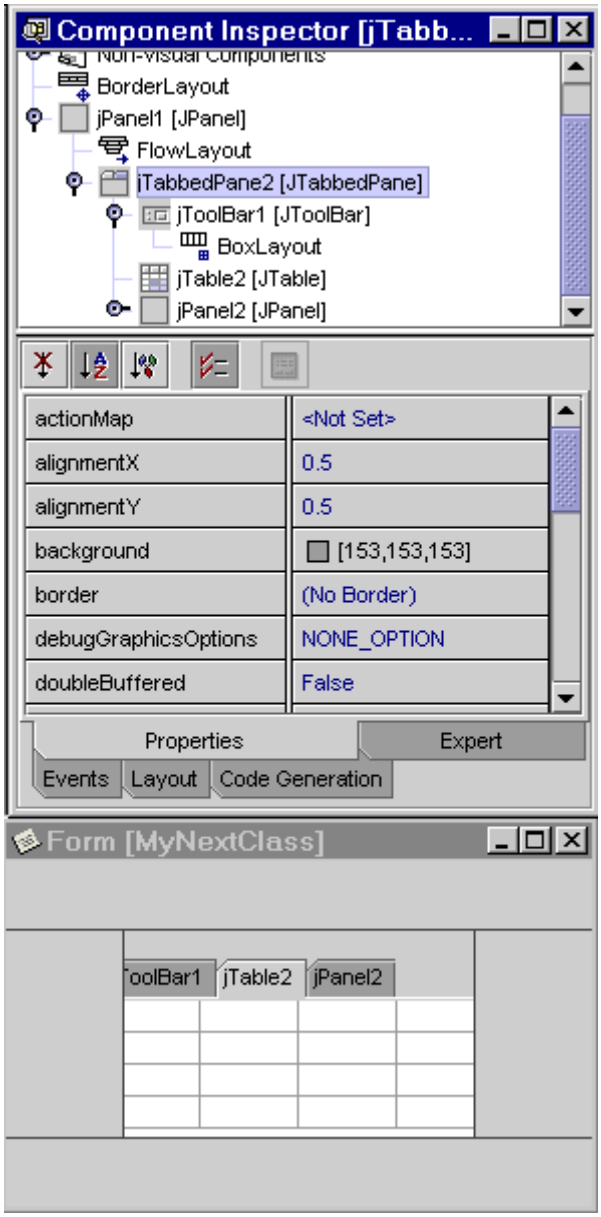
To use a JScrollPane, choose it from the Component Palette and click on the Form Editor window. Then select the component to be added from the Palette to the JScrollPane and then click on the JScrollPane.

Remember that you can still select the underlying JScrollPane (for instance, to set its scrolling properties) by clicking on its entry in the Component Inspector.

JTabbedPane

A JTabbedPane manages multiple components (usually JPanels with their own layouts and sub-components) within its pane. The IDE automatically gives each component its own labeled tab. When the user clicks on the tab, its component is selected and comes to the front of other components.

JTabbed Pane in the Form Editor



The above figure shows a Form Editor window over the Component Inspector window. A JPanel has been added to the JFrame, a JTabbedPane has been added to the JPanel, and three components have been added to the JTabbedPane.

By default, each tab is named for its own component. Each component in the JTabbedPane has its own layout settings, where the tab parameters (`Tab Icon`, `Tab Name`, `Tab ToolTip`) are set.

Note: Be careful where you click when adding new components to a JTabbedPane. If you click on an existing tabbed component, your new component might be added to the existing component rather than to its own new tab.

JTable, JList

A JTable is a grid of cells, each cell having its own content. A JList is similar but with just one dimension. The special support for JTable and JList consists of a custom property editor that controls the number of rows (and columns), the titles, object types, and so on.

To open the custom property editor for a JTable or JList:

1. Access the object's properties (for example, through the Component Inspector).
2. Choose the **Properties** tab.
3. Click on the **...** button for the entry for the `model` property.

The JTable custom property editor has tabs for **Table Settings** and **Default Values**.

The JList custom property editor enables you to add, delete, and rearrange list items.

Note: These custom editors only provide the ability to edit simple models of relatively fixed structure. More demanding applications can not use these visual editors. For live data with changing structure, create your own model in code and assign it to the table or list in the form's constructor or as the value for the `model` property using the Form Connection property editor.

MDI Applications: Using JDesktopPane and JInternalFrames

Swing makes MDI (Multiple Document Interface – windows within windows) easy to implement. The model is like traditional computer windowing systems. In a windowing system, you have a desktop with windows on it. With the Swing MDI, you use a JDesktopPane with JInternalFrames on it. The user can position the JInternalFrames like windows on a traditional desktop as well as resize, close, and minimize them. Forte for Java, Community Edition lets you lay out and define all of this interactively.

You typically start with a JFrame and then add a JDesktopPane to it. To add JInternalFrames, select them from the **Swing (Other)** tab of the Component Palette and click on the JDesktopPane. This adds internal frames with fixed structure. Alternatively, you can create separate forms for the types of frames you want, and then (construct and) add these in code to the desktop pane, thus giving you more flexibility.

You can add other components to the JDesktopPane such as a JTable or JSlider. However, these have their standard properties, and users can't manipulate them in the same way that they can manipulate a JInternalFrame containing those components.

JSplitPane

A JSplitPane has two sides, each of which might have a component placed in it. A user can move the divider between the sides to make one side bigger and the other smaller. As with other special components of this type, Forte for Java enables you to manipulate it during design – for instance, to drag the divider. You can change the orientation of the divider from vertical to horizontal with the `Orientation` property (under the **Properties** tab of the property sheet). You may find that it is easiest to select the JSplitPane itself by clicking on its entry in the Component Inspector.

Configuring the Form Editor

It is possible to configure the behavior of the Form Editor and the appearance of the Form Editor window under the `Form Objects` node in the Global Options window. See "Form Objects reference" on page 269 for a description of configurable options. To open the Global Options window, select **Tools | Global Options** from the main menu.

Chapter 7

Developing Java Server Pages

Forte for Java, Community Edition provides support for creating, compiling, and editing Java Server Pages (JSPs). In addition, you can use the JSP Wizard to create standard and error-handling JSPs, using JavaBeans properties for dynamic content display and server-side processing.

Creating and editing Java Server Pages

To create a new JSP file:

1. Choose **File | New from template** from the main menu.
2. In the Create From Template wizard that appears, expand the node of the `JSP and Servlet` folder, select `JSP` and click **Next**.
3. In the next step of the wizard, choose the name and folder (package) of your JSP file and click **Finish**.

The JSP file will be created in the specified package.

To edit a JSP file:

1. Double-click on the file in the Explorer or Object Browser.
2. Type your JSP code directly into the Editor window that appears.

To create a JSP with the JSP Wizard:

◊ Choose **Tools | JSP Wizard** from the main menu.

Note: There is a default set of abbreviations especially for Java Server Pages that you can use when editing JSP files. When you type one of the designated abbreviation and then enter a space, the abbreviation is expanded into a longer string. For a list of default abbreviations, see "Default Editor Abbreviations" on page 241. For information on changing or adding abbreviations, see "Customizing Editor abbreviations" on page 211.

Compiling JSPs

Once you have edited your JSP file, you can validate it by compiling it into a class file. The compilation proceeds in two stages: First, the JSP page is translated into a servlet. Then the servlet is compiled into a class file.

To compile a JSP:

◊ Right-click on the JSP file in the Explorer or Object Browser and choose **Compile** from the contextual menu.

Handling compilation errors

Any compilation errors encountered in either of these stages are displayed in the compiler Output Window. If the translation to servlet succeeds, you can view the servlet source to better identify and analyze any errors which may have occured in the second stage.

To analyze servlet source for compilation errors:

◊ Right-click on the JSP file in the Explorer or Object Browser and choose **View servlet** from the contextual menu.

The Editor window will appear with the servlet code displayed. Once you determine the error, you can fix it in the JSP page. The servlet itself may not be modified, since it will be regenerated the next time you compile the JSP.

Selecting a compiler

You can choose which compiler to use for compiling the servlet into a class.

To select a compiler for the JSP:

◊ In the file's property sheet, select the **Execution** tab, select the `Servlet Compiler` property, press the drop-down arrow that appears, and choose the compiler from the combo box.

Included beans, others JSPs, and error pages in compilation

If you compile a JSP page that calls any JavaBeans components (with the `<jsp:useBean>` tag), these beans will be compiled along with the JSP page.

Similarly, if your page references any other JSP pages by using the `<jsp:include>` or `<jsp:forward>` directives, the referenced pages will also be included in the compilation.

If your JSP uses an error page (by specifying `<%@page errorPage %>`), the error page will be compiled with your page.

Note: Currently Forte for Java supports version 1.0 of the JSP specification. The built-in server used for testing and running JSP pages is Sun JavaServer Web Development Kit version 1.0.1.

Running JSPs

After successfully compiling your JSP page, you can run it using the built-in web server.

To run a JSP page:

◊ Right-click on the JSP's node in the Explorer or Object Browser and choose **Execute** from the contextual menu.

The servlet and JSP engine will be started and the JSP page will be displayed in the web browser.

Specifying query parameters

You can specify query parameters to pass in the request from the browser to the server. In the page's property sheet, edit the `Request Parameters` property in the **Execution** tab. Use

standard query string notation (in the form of *`param1=value1&param2=value2`*).

Restarting the server

When you execute a JSP page more than once, the server is not restarted. Instead, a new version of the page is loaded into the server. This saves startup time for the page. However, any other classes used by the page are not reloaded. So you may need to restart the server if, for example, you change a bean used by your JSP.

To restart the server:

◊ Right-click on the JSP's node in the Explorer or Object Browser and choose **Execute (restart server)** from the contextual menu.

Configuring JSP Execution Types

Sometimes you may want to change settings of the server used for execution. These are all accessible in the property sheet for JSP Execution.

To access the JSP property sheet for execution:

1. Select the JSP's node in the Explorer.
2. Choose **View | Properties** from the main menu to open the Properties window.
3. In the Properties window, select the **Execution** tab, click the **Executor** property, and select the **...** button that appears.

The Executor property editor will appear. In the left pane, `JSP Execution` will be selected. In the right pane are properties, including the following:

- `Execute unregistered servlets` – when false, links to servlets in the form *`URL/servlet/package.ClassName`* are disallowed.
- `MIME types` – specifies mapping of file extensions to MIME types.
- `Port` – port on which the server runs. Make sure that the port number does not conflict with any other service running on your machine.
- `Welcome files` – files to look for when there is a request to a directory.

The other properties (including those displayed under the **Expert** tab) are related to the Java process in which the server runs, and are the same as those for External Execution.

An important property of the server is the root directory from which the JSPs, HTML pages and other files are served. This directory is always the root directory of the filesystem on which your JSP page resides.

Setting the web browser

By default, when you run a JSP, the page is displayed in the IDE's built-in web browser. However, you can also set the IDE to display JSP pages in an external web browser.

To change the web browser used for JSP pages:

1. Choose **Tools | Global Options** from the main menu.
2. Select the `JSP & Servlets` node and change the value of the `Web browser` property to `External`.
3. Select the `External browser` property and press the **...** button that appears to open its custom property editor.
4. In the **Process** field of the custom property editor, enter the path to the executable for the browser of your choice, either manually or by pressing the adjoining **...** button to use the file chooser.
5. Once the path is entered, close the property editor by pressing **OK**.

You can also access the Executor property editor in the Project Settings window by selecting the `Execution Types | JSP Execution | JSP Execution` node. All of the properties are shown in the right pane of the window.

Note: You can only compile and execute JSP pages located on a local filesystem. You can not compile and execute pages packed in JAR or WAR files.

Chapter 8

Organizing work into projects

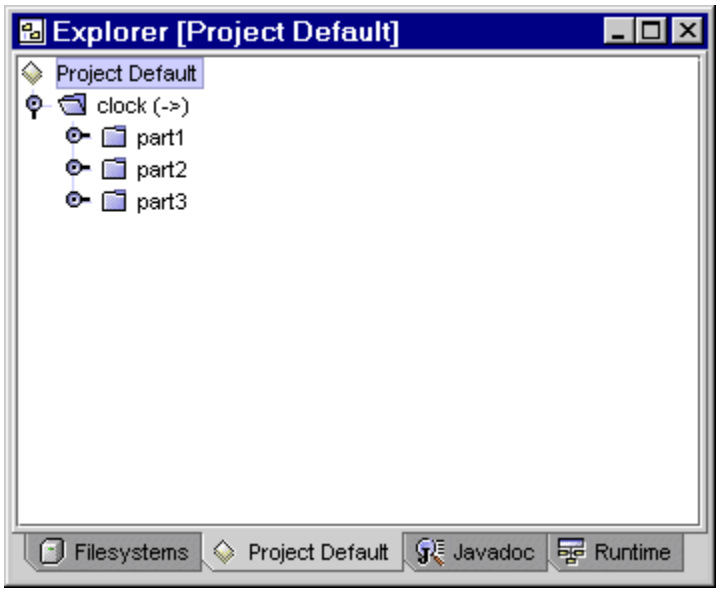
Forte for Java, Community Edition enables you to organize your files into projects. This enables you to save different IDE configurations for each project as well as to conveniently compile, execute, and debug by project.

What is a "project" in Forte for Java?

A project in Forte for Java physically consists of its own subdirectory stored in the `Projects` folder of your installation directory, which contain the files that hold the information about the state of the IDE when you were last working in that project. When you switch to a different project, the information from that project's files are loaded and the IDE's state (meaning open files and windows, active workspace, and configuration of Project Settings) is accordingly changed.

Dividing your work into projects provides the following advantages:

- You can compile the whole project without having to specify which files to include.
- If you have set a main class for the project, you do not have to seek out this class when you want to run or debug the project.
- You can view all files added to the current project under the **Project** tab in the Explorer and work with them as you would under the **Filesystems** tab, without having other files interfere with the view.



- You can configure the IDE uniquely for each project, affecting such things as the mounted file systems, compiler types, executors, and Editor configuration. When you switch projects, the IDE is automatically reconfigured to match the Project Settings you have set for that project.
- The IDE's visual state is stored for each project when you exit the project or the IDE. When you open a project, the same files and windows will be open and in the same positions as they were when you last worked in the project.

If you have any projects that you have created with other Java IDEs, you can import them into Forte for Java.

Creating projects

When you work in the IDE, you always work in a project, whether it is the default project or one that you have created. The current project you are working in is indicated by the the name of the root node under the **Projects** tab in the Explorer.

The default project

When you start the IDE for the first time, a default project, representing the default state of the IDE, is already in place. If you do not create any new projects, this default project will hold any changes made to the IDE at the project level (such as changes in workspaces and Project Settings) when you end your session with the IDE.

Creating new projects

When you create a project, you give a name to the current IDE state, and then that information is stored in the IDE. Current Project Settings, such as settings for compiling, execution, and debugging, are saved for the new project. The only Project Settings not saved are the Filesystems Settings. Any directories mounted under **Filesystems** (except for the `Default system` directory, the default Javadoc directory, and `Timerbean.jar`) are removed from the new project.

To create a new project:

1. Choose **New Project** from the **Projects** menu.
2. When prompted, type a name for your new project and click **OK**.
3. You will be prompted whether to use the file system or file systems currently mounted in Filesystems or whether to remove them and mount a new one. If you click **New**, the `Development` directory and any other directories or JAR files you have mounted in Filesystems will be removed, and a file chooser will appear for you to select a new file system to mount. If you click **Old**, all file systems currently mounted in Filesystems will remain there.
4. You will be prompted whether to save the current project. If you click **Yes**, any changes you have made to your current project will be saved before it is closed and the new project is opened.
5. The file chooser will then appear prompting you to pick a directory to mount under the **Filesystems** tab.

   The packages and files in any directories that you mount will then be available for inclusion in your project. See "Adding packages and files to a project" below.

   Once the new project is created, the name of the root node under the **Projects** tab will be updated with the name of the new project.

Working with projects

When you use the IDE, you always work within a single project. You can work within the IDE's default project or you can create projects of your own. For the default project and each

project you have created, you can add specific files and file systems, designate a main class, and configure its Project Settings. You can easily switch between the projects you are working in.

Adding packages and files to a project

You can add packages and files (including source, HTML, and plain text files) to a project so that they are viewable on the **Projects** tab in the Explorer. This enables you to view and access all of the files crucial for your project from one place. In addition, it enables you to compile all project files at the same time.

This does not affect which files or file systems are mounted under **Filesystems** in the Explorer or which files are available for editing. The files mounted in **Filesystems** are always available for editing, whether you have added them to the project or not.

To add an existing file or package to a project:

1. Select the **Project** tab in the Explorer, right-click on the root `Project` node, and choose **Add Existing...** from the contextual menu.
2. In the Select Objects dialog box that appears, browse to the directory, package, or file you want to add to the project and select **OK**.

You can also add existing files or packages to a project by selecting files or packages under the **Filesystems** tab in the Explorer and then choosing **Tools | Add to Project** from the contextual menu or **Tools | Tools | Add to Project** from the main menu.

When you switch to the tab of your project in the Explorer, the files you have added will appear under the main project node. You can work with the files from either the project's tab or the **Filesystems** tab in the Explorer.

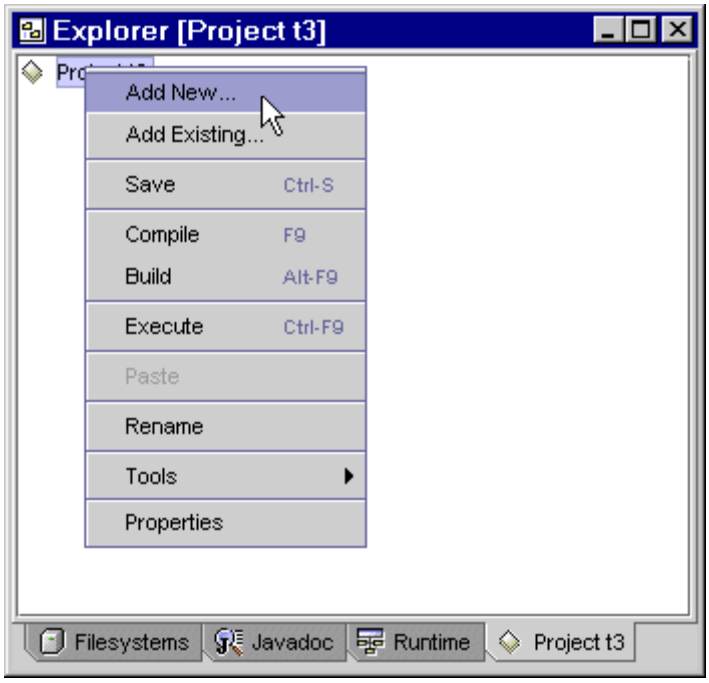
Important: The package or file that you have added will appear under the **Project** tab as a link. If you delete its node, the files in its hierarchy will be deleted from the project but will remain in Filesystems. However, if you delete any of other the nodes in the hierarchy, those files will be completely deleted from Filesystems.

When you are working in the Explorer with the **Project** tab selected, you can create new objects by using the contextual menus of the Explorer nodes. When you create new files this way, the objects are added to the selected file system (as represented on the **Filesystems** tab) and to the project (as represented on the **Project** tab).

To add a new file to the root of a project:

1. Select the **Project** tab in the Explorer, right-click on the root `Project` node, and choose

Add New... from the contextual menu.



2. In the New From Template wizard that appears, expand the appropriate template category node (for example, **Classes**) and then select one of the templates. In the right panel of the dialog, a description of that template will appear.
3. Press **Next** to move to the Target Location page of the wizard.
4. In the **Name** field, type the name you want to give the object. Do not type the extension for the file type. It will be added automatically.
5. In the same page of the wizard, choose a package for the template. You can do this either by typing a package name in the **Package** field or by selecting one of the packages shown in the tree view just below it.
6. Press **Finish** to exit the wizard and have the template created.

You can also create new files in existing packages, much as you would under the **Filesystems** tab in the Explorer.

To add a new file to a package in the project:

1. Select the **Project** tab in the Explorer, right-click on a package in its hierarchy, and choose **New From Template...** from the contextual menu and navigate the submenus to choose a template. The second page of the New From Template wizard will appear.
2. Fill in the wizard page as described in steps 4 and 5 of the previous procedure.

Note: Any file that you add to a directory that is already in the current project will also be added to that project, even if do not have the **Project** tab in the Explorer selected when you are adding the file.

Setting the main class in the project

When you have created a project and added some objects to it, you can then set the main class for it. This is the class that will be executed when you execute the project as whole (CTRL + SHIFT + F9). This saves you from having to find and select the main class of your application in order to run your program.

To set the main class for a project:

◊ Right-click on the class's node in the Explorer (under either the **Filesystems** or **Projects** node) or the Object Browser and choose **Tools | Set As Project Main Class**; or

◊ On the **Project** menu in the main window, choose **Set Main Class**. A dialog box will open in which you can select the main class. If there is a class file currently selected under the **Filesystems** tab in the Explorer, the text field will be filled with the name of that class.

Accessing and working with files in projects

No matter which project you are working in, you can work on any file listed in the **Filesystems** tab of the Explorer, whether you consider it to be part of your project or not. If you have added files to the project ("Adding packages and files to a project" on page 156), you can access these files under the **Projects** tab and work with them just as you would under **Filesystems**.

Mounting different file systems for different projects

You can mount different file systems for different projects.

When you create a new project, all **Filesystems** are removed from the **Filesystems** tab in the Explorer. You are then prompted to mount file systems that you need for that project. See "Mounting file systems" on page 75.

Likewise, you can configure Filesystems Settings (such as whether to display or hide each mounted file system) individually for each project. See "Filesystems Settings" on page 178

Compiling, building, executing, and debugging projects

You can simultaneously compile or build all files you have added to a project and execute or debug a project (according to the main class you have set). You can access these commands from the **Project** menu or with the keyboard shortcuts given below.

- **Compile Project** (SHIFT+F9) – compiles all classes in the projects which are not compiled or have been changed. It recursively goes through all packages added to the project (like Compile All does).

- **Build Project** – recompiles all classes in the project, recursively going through all packages

added to the project (like Build All).

- **Execute Project** (CTRL + SHIFT + F9) – executes the project by executing the main class. If necessary, the project is first compiled.

- **Debug Project** (CTRL + SHIFT + F5) – executes the project by executing the main class. If necessary, the project is first compiled.

You can also set the default compiler, executor, or debugger for the whole project in the Projects Settings window on the property sheet for `Java Sources`. See "Adjusting Project Settings" below.

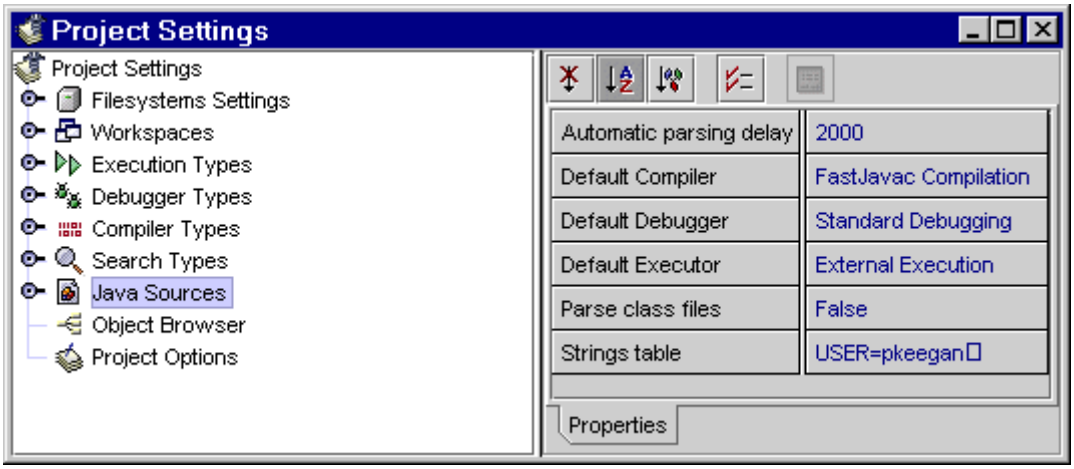
Adjusting Project Settings

You can save a unique configuration of Project Settings for each project you have created. When you open a project, the IDE is automatically reconfigured to the state it was in when you last saved that project. When you close the IDE, any changes you have made to Project Settings during the session will be saved to the current project.

Important: Whenever you change Project Settings, it only affects the current project. However, if you subsequently create a new project, it will start with the default Project Settings.

To adjust any of the Project Settings for a project:

1. Open up the Project Settings window by choosing **Project | Settings...** from the main menu.
2. Select the node you want to change in the left pane and then adjust the properties you wish to change in the right pane.



3. Save the project by selecting **Project | Save Project** from the main menu.

   If you do not save the project, it will be saved automatically when you exit the IDE.

For more information on the various configurations available for projects, see "Project Settings" on page 177.

Saving projects

When you exit the IDE, any changes made to the current project are automatically saved. When you switch to a different project, you are prompted whether or not to save any changes made to the project you are closing. You can also save changes to a project without leaving it.

To save a project without closing it:

◊ Choose **Save Project** from the **Project** menu.

Note: When you save a project, you do not actually save the files you have associated with the project. Only the project information such as the IDE state, Project Settings, and a record of which files you have associated with the project is affected. To save the files themselves, select **File | Save All** from the main menu. This will save all files that have been modified, whether they have been modified or not.

Switching between projects

If you have more than one project saved, you can easily switch between them by opening a different project.

To switch projects:

◊ On the **Projects** menu in the **Main Window** choose **Open Project**; or

◊ Expand the `Project` node in the Global Options window, right-click the node for the project you want to open and choose **Open** from the contextual menu.

A dialog box will appear enabling you to choose the project you want to switch to. Once you have selected a project, the IDE will prompt you to save changes to the previous project before it opens the next one.

The name on the root node under the **Project** tab in the Explorer will then change to indicate the project you have just switched to, and the IDE state will change accordingly. These changes include the current Project Settings and the files shown under the **Project** tab.

Importing projects

If you have projects created in other Java IDEs that you would like to work on in Forte for Java, you can import those projects and have them mounted under the **Filesystems** tab in the Explorer.

To import a project:

1. Choose **Import Project** from the **Projects** menu.
2. Using the file chooser that appears, search your computer for the project. You can specify Microsoft J++, JBuilder, or Visual Cafe projects in the **Files of Type** field.
3. Once you have selected a project file, click **Open** to have the file mounted.

Chapter 9

Integrating a CVS version control system

In team development projects where more than one developer may work on the same code, there are a number of challenges, such as:

- knowing who changed what, and when;
- knowing why a given change was made; and
- when a new bug surfaces, finding out what change caused it

Forte for Java helps you meet these challenges by making it possible to use the popular (and free) Concurrent Versions System (CVS) version control software from within the IDE. Using CVS also can be useful for you to keep a history of revisions you make to your applications as they are in development.

How the CVS support works

Forte for Java's CVS support is effectively an object wrapper which integrates CVS into the Forte for Java environment. Files accessed through it have all the properties of files on the local file system (access dates, attributes, and so on) – along with two additional properties: file state and locker status. The file state appears next to the file's name on its Explorer node. The locker status can be viewed in the output generated when you choose the **Log** command. For each action performed on your files, your version control system is called, and the information it returns on the command line is processed by the module.

Note: Every operation is called in an asynchronous manner, so that you can do your work and do not have to wait for commands to complete. Therefore, the file state and locker status of a file will not necessarily be visible until the CVS module needs this information (for example, when an attempt is made to modify a file) and requests it from your version control system.

Many of the calls to your version control system (for example, to retrieve status information) happen automatically, behind the scenes. However, there are some actions that must be performed manually, such as committing changes. When you have mounted a CVS file system, a new node appears under the **Filesystems** tab of the Explorer with the label `CVS *your CVS*`. To dispatch commands to the version control system, right-click on the file or directory you want to apply the command to, choose **CVS** from the contextual menu, and then choose the command from the submenu.

Note: Since the CVS module is designed to work across many platforms and operating systems, some configuration is necessary to get the systems interacting properly. CVS is called from within the context of a command shell.

Using the CVS module

Before you begin

If you have not already done so, download and install CVS on to your system. The GNU web site at http://www.gnu.org/ is one place where you can obtain a copy of CVS as well as more information.

You will then need to configure the CVS module according to your needs and set up the working directory it will use (see the sample project below).

Important: If you have not worked much with CVS, we strongly recommend that you take the time to familiarize yourself with the documentation for it. The documentation is available from http://www.gnu.org/manual/cvs-1.9/html_chapter/cvs_toc.html.

Setting up the repository and working directory

You should have the CVS system installed on your system before you start using it from within Forte for Java. If you are not sure if it is installed, start a command shell and type `cvs`. If it is installed, a help message will appear. Ensure that you have the `PATH` environment variable to CVS properly set.

If you have not already done so, you will then need to set up a repository and a working directory for files that you work on in CVS.

To set up the CVS repository:

◊ Outside of the IDE, create a folder or directory, such as

```
/home/Repository
```

Preferably, the repository should be put in a place where each developer working on the project can access it but where it is unlikely that accidental deletions could occur.

To set up the CVS working directory:

◊ Outside of the IDE, create a folder or directory on your local drive, such as

```
/home/bob/MyDocuments/work
```

Creating a CVS project

Once you have CVS set up on your machine and you have set a repository and working directory, you can begin creating CVS projects.

To create a CVS project:

1. In the Explorer window of the IDE, select the **Filesystems** tab, right-click on the `Filesystems` node, and choose **New CVS** from the contextual menu.
2. In the Customizer dialog box that appears, configure the module to communicate with CVS by following these steps:
   a. Choose the destination of the CVS repository in the **CVS Connection Type** panel. If it is located on your local drive, select `local`. Otherwise select `server` or `pserver` to access the repository on a server.
   b. Move to the **CVS Settings** panel. If your CVS Connection Type is server or pserver, enter your server name in the **Server** field.
   c. Next to the **Repository** field, click **Browse** to open up a file chooser. In the file chooser, navigate to the CVS repository you have created for your project. This is where the versions of your files used by the version control system will be stored.
   d. If you have any sub-directories in your CVS repository, you can treat them as CVS

modules. Enter the name of the module you want to work with. If you want to work with the whole directory, then leave the field blank.

e. Click **Browse**, navigate to directory you have set up as your working directory, and then press **OK**.

Note: If you are working on Windows 95/98, you have to use a Unix shell as an interpreter of CVS commands (since the command shell used on Windows NT/2000 does not work properly on Windows 95/98). To use a Unix shell you have to install one first and then choose **Use Unix Command Shell** and enter the path to the interpreter. The recommended Unix shell is `sh.exe` or `bash.exe` from Cygwin (http://sourceware.cygnus.com/cygwin/).

3 If the CVS repository has not yet been initialized, right-click on the CVS file system under the **Filesystems** tab in the Explorer and select **CVS | Init** from the contextual menu.

4 Right-click again on the CVS file system's node and choose **CVS | Check Out** from the contextual menu.

You can now create packages and files in the IDE and add them to your CVS project.

To add a package and its files to your CVS project:

1 Right-click on the root node of your CVS working directory and choose **New Package** from the contextual menu to create a new package. Name the package `cvsproject`.

2 Right-click on the newly-created `cvsproject` node and choose **New from Template | Classes | Class** from the hierarchy of contextual menus. Name the class and click **Finish**. The class will appear with its file status (`[Local]`) in brackets after its name on the node.

3 Repeat the previous step to create other classes.

4 Add the package and classes to the CVS repository by right-clicking on the `cvsproject` package node and choosing **CVS | Import** from the contextual menu.

5 CVS works best when imported files have been checked out from the repository, so you should then do the following

a. Right-click on the `cvsproject` node and choose **Delete** from the contextual menu. All of the files in `cvsproject` will be deleted from the working directory.

b. Right-click on `cvsproject` and choose **CVS | Check Out** from the contextual menu to put the files back in the working directory on the regular working branch.

c. Right-click on `cvsproject` and choose **CVS | Refresh Recursively** from the contextual menu to display the checked out files and their status. (The files should appear with the status `[Up-to-date]`).

6 Begin editing one of the files.

You can now work with your files as you wish. Try adding some code to one of the classes you haved added, and save your changes. If you then right-click on the class and choose the **CVS | Refresh** command from the contextual menu, the results will show that the files have changed. Then right-click on the class's node and choose **CVS | Commit** from the

contextual menu. Your file's status will change from `Locally Modified` to `Up-To-Date`.

Note: If you use CVS keywords such as `$Log: $` in a file, you will be prompted with a dialog box informing you that your file has been updated outside of the IDE when you choose the **CVS | Commit** command on that file.

Tip: At any time you can use the **CVS | Refresh** command on any file or directory to refresh the status information. Use the **CVS | Refresh Recursively** command to start asynchronous status retrieval of an entire directory tree. This can improve the speed of browsing files on large projects (otherwise Refresh commands are dispatched to the version control system as you open folders). It is advisable to refresh recursively after major changes to your version control directory structure. The refresh process can be stopped at any time if desired.

CVS commands

The following are the various CVS commands available under the CVS submenu in the contextual menu for CVS file systems. In parentheses are the command line CVS equivalents where they differ in name.

- **Refresh** (`cvs status -l`) – refreshes the file status of the file or files in the selected package.
- **Refresh Recursively** (`cvs status -l`) – recursively refreshes the file status of all files in the selected package.
- **Init** – initializes the CVS repository. This is necessary only after creation of a new repository directory.
- **Update** – updates the file with changes that other developers have made to the source in the CVS repository and, if necessary, merges local changes.
- **Commit** – use this to incorporate your changes to the CVS repository. When you choose this command, you will be prompted by the Reason dialog box, where you can enter a brief explanation of changes you have made.
- **Add** – use this command to add new files (with status `Local`) to the CVS repository. After this command you have to use **Commit** to actually incorporate the file to the CVS repository.
- **Remove** – use this to eliminate the file from the CVS repository. The file's node will remain in the working directory, but the file status will change to `Locally Removed`. The removal is not final until you run **Commit**.

- **Import** – this command imports the whole directory structure to the CVS repository.
- **Check Out** – this command makes a local copy of the CVS repository in your working directory.
- **Status** – display a window with the file's status information.
- **Log** – display a window with the file's log information.
- **Check Out Revision** (`cvs checkout -r revision_number`) – use this command to extract any desired revision from the CVS repository into your working directory. A dialog box with available revisions is displayed.
- **Update Revision** (`cvs update -r revision_number`) – use this command to update your file in the working directory with any desired revision from the CVS repository. A dialog box with available revisions is displayed.
- **Commit To Branch** (`cvs commit -r branch_number`) – use this command to commit your file to a specific branch. A dialog box with available branches is displayed.
- **Merge With Branch** (`cvs update -j <branch number>`)– use this command to merge your changes with a desired branch. A dialog box with available branches is displayed.
- **Remove Sticky Tag** (`cvs update -A`) – use this command to remove any sticky tag which is associated with your file after update of a given revision. This also updates your file with the head revision.
- **Add Tag** (`cvs tag -b -r RevisionNumber` or `cvs rtag -b -r RevisionNumber`) – use this command to add a symbolic tag to your file. You may specify a desired revision number.
- **View Branches** – display a window with a revision tree. You may select any two revision numbers and compare their differences. Select the first revision to compare with the left mouse button and select the second with the right mouse button. This command parses output from the `cvs log` command
- **Diff** – display the differences between your working file and the repository head revision (meaning the most recent version in the repository). This command parses output from the `cvs diff` command

Chapter 10

Searching and Creating Javadoc Documentation

Forte for Java, Community Edition enables you to easily browse standard API documentation and other Javadoc files from within the IDE and create Javadoc documentation for your own classes.

Searching and browsing Javadoc

When you are editing files in the IDE, you can quickly obtain information about the Java classes and methods you need by bringing up the Javadoc Search Tool and then viewing the selected documentation in the web browser. You can also select a class in a file in an Editor and press Alt+F1 to view API documentation on that class without having to enter the class name in the Search Tool.

Preparing the Explorer's Javadoc tab

Before you can do Javadoc searches within the IDE, directories (or JAR or ZIP files) with standard API documentation need to be in the search path.

There should be two directories already mounted under the **Javadoc** tab in the Explorer:

- The Java API `docs` directory (if it is installed under the Java 2 installation directory). It will appear there with a name such as `/usr/jdk1.2.2/docs/api`.
- The `Javadoc` directory (which can be found in the root of the IDE's installation directory).

Note: By default, Javadoc search directories are set to hidden, meaning they will not be displayed under the **Filesystems** tab in the Explorer. If you would like to have a Javadoc directory displayed under **Filesystems**, choose **Project | Settings...** from the main menu, expand the `Filesystems Settings` node in the Project Settings window, select the Javadoc directory's node, and set its `Hidden` property to `False`.

If neither directory is present or you would like to add a new directory with Javadoc documentation, you can add them by following this procedure.

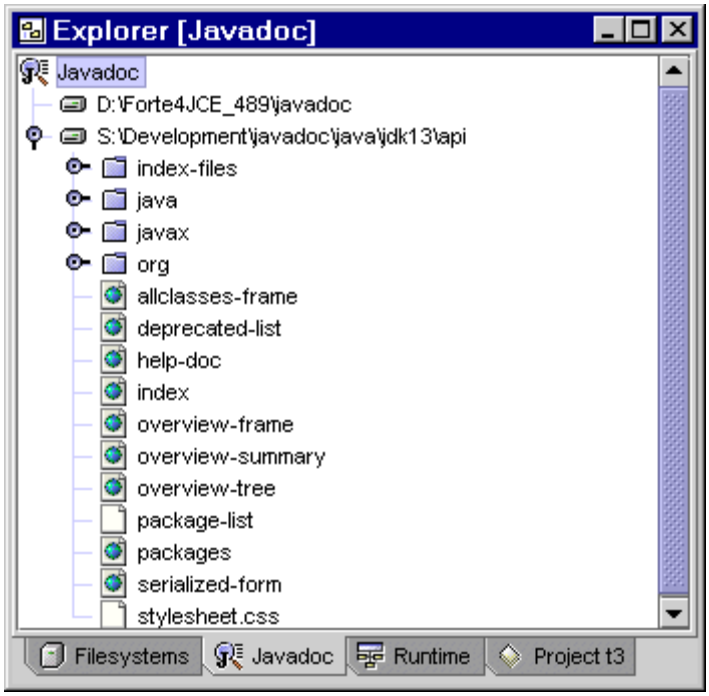
To add a Javadoc documentation directory to the Javadoc tab in the Explorer:

1. In the Explorer, select the **Javadoc** tab, right-click on the `Javadoc` node, and choose **Add Directory...** from the contextual menu. (If you are adding a JAR or ZIP file with Javadoc documentation, choose **Add JAR...** for JAR or ZIP files).

2. In the Mount Directory dialog box (or Mount JAR Archive dialog box when mounting a JAR or ZIP archive) that appears, choose the directory with the Javadoc documentation.

When selecting the Javadoc documentation directory, you should look at all original documentation distributed with the SDK and be sure that it has an `index-files` directory

or `index-all.html` file in the top level or the second level of its hierarchy.



Browsing documentation for the selected class or element

When you are working in the Editor, you can bring up the Javadoc documentation by placing the insertion point in the word you want to search and pressing Alt+F1. If the documentation for that class or element is mounted in the Explorer under the **Javadoc** tab, the web browser will open and display its Javadoc documentation.

Searching in Javadoc directories

Assuming the Javadoc directory is installed, you can use the Javadoc Search Tool to find and view documentation from within the IDE.

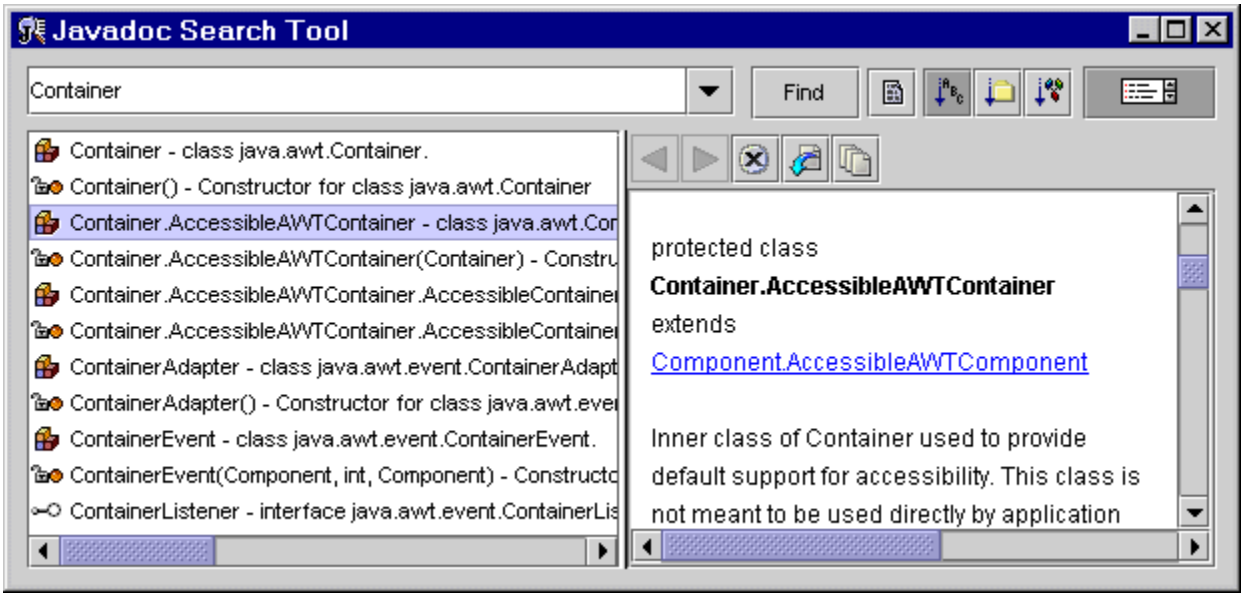
To search on Javadoc documentation, either:

1. Bring up the Javadoc Search Tool dialog box by pressing CTRL+F1 or choosing **Help | Javadoc Index Search** from the Main Window.
2. Use the combo box in the dialog box to type or select the search string, and then press **Find** or press ENTER on your keyboard.

or:

◊ Select the string or click on the word you want to search in the Editor window, and then

press CTRL+F1 or choose **Javadoc Index Search** from the **Help** menu.



The `Javadoc Search Tool` will then begin searching for your string.

Search dialog box

The top part of the Javadoc Search Tool dialog box has a combo box for selecting or typing the search string and a **Find** button for starting the search (after the search is started, it changes to **Stop**).

Next to the **Find** button is the **Show Source** button, which you can click to open the source file in the Editor, assuming the source is mounted in Filesystems.

The three icons to the right of the **Show Source** icon serve as toggles for sorting the pages found. Selecting the first sorts the items alphabetically, selecting the second sorts by package and alphabetically, and the third sorts by element (exceptions, classes, constructors, and so on).

To the right of the sort icons is the **HTML Viewer** toggle button, which you can click to turn on or turn off the right pane.

Below the combo box and icons are two panes. The first pane displays the list of pages found in the search, and the second pane shows the page selected in the first pane. The left pane can be expanded to cover the whole dialog box area by deselecting the icon to the right of the sort icons. The size of the panes can be changed by dragging the divider in the center.

If you double-click on a page in the left panel, it opens in a separate web browser window.

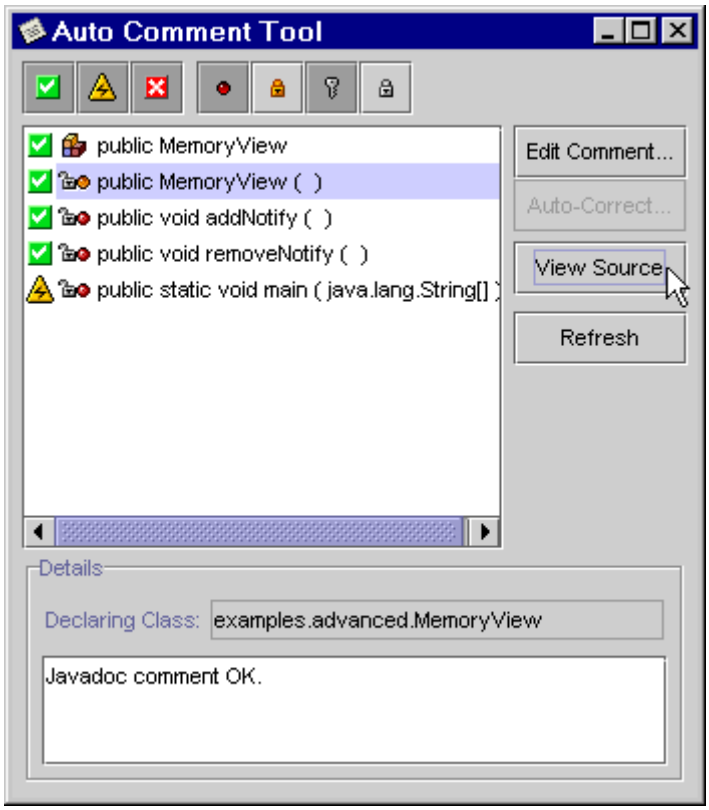
Using Javadoc Auto Comment

The Auto Comment utility enables you to comment your source code automatically and view all parts of your source code (methods, constructors, inner classes, variables, and so on) and document them individually.

Auto Comment Tool dialog box

You can open the Auto Comment Tool dialog box by right-clicking on a source file, or one of its elements in the Explorer or Object Browser and choosing **Tools | Auto Comment**. The Auto Comment Tool dialog box consists of two panes, seven filter icons and four buttons.

Tip: You can use the Auto Comment Tool for multiple files simultaneously. To do so, hold down the SHIFT key while selecting the different files and then right-click and choose **Tools | Auto Comment** from the contextual menu. However, be sure to not select any item that does not have **Tools | Auto Comment** on its contextual menu (such as an image or a text file).



The first pane lists constructors, variables and other elements for comment. Below that, the `Details` pane contains the **Declaring Class** field, which shows the class that selected element belongs to, and information about missing or invalid comments on the item selected in the first pane.

The first three filter icons, respectively, enable you to view class elements

1. with completely valid comments
2. with partially valid comments (something could be missing or invalid in the Javadoc tags)
3. without comment

The next four icons are filters which enable you to view public, package private, protected and private elements of class source.

The dialog box also includes the following buttons:

- **Edit Comment...** – brings up the Javadoc Comment dialog box, where you can edit the comment. See the following section for more information.
- **Auto-Correct...** – opens the Javadoc Comment window and automatically corrects any errors in the tags.
- **View Source** – opens the source file which you are currently documenting and moves the insertion point to the element selected in the Javadoc Comment dialog box.
- **Refresh** – refreshes the display in the first pane to reflect any changes made in the source file (for example, through the Editor).

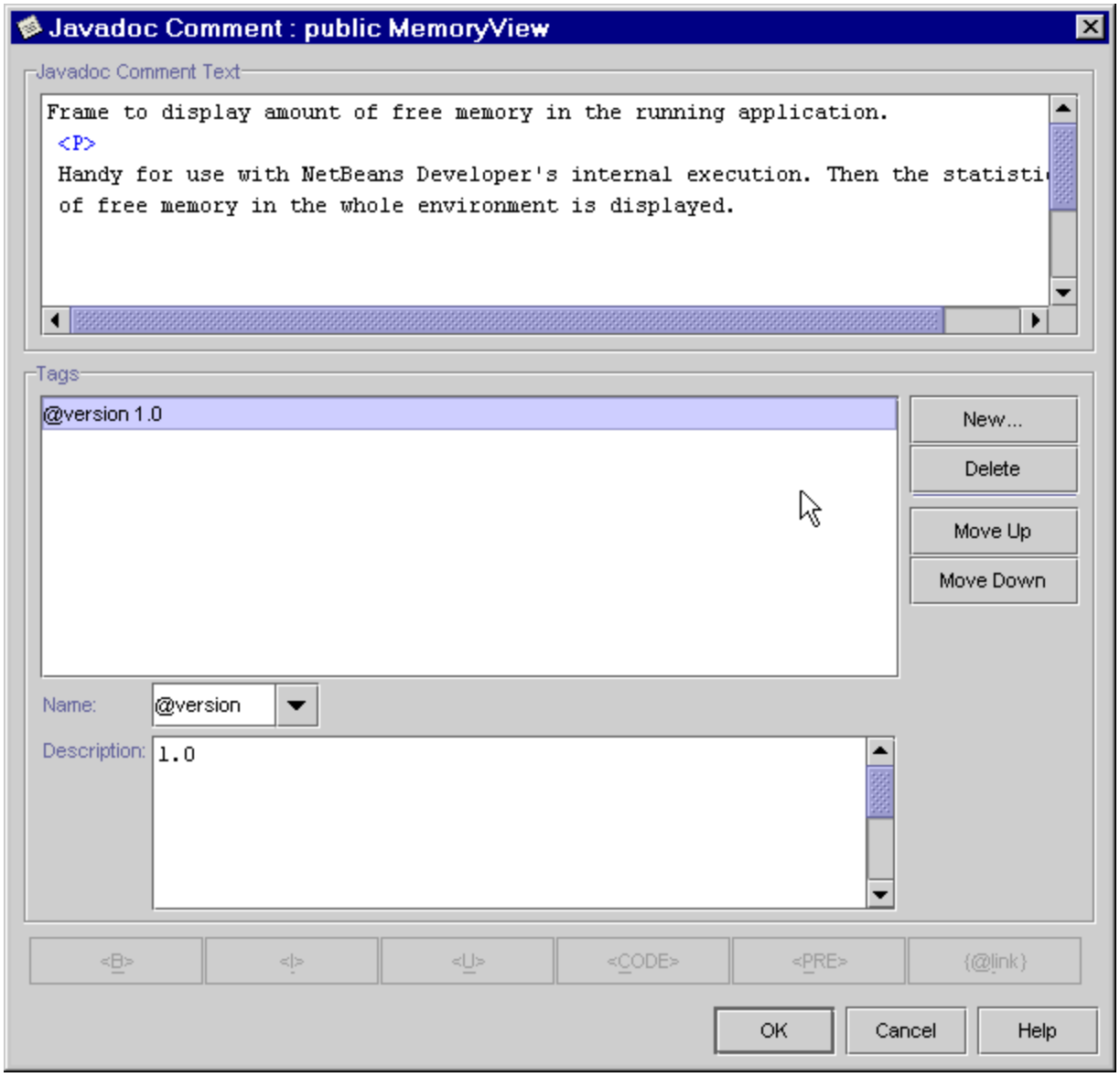
Javadoc Comment dialog box

The Javadoc Comment dialog box lets you add comments to class elements separately.

To open the Javadoc Comment dialog box:

◊ Press **Edit Comment...** in the AutoComment Tool dialog box; or

◊ Open the property sheet for the element you want to document, click on the value of the `Javadoc Comment` property, and then click on the **...** button that appears.



The Javadoc Comment dialog box has two main panes. The **Javadoc Comment Text** pane displays comment text. The **Tags** pane displays all Javadoc tags used in the comment. When you select a Javadoc tag, a combo box appears along with the **Description** text field. (If the `@see` tag is selected, two read-only text areas named `Class` and `Type` also appear.)

In the Description pane, you can change the description of the Javadoc tag selected in the combo box. In the combo box you can change the selected Javadoc tag to another Javadoc tag.

There are four buttons on the right side of the **Tags** pane:

- **New** – brings up the New Tag dialog box, which enables you to choose from predefined

Javadoc tags or type another tag. It is element sensitive, so only appropriate tags on class members are shown.

- **Delete** – deletes the selected tag in the **Tags** combo box.
- **Move Up** and **Move Down** – change the order of the Javadoc tags in the comment by moving the selected tag up or down.

On the bottom of the Javadoc Comment dialog box are buttons with predefined HTML tags and the Javadoc `{@link}` tag, which you can add to your comment and have displayed in the **Javadoc Comment Text** pane.

Press **OK** to accept all changes or **Cancel** to reject all changes.

Note: For detailed information about Javadoc tags, visit Sun's Javadoc web page, at http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html#javadoctags.

Generating Javadoc documentation

You can have documentation for entire classes and packages generated automatically.

To generate Javadoc documentation:

1. Select the packages and/or source files for which you would like to create documentation in the Explorer (under the **Filesystems** tab) or in the Object Browser.
2. Right-click on the object's node and choose **Tools | Generate Javadoc** from the contextual menu. A dialog box will appear asking you to name the directory where you want the Javadoc documentation generated.

   Once you enter a directory, another dialog box will appear and ask you whether to open the generated documentation in the internal web browser.

   By default, the generated Javadoc documentation will be stored in the directory `javadoc` in the IDE's home directory. Since this is mounted under the Javadoc tab by default, you can later search its contents using the Javadoc Index Search. See "Searching in Javadoc directories" on page 170.

Javadoc properties

You can set options for both Javadoc and the standard doclet to modify the way the documentation is created.

Note: Doclets are programs you can use within Javadoc to customize the Javadoc output.

Forte for Java uses the Javadoc standard doclet.

To view or change Javadoc and standard doclet options:

1. Choose **Tools | Global Options...** from the main menu to open the Global Options window.
2. Select the `Internal Javadoc` or `Standard Doclet` node.

   The property sheet for the selected node will appear in the right pane of the window.

For more information on Javadoc properties, go to the Javadoc page on Sun's web site: http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html.

Changing the directory for generated Javadoc documentation

To change the directory for generated documentation:

1. Choose **Tools | Global Options...** from the main menu and select the `Documentation | Standard Doclet` node in the Global Options window.
2. Select the `Destination` property and change the directory manually or click the **...** button that appears to change it using the File Chooser.

Chapter 11

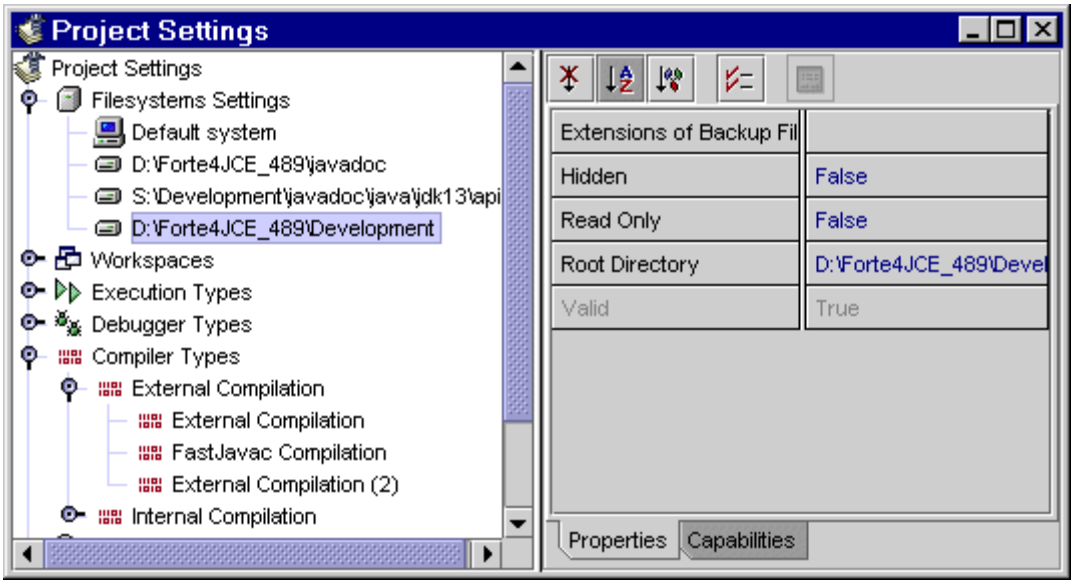
Configuring and customizing the IDE

This chapter provides an overview of the various parts of the Forte for Java, Community Edition IDE, details advanced functions, and tells you how you can customize it to your own specifications.

Project Settings

This window holds all of the IDE system settings that can be set for a project. Whenever you change one of these settings, the change only affects work in the current project. If you switch to a different project, the IDE automatically switches to the Project Settings for that project. See "What is a "project" in Forte for Java?" on page 153 and "Organizing work into

projects" on page 153 for more information on projects.



Project Settings include Filesystems settings, internal and external compiler configurations, debugger types, window settings, external browser or applet viewer configuration, and more. The left pane provides a tree view of the Project Settings items, and the right pane displays the property sheet, where you can view and edit the various options for the selected item. As in the Explorer, contextual menu items such as **Customize Bean**, **Copy**, and **Paste** are available for many of the Project Settings nodes.

To open the Project Settings window, choose **Project | Settings...** from the main menu.

For information on IDE settings that can not be set separately for each project, see "Global Options" on page 182.

Changes made to Project Settings are automatically saved to the current project when you exit the IDE. To save changes without closing the IDE, choose **Project | Save Project** from the main menu.

Below are descriptions of each item in Project Settings.

Filesystems Settings

`Filesystems Settings` lists all mounted file systems. You can customize the appearance and behavior of file systems that are currently mounted – local directories as well as JAR and ZIP archives. If a new file system is added to the Explorer, it automatically appears as one of the items in the `Filesystems Settings` tree. You can use the contextual menu on the `Filesystems Settings` to add new file systems and re-order current file systems. See "Filesystems" on page 75 for more information.

On the property sheet, you can hide file systems or make them read-only. Under the **Capabilities** tab on the property sheet, you can disable the IDE from performing certain tasks on the file system (searching for sources to compile, searching for classes to execute, searching for classes and sources to debug and searching for HTML pages with documentation). These properties are of particular interest if you want to mount the Java 2 SDK itself and debug it. You can set the `compile` property on the SDK file system to `False` to prevent the IDE from trying to compile it.

Tip: `Capabilities` settings apply to entire file systems. However, if you would like to keep individual sources in a file system from being accidentally compiled, executed, or debugged, you can change the source's `Compiler`, `Executor`, or `Debugger` property (under the **Execution** tab of its property sheet) to `(do not ...)` to accomplish this.

Workspaces

Under `Workspaces` are nodes for every workspace, and under these are nodes for each window inside of the workspace.

Since the configuration of each workspace is updated whenever you open or close a window within that workspace, the window nodes also change to reflect the windows that are currently part of that workspace. In addition, the properties of the given windows reflect the size and positions you have given the windows in the various workspaces.

The workspace and window nodes each have their own property sheets. On the property sheets for the workspaces, you can set the name for the workspace and the toolbar configuration used for that workspace. For the window nodes, you can set the size and position of the window with the `Bounds` property, and you can also set the `Display Name`.

You can also configure workspaces and set up new ones in the Explorer. See "Customizing workspaces" on page 222 for more information.

Execution Types

Here you can set specific configurations for each "executor" (the actual system command that is called when you execute an object, which includes the path to Java, the working directory, and any arguments). You can use the default executors of each category (external, internal, applet, or JSP execution), modify the default executors, or write new ones. You can then associate any of your classes to one of these executors if the default executor assigned for that class does not suit your needs.

There is a node for each category of executor (such as `External Execution`, `Internal Execution`, `Applet Execution`, and `JSP Execution`) and under these nodes are listed the executors themselves. See "Adding and modifying service types" on page 204 for information on creating custom executors.

You can also access the custom property editor for executors from the property sheet of individual classes. See "Editing service types from the Filesystems tab in the Explorer" on page 207.

For Applet Execution, you can modify the `External Viewer` property. It is not possible to modify the Thread (internal) executors.

Debugger Types

Just as with Execution Types, you can make custom configurations for debugger types (the actual system command called when you debug an object, including the path to Java, the working directory, and any arguments). You can use the default debugger types of each category (for example, standard debugging or JPDA debugging), modify the default debugger types, or write new ones. You can then associate any of your classes to one of these debugger types if the default debugger type assigned for that class does not suit your needs.

You can also access the custom property editor for debugger types from the property sheet of individual classes. See "Setting the debugger" on page 68 and "Adding and modifying service types" on page 204 for more information.

Compiler Types

Here you can set specific configurations for each type of compiler type (the actual system command that is called when you compile an object, which includes the path to Java, the working directory, and any arguments). In Forte for Java, you can use the default compiler types of each category (for example, internal compilation and external compilation), modify the default compiler types, or write new ones. You can then associate any of your classes to one of these compiler types if the default compiler type assigned for that class does not suit your needs.

You can also access the custom property editor for compiler types from the property sheet of individual classes. See "Switching compilers by class" on page 52 and "Adding and modifying service types" on page 204 for more information.

Search Types

Here it is possible to configure and create sets of reusable search criteria ("search types"), which you can later reference when you conduct searches of files in the mounted file systems. (You can also configure criteria from the Customize Criteria dialog box, which appears when you select **Tools | Search Filesystems** from the main menu.)

Under the `Search Types` node are three category nodes (`Object Name`, `Full Text`,

and `Date`). Under each of these nodes are nodes containing the actual search criteria. You can modify the existing search criteria by modifying the properties on their property sheets.

See "Searching for files in Filesystems" on page 83 for more information on conducting searches. The properties for the Search Types node are detailed in "Search Types reference" on page 260.

Java Sources

The Java Sources settings enable you to set the default compiler, execution, and debugger types, set the source parser delay, and set a strings table.

When you change the default for a service type (compiler type, executor, or debugger type), all classes or templates using the default of that service type are affected.

Note: Once you explicitly change a class or template's service type, the IDE will never again recognize the class as using the default of that service type, even if you switch that service type for the class back to the one that is the IDE default. Therefore, if you change a service type for a class and then change it back to the service type that is set as the default, the class's service type will not be affected if you change the default service type.

The strings table is a list of substitution keys for templates. When the key appears in a template (marked by two underscores on each side of the substitution key in the template), the string assigned to the key appears in place of the key in any objects created from that template.

For example, the `USER` substitution key is assigned your user name. If a template has the key `__USER__` placed anywhere in the text, your user name will appear in place of `__USER__` in any object created from this template.

The `Java Sources` node also contains the `Source Synchronization` subnode where you can enable or disable the source synchronization feature as well as set its return mode.

Object Browser

The sole property for this node enables you to set custom package filters for use in the Object Browser. (You can also do this from the Object Browser window.) See "Guide to the Object Browser" on page 86 for information on using the Object Browser.

Project Options

The sole property for this node enables you determine whether files newly created from

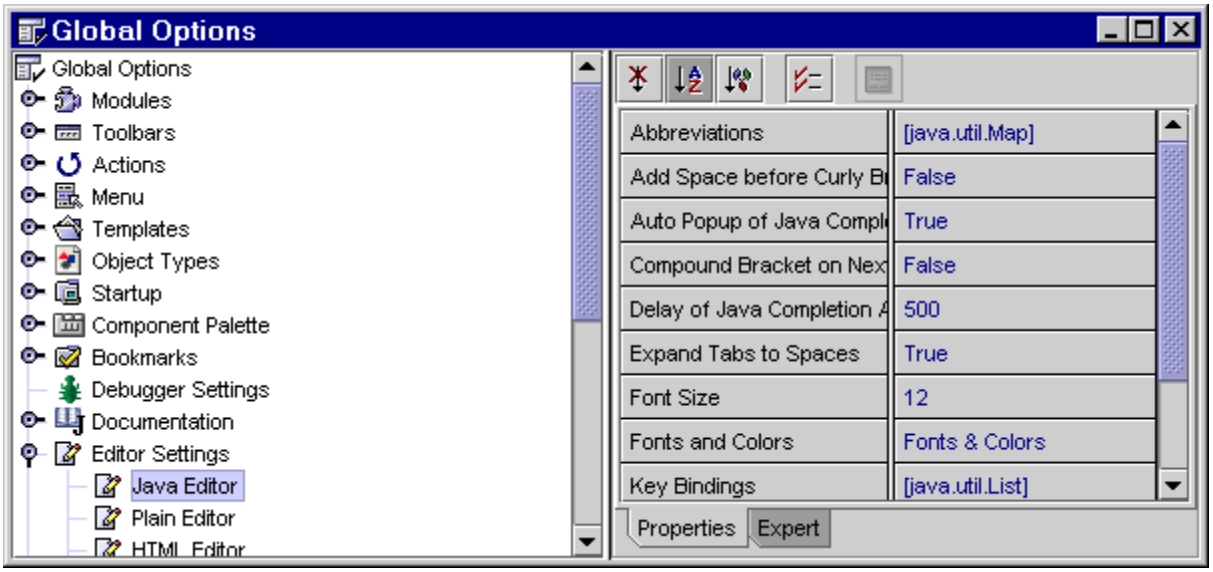
template are automatically added to the current project or not.

If set to `Ask`, each time you create a class from template, you will be prompted whether or not to add the class to the current project. If set to `Never`, you will not be prompted, and the class will not be added to the current project when you create it. If set to `Always`, you will not be prompted, and the class will automatically be added to the current project when you create it.

See "Organizing work into projects" on page 153 for more information on working with projects.

Global Options

The Global Options govern the overall look, content, and performance of the IDE, regardless of the project you are working on. The left pane of the Global Options window provides a tree view of the Global Options items, and the right pane displays the property sheet, where you can view and edit the various options for the selected item. As in the Explorer, contextual menu items such as **Customize Bean**, **Copy**, and **Paste** are available for many of the Global Options nodes.



To open the Global Options window, choose **Tools | Global Options...** from the main menu.

For information on IDE settings that can be configured by project, see "Project Settings" on page 177.

Below are descriptions of each item in Global Options.

Modules

This node lists all of the modules currently installed in the IDE. The property sheet for each module shows the module's name, version number, and whether it is enabled.

Toolbars

Under `Toolbars`, you will find various customization settings for the buttons of the toolbars on the Main Window and all of their elements. You can add and remove commands for each toolbar. You can also change the order of the toolbar items and insert separators.

Actions

The action pool under `Actions` in the Global Options window stores all actions (commands) that are available in the IDE, most of which (but not all) are already represented in the menus and toolbars. Since you cannot delete actions from the action pool, this gives you the freedom to remove whatever items you want from the menus and toolbars without permanently removing them from the IDE. You can add actions to the main menu and toolbars by copying them from the action pool and pasting them to folders beneath `Menu` or `Toolbars` in the same window.

Menu

Under `Menu`, you will find various customization settings for the menus and menu items in the Main Window. Under the `Menu` node, you will find settings for each menu on the Main Window menu bar, including `File`, `Edit`, `View`, `Projects`, `Build`, `Debug`, `Tools`, `Window`, and `Help`. Under these are subnodes for the menu items. You can add, remove, and customize commands for each menu and menu item. You can also change the order of the menu items and insert separators.

Templates

Here you will find the standard set of templates, in several categories: `AWT Forms`, `Beans`, `Classes`, `Sample Forms`, `JSP and Servlet`, `Swing Forms` and `Other`. Each category contains several templates for creating new objects. See "Using templates" on page 200. You can also create your own categories and templates – see "Creating your own templates" on page 201 – and modify existing templates (including their property sheets).

Object Types

Under this node, you can change the order of the object types and customize their properties. Each object type recognizes specific types of files (usually by their extension) and groups them with similar files. This affects where new objects that can fall under multiple object types are placed in the IDE (and ultimately how they are handled) since they are automatically dropped into the first category that they match with when they are used by the IDE.

Important: In some cases, the order is crucial: for example, form objects must come before Java source objects, which must come before sourceless class objects. Only experts should modify these settings.

The items that appear on contextual menus for nodes in various IDE windows are determined by the object type. You can determine which contextual menu items appear in the contextual menu for each type of object. See "Changing commands in contextual menus" on page 219.

For some object types, such as text, image, HTML, and class objects, you can set the extensions that the loader recognizes (under `File Extensions` on the object type's property sheet).

Startup

Under this node, you can insert classes that the IDE will run at startup. Internal execution is usually required for startup classes.

Component Palette

In the Component Palette section of the Global Options window, you can add, remove, move, change the order of, or edit components. From a component's contextual menu, you can move it up or down, cut, copy, delete, or set properties. Some items have an `Is Container` property, which, when set to `True`, means that other components can be added to that component in the Form Editor. See "Is Container property" on page 108 for more information.

Bookmarks

This node holds all of the bookmarks (URLs) that appear under **Help | Bookmarks** in the main menu. Using the contextual menus, you can add, remove, move, change the order of, or edit bookmarks, as well as create sub-folders to organize them.

Debugger Settings

The Debugger Settings property sheet contains settings enabling you to choose what action should be performed when tracing into a method without source, whether to compile classes before debugging, which workspace to use when debugging, and so on. See "Debugging Java classes" on page 60 for a complete guide to debugging in the IDE and "Search Types reference" on page 260 for a description of each setting.

Documentation

The Documentation property sheet contains options for the IDE's Javadoc generation functionality, with nodes for Internal Javadoc (options for the IDE's Javadoc module) and Standard Doclet. See "Generating Javadoc documentation" on page 175 and "Documentation settings reference" on page 263 for more information.

Editor Settings

The Editor Settings property sheet gives you broad control over the behavior of the Editor and appearance of files displayed in the Editor window. You can set key bindings, make abbreviation lists, use the Color Editor to customize foreground and background colors for different types of text, choose caret (insertion point) type, and so on. Under the `Editor Options` node are four subnodes, representing types of files that can be edited in the Editor: `HTML Editor`, `JSP Editor`, `Plain Editor`, and `Java Editor`. These subnodes each have separate property sheets, enabling you to make distinct customizations for each type of editor. For a description of all of these settings, see "Editor Settings reference" on page 265.

Execution Settings

The Execution Settings property sheet gives you the option of clearing the Output Window when executing, having separate Output Window tabs for each execution run, and automatically compiling sources before execution. You can also choose the workspace used when running applications.

Form Objects

The Form Objects settings affect the appearance of the Form Editor window during design time, code generation settings, and so on. See "Form Objects reference" on page 269.

HTML Browser

The HTML Browser settings enable you to set the default background color and the default fixed and proportional fonts used by the internal web browser when displaying pages. See "HTML Browser reference" on page 271 for specific details.

HTTP Server

The HTTP Server property sheet controls the built-in HTTP (Hypertext Transfer Protocol or "web") server. Among other things, you can establish the host, determine access to the server, and change the port. See "HTML Browser reference" on page 271.

JSP & Servlets

On this node are properties which enable you to choose the web browser on which to display your running JSP. See "Setting the web browser" on page 152.

Java Elements

These settings control the display names of element nodes in the Explorer. You can enter a combination of plain text and substitution codes. For example, if you enter `class {n}` in the `Classes` property, a class called `MyClass` will be represented as `class MyClass` in the Explorer hierarchy. See "Java Elements reference" on page 272 for a complete list of the substitution codes and a description of advanced substitution features.

Open File Server

The property sheet for this node enables you to configure the server that listens to open requests if you use the IDE as an external viewer or editor or double-click Java sources from the desktop of your machine. See "Opening files from outside of the IDE" on page 224.

Output Window

These settings control the colors, fonts, and tab size used in the Output Window. For the color settings, you can select the property and then either click the drop-down arrow to select a color from the combo box or click the **...** button to use the Color Editor. See "Color Editor" on page 288.

Print Settings

These settings control the appearance of files printed from the Editor. From the `Print Settings` node, you can modify header, footer, line wrapping, and line ascent correction properties for printouts of all Editor files. On the property sheets for each of the three subnodes (`Java Editor`, `Plain Editor`, and `HTML Editor`), you can set line numbering and specify detailed background and foreground coloring for different types of text within the files.

Property Sheet

Here you can set the appearance and behavior of property sheets. See "Property Sheet reference" on page 277.

System Settings

On the System Settings property sheet, you can choose whether or not to have the IDE prompt you to confirm deletions, and choose whether to show tips on startup. There are also settings for proxy port and proxy server, which you can set if you need to use a proxy server for HTTP or FTP connections with the web browser and Update Center. If you want to use a proxy, you must also set the `Use Proxy` property to `True`. See "JSP and Servlets reference" on page 272.

Update Center

The Update Center settings enable you to choose whether to be prompted when you start up the IDE and whether to check the Update Center for new and updated modules. You can also set the frequency (ranging from every startup to once per month) that you are prompted to check for updates.

See "Adding modules with the Update Center" on page 194 for information on using the Update Center.

Window management

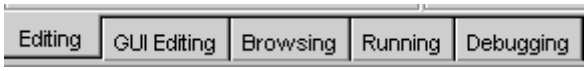
Forte for Java has the following features which, when combined, give you a great deal of flexibility in managing windows on your desktop:

- **Workspaces**, meaning window sets, each geared toward a given task (for example, editing, GUI editing, browsing, running, or debugging). When you switch workspaces, the current set of windows closes and a another set opens.
- **Multi-tab windows**, which hold multiple open files at once and enable you to flip between files by clicking on a tab.
- **Docking** of windows, which enables you to change the windows in which window contents are displayed (for example, you can move a source file from a multi-tab Editor window to its own window or to a different already existing container window, such as the Output Window).
- **Cloning** of windows, which enables you to have the same file open in two different windows, making it easier to work on two different parts of the file at the same time. Cloning is available for the Editor, web browser, and help windows.

Workspaces

As you progress through the development cycle (write, compile, debug, execute, edit, and so on) when working on any significant project, the screen can become cluttered with windows. While all of these windows are necessary for development, they are generally not all needed simultaneously.

Forte for Java workspaces enable you to efficiently manage a large number of windows and to group these windows into useful, logical sets. You can flip between workspaces by simply clicking the different workspace tabs on the Main Window.



Standard workspaces

The default workspace configuration is a standard and logical grouping of the most commonly used windows:

- **Editing** – Explorer and Properties window; if any files are open, the Editor window as well. After compilation of files, the Output Window opens if there are any compilation errors.
- **GUI Editing** – Explorer; if any visual forms are open, the Editor window, Form Editor window, and Component Inspector as well. After compilation of files, the Output Window opens if there are any compilation errors.

- **Browsing** – Object Browser and Properties window, and then the Editor window if any files are opened.
- **Running** – Execution View, Output Window.
- **Debugging** – Debugger Window, Output Window, and Editor (if any files are open).

Automatic workspace switching

By default, the IDE automatically switches to the GUI Editing Workspace when you open a visual form, to the Running Workspace when you execute a program, and to the Debugging Workspace when you start a debugging session. You can change this automatic switching by opening the Global Options window and adjusting the `Workspace` property for `Form Objects` (GUI Editing Workspace by default), `Execution Settings` (Running Workspace), and `Debugger Settings` (Debugging Workspace).

Using workspaces

Being in a given workspace does not constrain what windows you can have open. You can open (for example, from the **View** menu) or close any window without having to change the workspace.

Upon exiting the IDE, the current state of each workspace is saved. This information includes open windows and their sizes and positions. When you next launch Forte for Java, Community Edition, your workspaces will appear exactly as you left them.

Any given window can be open on any workspace—and can be open on more than one workspace simultaneously.

For example, to view the Editor window on both the Editing and Running workspaces:

1. While in the Editing workspace, double-click on a Java object displayed in the Explorer.

   The Editor window will open with that source.

2. Flip to the Running Workspace by clicking on the **Running** tab in the Main Window.

   The first Editor window will no longer be visible, and you will see either the default set of Running Workspace windows or those you left open when last using this workspace.

3. Choose **Window | Windows | Editing | Editor** from the main menu.

   (The **Windows** submenu reveals submenus for each workspace, such as the **Editing** workspace. Each workspace submenu shows a list of currently open windows in that workspace. When you choose a window from the submenu of a different workspace than the one you are currently using, that window opens up in your current workspace as well.)

The Editor will then be visible on both the Editing and Running workspaces.

Workspaces are completely customizable: you may add, delete, and rename the available workspaces. You can also customize which toolbars appear in each workspace. See "Customizing workspaces" on page 222 for further details.

Multi-tab windows

A multi-tab window presents multiple files as separate tabs in a single frame, enabling you to quickly and easily flip between these files, either by clicking on the tabs or by using the ALT+LEFT and ALT+RIGHT keyboard shortcuts.

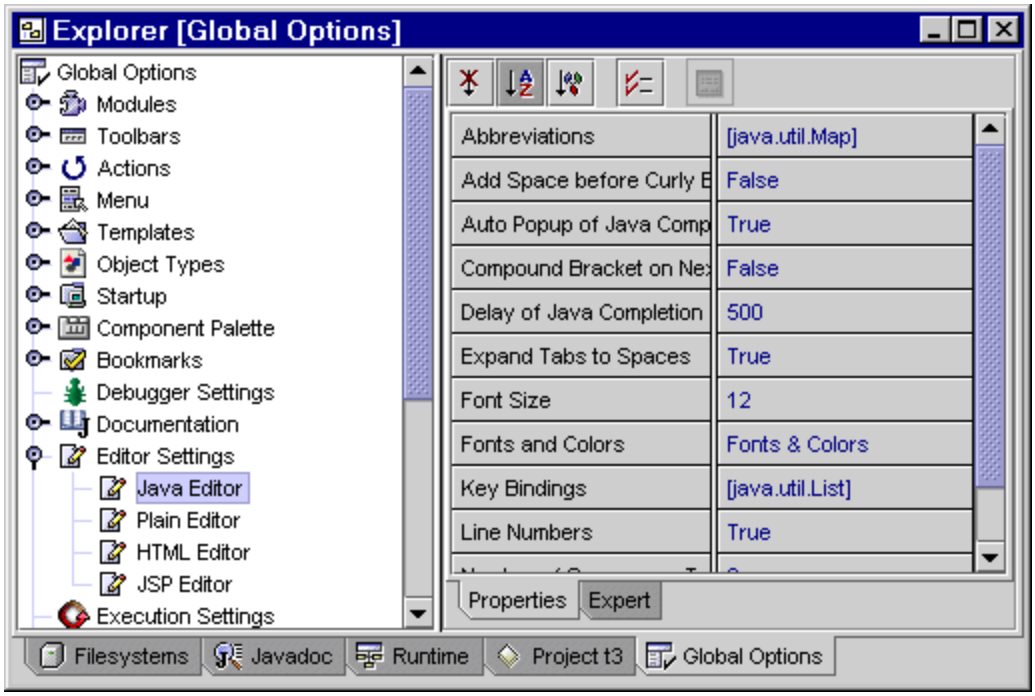
Each tab has a contextual menu of commands (**Save**, **Clone View**, **Close**, and **Dock Into...**), accessible by right-clicking on the tab itself. See below for more information on cloning, docking, and undocking windows.

Note: Multi-tab windows with only one file or component look like single windows – that is they do not have a tab. Thus commands that appear on contextual menus for tabs must be accessed elsewhere (such as from the Main Window) when in a single window. Once a second file is opened in or docked to the window, tabs for both items appear.

Undocking and docking windows

While the multi-tab window enables quick and efficient access to more than one file or view, only one of those files is visible at any one time. At times it is useful to look at different files or views simultaneously, side by side. Forte for Java makes this possible with its **docking** feature, which enables you to dock (move objects into multi-tab windows) and undock (move object views to single-view windows).

The Explorer with the Global Options window docked into it



Undocking

Any object open in a multi-tab window (with at least two objects open) can be undocked and presented in its own stand-alone, independent window. In this way you can simultaneously view separate sources, side by side. Undocked windows may also be docked back to the "parent" multi-tab window (See "Docking" below.)

To undock the active tab of the active window:

◊ Choose **Undock Window** from the **Window** menu on the Main Window; or

◊ Choose **Dock into...| New Single Frame** from either

- the **Window** menu on the Main Window; or
- the tab's contextual menu (available by right-clicking on the tab).

The active object will be removed from its original window and will appear in a new window. This new window may be repositioned, resized, moved to another workspace, and even closed – all independently of the parent window. You can position the windows side-by-side to view sections of source simultaneously or copy and paste code between windows.

Docking

Docking windows enables you to reduce clutter on your desktop without having to close any objects and to view objects in a window that is more convenient for you.

You can dock any window tab or single window into any named container window such as:

- the Output Window
- the Debugger Window
- the Editor window
- the Explorer window

or

- other windows such as the Debugger Window, Object Browser, and Form Editor window when they are already open

or

- a new unnamed multi-tab or single-frame window

Almost any single-frame window or frame from a multi-tab window can be docked into one of the windows above. These include single-frame windows such as the Properties window, Object Browser, internal web browser, Execution View, Javadoc Index Search, User's Guide, Auto Comment, Project Settings, Global Options, Search Results, and Form Editor windows. Individual frames from multi-tab windows such as the Explorer, Editor, Debugger Window, and Output Window can also be docked. Frames from multi-tab windows can also be docked into new unnamed single-frame windows.

To dock an undocked window into a different window:

1. Click on the window to be docked to activate it.
2. Choose **Dock Into...** from the **Window** menu on the Main Window and then one of the windows given in the submenu.

   The undocked window will close, and the object will open as a new tab in a multi-tab window.

To dock a tab in a multi-tab window into a different window:

1. Choose **Dock Into...** from either
   - the **Window** menu on the Main Window (making sure that the undocked window is the active window); or
   - the tab's contextual menu (available by right-clicking on the tab).
2. Choose one of the window choices in the **Dock Into...** submenu.

Cloning windows

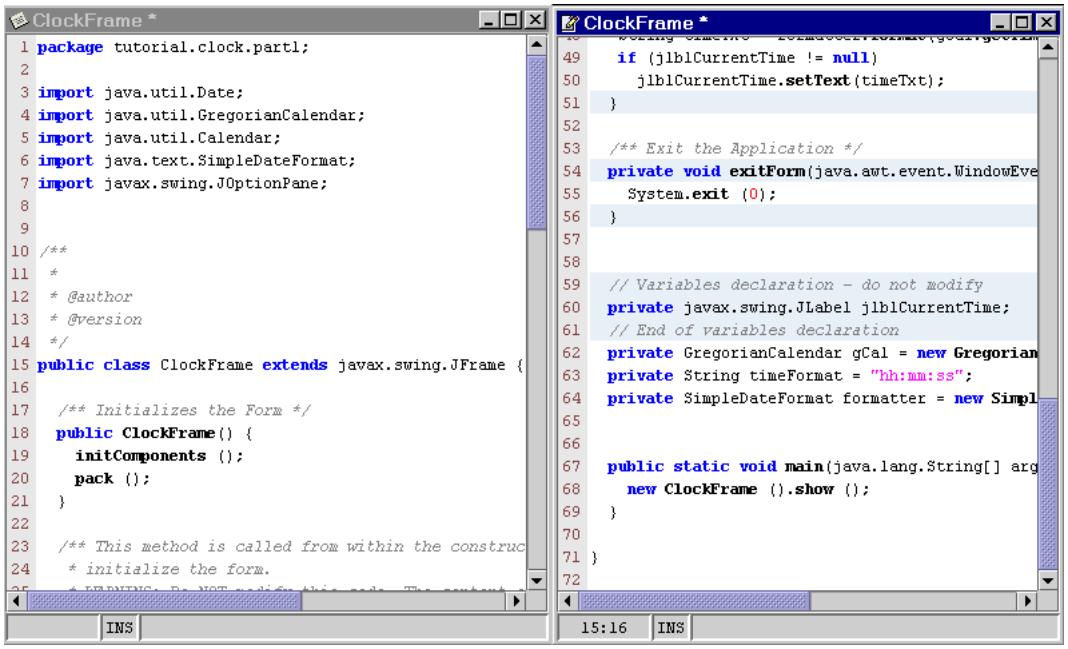
It is often useful to view separate sections of the same file simultaneously. Forte for Java has a **clone** function to accomplish this. You can clone the view of any Editor file, the internal web browser, and the JavaHelp browser (for the User's Guide and context help).

To clone a window:

1. Make sure the file to be viewed in the cloned window is on the active tab.
2. Choose **Clone View** from either
   - the **Window** menu of the Main Window; or
   - the tab's contextual menu.

   A new tab will open in the same multi-tab window, displaying the same content.
3. If you so desire, undock this new view of the file or dock it into a different window.

An Editor window with a clone



For more information on docking, see "Undocking and docking windows" on page 190.

Note: You can not use the clone command for the Explorer window. However, you can open up a new Explorer window showing a limited part of the hierarchy by right-clicking on a node in the Explorer and selecting **Explore from Here** from the contextual menu.

Redisplaying windows obscured by windows from other

applications

When switching back and forth between Forte for Java and other applications by clicking on the windows, much of the Forte for Java IDE may remain obscured by windows from the previously active application.

To redisplay the entire Forte for Java IDE:

1. Minimize the Forte for Java Main Window.
2. Redisplay Forte for Java by clicking the icon for Forte for Java's Main Window.

   All of the open windows will then be displayed over the windows of any other open applications.

If you select the icon for an individual Forte for Java window, that window will be made active, but it does not ensure that all of the other open IDE windows will be displayed over windows of other applications.

Modules in Forte for Java

Forte for Java's modular architecture means that all parts – even those central to the functionality of the IDE, like the Editor, Debugger, and Form Editor – are in module form. There are also base modules for things such as the HTTP server, the web browser, and the creation of JavaBeans components. In addition, you can create your own modules or add third-party modules.

Updates to existing modules and new modules can be easily obtained and added to the IDE.

Managing modules

All modules installed in the IDE have their own node under `Modules` in the Global Options window. Their property sheets contain the `Enabled` property which you can use to activate or deactivate the module. The other five properties (including two on the **Expert** tab) are read-only and give basic information about the module.

Adding modules with the Update Center

When you use the Update Center, the IDE leads you through a wizard which compares the IDE's modules with the currently available modules and enables you to choose which ones to update or add. The Update Center is designed to update the IDE by installing modules

directly from the Forte for Java web site, but it can also work with ZIP files previously downloaded to your computer.

To update the IDE directly from the Forte for Java web site:

1. Choose **Help | Update Center...** from the main menu to bring up the Update Center wizard.
2. On the first page of the wizard, choose the **Check the web for available updates** radio button. If you are checking for updates for the Community Edition, you do not need to enter a registration number. If you are looking for updates for non-Community Edition modules (such as those available to members of the Forte for Java Early Access Program), enter the appropriate registration number.

   If you need to set a proxy or the proxy host or proxy port, press **Configure Proxy...** to bring up the Proxy Configuration dialog box. See "Adjusting proxy and firewall settings" on page 196 for more information.
3. Click **Next** to move to the second page.
4. The IDE will access the Forte for Java web site where the latest versions of all modules are available. Depending on your system/network configuration, this may mean your machine will dial and connect to your ISP. Once connected to the Forte for Java Auto Update web page, it will determine which modules are missing or not current in your IDE.

   Wait for the IDE to finish its check of the web site. When the check is finished, the status line at the bottom of the wizard page will report whether any new or updated modules are available. If it reports "Done. Updates Available", click **Next** to move to the third page.
5. The third page of the wizard displays a list of modules that you do not have the most recent version (or any version) of. In the **Available Modules** list, select the ones you would like to install and click **Add>** (or **Add All>** if you want to update all of the modules).

   If any modules have interdependencies, for example new module A requires the updated module B, the required additional module updates are automatically selected.

Note: After you select a module or modules to be downloaded and click **Add>** or **Add All>**, you will be presented with the license agreement, which you will need to accept before any modules can be downloaded.

The **Details** panel provides information about the module selected in the **Available Modules**, including the version numbers of the module installed and its update, the size of the module in kilobytes, and a description of its function. The **Total Size** field indicates the total size of all of the selected modules.

You can get a more detailed description of a module by selecting it in the **Available Modules** combo box and then pressing the **Home Page** button to link to the module's description on the web site.

6. Click **Next** when you have made your choices.

The IDE will then download the modules that you have selected and check for their digital signatures to determine if the module has arrived intact and whether it has a certificate. You will be prompted if the download is unsuccessful, the downloaded file is corrupted, or the file contains an unrecognized or untrusted certificate. See "Authenticating Update Center downloads" on page 198 for more information on certificates.

7 When the download is finished, click **Next** to move to the fifth panel.

8 Click **Finish** to restart the IDE and begin working with the newly installed modules.

Automatic Update Check dialog box

When the Automatic Update Check dialog box appears to prompt you whether to check the Update Center, you can configure the Update Center settings in the same dialog box. If you check the **Show Autocheck Configuration** checkbox, you will be presented with several options:

- The **Period of Automatic Update Check** combo box enables you to set the frequency that the IDE will prompt you to check the Update Center.
- If the **Show dialog before automatic update check** check box is checked, in the future you will be prompted with the Automatic Update Check dialog box before the IDE checks the Update Center.
- If the **Show check results even if no update available** check box is checked, the Automatic Update Check dialog box will reappear after the check, even if no new or updated modules are found in the check.
- In the **Update Center Registration Number** field, you can type your Update Center registration number, so that you do not have to type it every time that you access the Update Center.
- If you press the **Configure Proxy...** button, you can adjust your proxy settings. See "Adjusting proxy and firewall settings" below.

You can also adjust these settings in the Global Options window. See "Global Options" on page 182.

Adjusting proxy and firewall settings

If you are using Forte for Java from behind a firewall, or you access the internet through a proxy server, you will need to configure the IDE to use your proxy settings.

Configuring Forte for Java to use an HTTP proxy

You can configure Forte for Java's HTTP Proxy settings in Forte for Java in the Update

Center wizard, in the Global Options window, or on the command line.

To specify the HTTP proxy settings in the Update Center wizard:

1. Choose **Help | Update Center...** from the main menu to start the Update Center wizard.
2. In the first page of the wizard, press the **Configure Proxy...** button to bring up the Proxy Configuration dialog box.
3. Click the **Use Proxy** checkbox to select it, so that the IDE will use a proxy. (If you do not know them, ask your system or network administrator.)
4. Fill in the **Proxy Host** and **Proxy Port** fields, and press **OK**.

To specify the HTTP proxy settings in the Global Options window:

1. Choose **Tools | Global Options...** from the main menu to open the Global Options window and select the `System Settings` node. The property sheet for this node contains the settings `Proxy Host` and `Proxy Port`.
2. Fill in these settings for `Proxy Host` and `Proxy Port`.
3. Set the `Use Proxy` property to `True`.

To specify the HTTP proxy settings on the command line on a Windows machine:

◊ Go to the `bin/forte4j.cfg` file in the installation directory, open it in a text editor, and add the switch

```
-J-Dhttp.proxyHost=HOST -J-Dhttp.proxyPort=NUM
```

replacing ***HOST*** and ***NUM*** with your local settings.

To specify the HTTP proxy settings on the command line on a Solaris or Linux machine:

◊ Go to the `bin/forte4j.sh` file in the installation directory (or the `bin / forte4j_multiuser.sh` for multi-user installations), open it in a text editor, and add the command line switch

```
-Dhttp.proxyHost=HOST -Dhttp.proxyPort=NUM
```

replacing ***HOST*** and ***NUM*** with your local settings.

Configuring Forte for Java to use a SOCKS proxy:

To use Forte for Java with a SOCKS Proxy on a Windows machine:

◊ Go to the `bin/forte4j.cfg` file in the installation directory, open it in a text editor, and add the switch

```
-J-DsocksProxyHost=SOCKS-SERVER -DsocksProxyPort=1080
```

replacing *SOCKS-SERVER* with the hostname of your SOCKS proxy. Contact your system administrator if you do not know this.

To use Forte for Java with a SOCKS Proxy on Unix:

◊ Go to the `bin/forte4j.sh` file in the installation directory (or the `bin/forte4j_multiuser.sh` for multi-user installations), open it in a text editor, and add the command line switch

```
-DsocksProxyHost=SOCKS-SERVER -DsocksProxyPort=1080
```

replacing SOCKS-SERVER with the hostname of your SOCKS proxy. Contact your system administrator if you do not know this.

If you are still having problems with configuring your system to get the Update Center to work, you can check http://www.netbeans.com/ffj/community/autoupdate_info.html#proxy for more information or see below for instructions on updating the IDE manually.

Authenticating Update Center downloads

When you download modules through the Update Center, the IDE authenticates the module by checking its digital signature to make sure that it matches with a certificate in the IDE's key store.

The certificate for all Forte for Java modules originating from Sun Microsystems is in the IDE's key store, so all modules from Sun that arrive uncorrupted are trusted. If the module originates from another source, it will either have a different certificate or no certificate at all.

If a module is not trusted, you can choose **Trust For Module** (to install that module only), **Trust For Now** (to install all modules with that certificate in the current download), or **Trust Forever** (to enter the certificate into the IDE's key store so that all current and future modules with that certificate will be automatically trusted).

If you reject a certificate, the module will not be installed. Likewise, you can choose **Reject For Now** (to reject all modules with that certificate in the current download), or **Reject Forever** (to remove the certificate from the IDE's key store). After rejecting a certificate, you can later accept it.

Updating modules manually

If you are unable to configure your IDE to use the Update Center with your firewall or proxy, you can download a ZIP archive containing the available modules from the Forte for Java web site and update the IDE from the extracted files.

To update the IDE from files downloaded manually:

1. Download the most recent ZIP archive from the Forte For Java web site containing the modules.
2. Extract the files from the downloaded archive to a folder on your system.
3. Choose **Help | Update Center...** from the main menu to bring up the Update Center wizard.
4. On the first page of the wizard, select the **Install manually downloaded modules** radio button and click **Next** to move to the second page.
5. Click **Add...** to bring up the file chooser, navigate to the directory where you placed the files, select the directory, and click **OK**.
6. Click **Next** to go to the third page of the wizard and proceed according to steps 4 through 6 of the previous procedure for updating through the web site.

You can also add modules manually by right-clicking on the `Modules` node and choosing **New Module from File...**. The Open dialog box will appear enabling you to browse for the JAR file containing the module. If you select a valid module and click **OK**, the module will be installed into the IDE.

Uninstalling and reinstalling modules

If you rarely use certain modules, you may want to uninstall them from the IDE. Doing so can decrease IDE startup time and possible lower memory usage.

To uninstall a module:

1. Choose **Tools | Global Options...** from the main menu.
2. Expand the `Modules` node and select the module you want to delete.
3. On the property sheet, select the `Enable` property and set it to `False`.

   Any features provided by the module will be removed from the IDE.

The next time you start Forte for Java, Community Edition, the JAR file for that module will not be loaded into the IDE.

Should you wish to later reinstall the module, you can repeat the same procedure, but switch the `Enable` property to `True`.

Module help

If available, documentation for Forte for Java extension modules (any available modules that are not part of the base Community Edition) can be found under **Help | Documentation | *ModuleName***. The help file is part of the module itself and becomes available once the module is installed in the IDE.

Using templates

Templates are a powerful IDE tool. You can use them to create new objects, which then can be used as a basis for creating more complex objects. Java components such as Swing and AWT containers are one type of template. There are also various templates for classes, dialog boxes, and other objects such as HTML files, text files, and bookmarks. Even "empty" classes have templates. You can also create your own templates.

Creating new objects from templates

You can create a new object from a template in one of the following ways:

- By using the New From Template wizard, which you can access from the Main Window, Explorer, or Object Browser. For details, see "Creating new classes" on page 33.
- By double-clicking on the template itself, either under the `Templates` node in the Global Options window or in the Explorer under the **Filesystems** tab (if the object has been set as a template there).
- By copying the template (using the **Copy** command from the **Edit** menu or toolbar, contextual menu, or keyboard shortcut) and then selecting the target folder and choosing **Paste | Instantiate** from the contextual menu.

After the new object is created, whatever happens when that type of object is double-clicked in the Explorer will occur. For example, the Editor will open if the object supports editing.

Other templates

These templates are all found under the **Other** category.

Bookmark

Choosing this template automatically creates a bookmark with the URL http://www.netbeans.com/ and opens up that web page in the default web browser.

To edit the URL in a bookmark:

◊ In the Explorer, right-click on the bookmark and choose **Edit URL...** in the contextual menu.

A dialog box will appear prompting you to enter the new URL.

Group of Files

A group of files object is a cluster of aliases for files somehow related to each other. It is particularly useful if you want to simultaneously work on files located in separate parts of the IDE. The template itself is "empty". Once you create an object from the template, you can "fill" the object by pasting links from it to other files. You can also save such a group as a new template.

To add a file to a group:

1. In the Explorer, select an object you would like to add to the group and choose **Copy** from the contextual menu.
2. Right-click on the group's node and choose **Paste** from the contextual menu.

HTML

This template is for creating HTML files, which can be viewed in the internal or an external web browser and which can be edited (with syntax coloring features) in the Editor window.

Text

This template is for creating plain text files, which can be edited in the Editor window.

Creating your own templates

In Forte for Java, Community Edition, any object can be used as a template. There are two ways to set an object as a template:

To set an object as a template:

1. Right-click the desired object under the **Filesystems** tab in the Explorer and choose **Save as Template** from the contextual menu.
2. In the Save as Template dialog box that appears, choose the template category where you want the new template placed.

Or:

1. Select the desired object in the Explorer and open its property sheet.
2. Change its **Template** property to `True`.
3. If you want the new template visible in the menus and dialog boxes used for creating objects from templates, copy it and paste it to a template category under the `Templates` node in the Global Options window.

The difference between a template and a normal object is only in what is done when you

double-click on it in the Explorer. For a plain object, a default Editor window is opened (if available) on templates, a dialog box appears enabling you to create a new object from the template.

If you want to open the Editor on a template, choose **Open** from the template's contextual menu in the Explorer.

Modifying existing templates

You can also modify existing templates. This might be desirable if you generally work with non-default service types (for compilers, executors, and debuggers). You can modify the content of a template in the Editor and the properties of a template on its property sheet.

To modify the contents of a template:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, expand the `Templates` node and navigate to the template.
3. Right-click on the template's node and choose **Open** from its contextual menu.

   The template will open in the Editor, where you can edit its contents.

To modify the properties of a template:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, expand the `Templates` node and navigate to the template.
3. Make the necessary changes in the property sheet.

Tip: If you want to make the same change to multiple templates (for example, switch all of them to JPDA debugging), you hold down the CTRL key and select multiple template source files. Any changes you then make on the property sheet will apply to all of the selected templates.

Using macros in templates

You can also use macros in your templates to have various strings automatically generated when you create objects from template. Macros are called in templates in the form `__MacroName__` (with two underscores before and two underscores after the macro name).

The following macros are provided by the IDE and are available for use in templates but can not be modified:

- __NAME__ – name of the class.
- __PACKAGE__ – name of the package that the class is in.
- __PACKAGE_SLASHES__ – name of the class's package, but delimited with slashes (/) instead of periods (.).
- __QUOTES__ – inserts a double quote mark ("). This macro is necessary if you want to place a macro between quote marks. If you enter directly quotes, text substitution for any macros occurring with the quotes will not be performed when an object is created from the template.

Creating and editing macros

The IDE also has the USER macro (signifying the user of the IDE) which you can edit in the Strings Table property editor. In addition, you can add your own macros.

To edit macros or create new macros for use in templates:

1. Choose **Project | Settings...** from the main menu to open the Project Settings window.
2. Select the `Java Sources` node, select the `Strings Table` property, and click the **...** button to open its custom property editor.

   You will see a list of the macros in the form *`MacroName=Value`* (without the two underscores both before and after *`MacroName`*.
3. Make any desired changes to the macro values or add new macros to the list.

Advanced macro feature

The IDE also provides a feature enabling you to determine the suffix in the instantiated object conditionally. The syntax for this feature is

`__MacroName$`*`Param1`*`$`*`Param2`*`$`*`Param3`*`__`

where *Param2* replaces *Param1* if the value of *MacroName* ends in *Param1*. If the value of *MacroName* does not end in *Param1*, *Param3* is returned as the value for the macro.

For example, if you enter the following into a template:

```
__Name$Impl$Spec$Unknown__
```

and you create an object from the template called `MyClassImpl`, the text `MyClassSpec` would be returned by the macro in the new object. If the object you created was merely called `MyClass`, the value `Unknown` would be returned.

Tip: Macros are particular useful when you create groups using the Group of Files template.

You can create your own group, integrate macros into the name and text of the files, and then turn that group into a template.

Adding and modifying service types

Forte for Java, Community Edition provides editable **service types** to give you more control over how your applications are compiled, executed, and debugged. These service types specify not only which compiler, executor, or debugger to use but also how the service is to be called (what the path to Java is, what arguments are used, and so on).

The IDE comes with a set of default service types which suit most development tasks. A compiler type, executor, and debugger type is assigned to each class by default. You can set service types for each class individually on its property sheet.

You can also modify the existing service types and create new ones. For example, the default service type for external execution of applications may not suit every one of your classes, so you may want to have multiple execution service types (or "executors"). You can create a new execution type for external execution, and use this executor for some of your classes while you use the default executor for other classes.

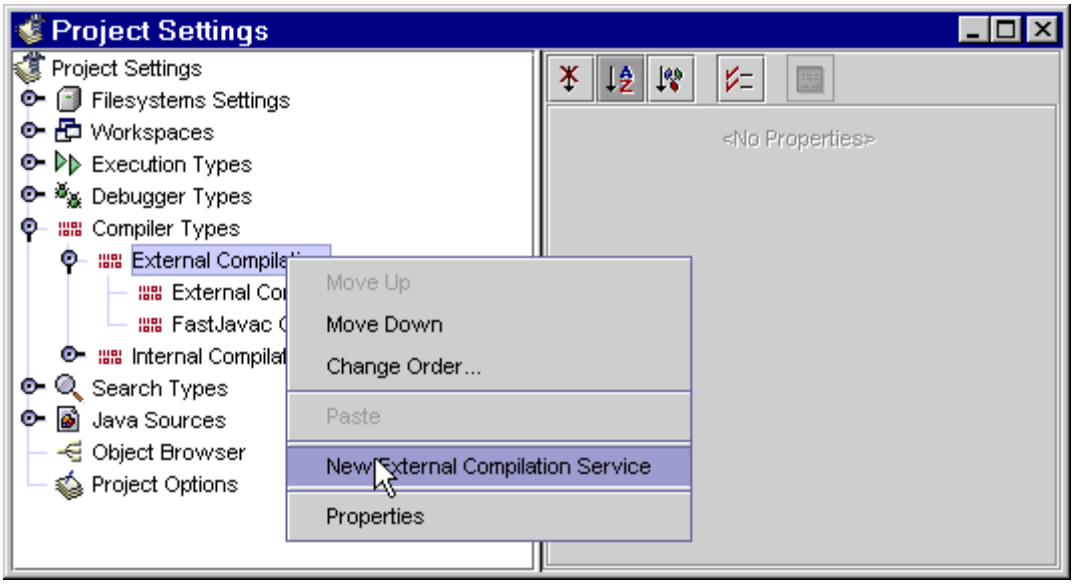
Service types are represented as subnodes under the various categories of `Compiler Types`, `Debugger Types`, and `Execution Types` in the Project Settings window.

Adding service types

To add a new service type:

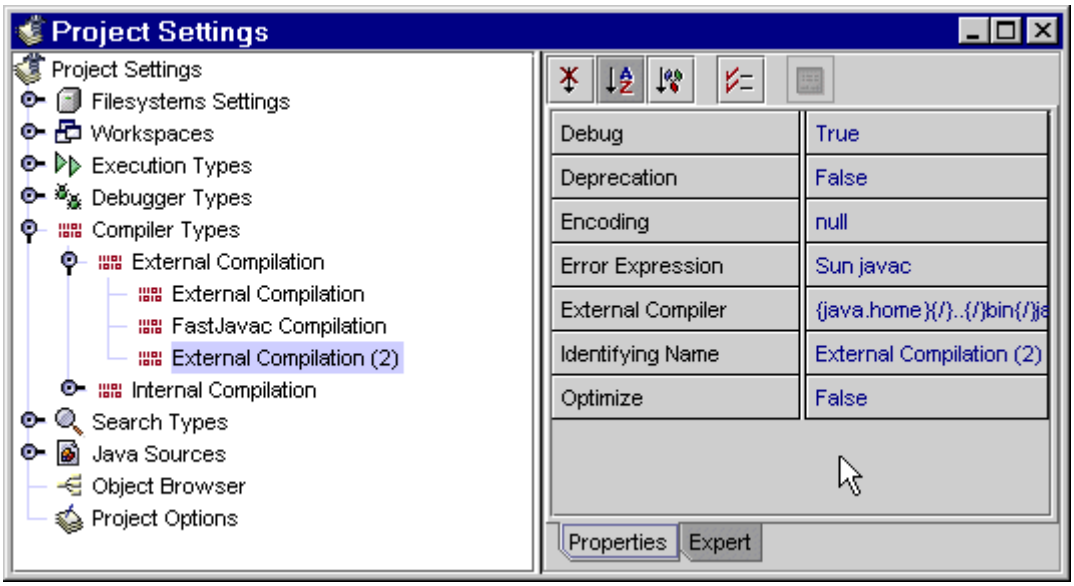
1. Choose **Project | Settings...** from the main menu.
2. Expand the `Compiler Types`, `Execution Types`, or `Debugger Types` node,

depending on the kind of service type you want to add.



3 Right-click on the node for the category of service type you would like to add and choose **New | *[service type]* Service** from the contextual menu.

4 Expand the category's node. You will see at least two subnodes. The last subnode is the newly created service type.



5 Select the newly added node and, on its property sheet, modify the `Identifying Name` property to give the new service type a distinct name (something that you will recognize when using the drop-down list to choose a service type for an individual class).

Note: When you create a new service type using the **New | *[service type]* Service** command, the service type is created with basic settings. If you would like to create a new service type based on the settings of an existing service type, you can select the service type's node in the Project

Settings window, choose **Copy** from its contextual menu, select the category node, and then choose **Paste** from its contextual menu.

Configuring service types

After you have added a new service type, you can then configure it. (You can also configure already-existing service types).

To configure a service type:

1. Choose **Project | Settings...** from the main menu.
2. In the Project Settings window, select the service type's node (under the service type category under `Compiler Types`, `Execution Types`, or `Debugger Types`) and open its property sheet.
3. Modify any of the editable properties under the standard **Properties** or the **Expert** tabs.

When a service type is set for a class, the IDE associates the name of the service type (as it appears in the `Identifying Name` property) to the class, not the actual configuration of the service type. It is possible to create different service types in different projects but give them the same name. Therefore, if you want to associate a different service type for a class, depending on which project you are in, you do not have to change the service type for the class each time you switch projects.

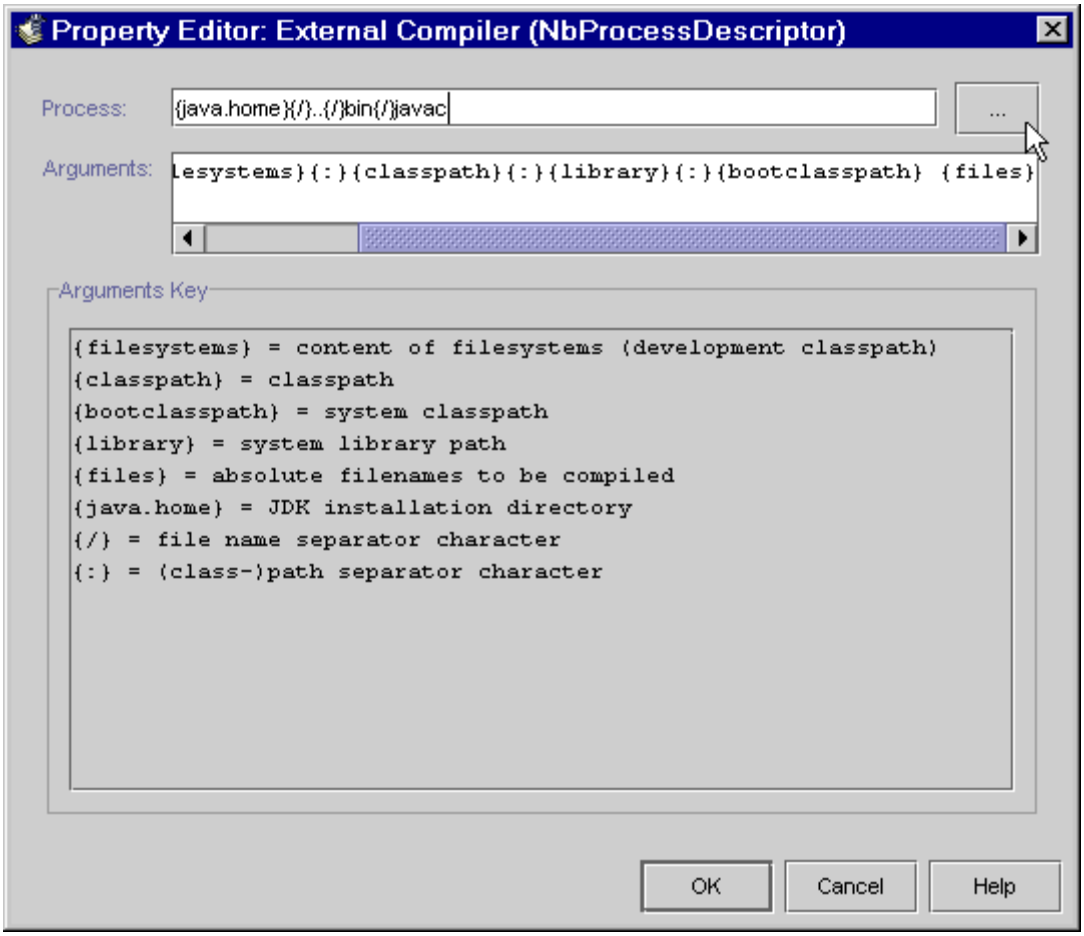
Note: Classes do not register the name changes of the service types they are using. Therefore, if you change the `Identifying Name` property of a service type already being used by any classes, those classes will then revert to the current default service type for that category.

Process Descriptor property editor

Some of the service types have a property (for example, `External Process` for external executors and `External Compiler` for external compiler types) that can be edited with the Process Descriptor property editor. In this custom property editor, you can set the process and arguments for the service type.

You can fill in the Process field manually or press the **...** button to bring up the file chooser to select the process.

The Arguments Key panel displays the substitution codes (enclosed in curly braces) which you can use (and are used by default) in defining the service type. The values for substitution codes such as `{repository}` and `{classpath}` are shown under the **Expert** tab of the service type's property sheet.



Note: The `{arguments}` substitution code uses any arguments entered in the `Arguments` property of the class being run.

Tip: If you use a lot of different service types, you can save time by incorporating them into object templates, which you then can use each time you create a new object. Just create a generic class, set the desired service types on the class's property sheet, and then save the class as a template. For more information, see "Creating new objects from templates" on page 200.

Editing service types from the Filesystems tab in the Explorer

You can also access the property sheet for the different service types from the property sheet of an individual class's node in the Explorer.

To access a service type's property sheet from the Explorer:

1. Select a class under the **Filesystems** tab in the Explorer and open its property sheet.
2. Under the **Execution** tab of the property sheet, select the `Compiler` (or `Executor` or `Debugger`) property and then press the **...** button to bring up the custom property

editor.

3 In the list on the left side of the custom property editor, select the compiler/executor/debugger you want to modify. The property sheet for that service type will appear. (You can select the `External Compiler` (or `External Process` or `Debugger`) property and press the **...** button in order to edit the service type's process and arguments.)

Note: When you reconfigure a service type, even if you are invoking the custom property editor(s) from the property sheet of a class, the changes you make to the service type will affect all other classes that use that service type.

Removing service types

To remove a service type:

◊ Right-click on the service type's node in the Explorer and choose **Delete** from the contextual menu.

Customizing the environment

A big advantage of Forte for Java is that you can customize the graphical user interface (GUI) to best fit the way you work. That means that you can determine what commands are shown in the existing menus and toolbars, add your own commands, and create new menus and toolbars. You can rearrange the toolbars with your mouse. You can also develop special workspace configurations, set your own keyboard shortcuts for the available commands, and customize the contextual menus for various nodes.

Changing the look and feel

The Forte for Java user interface offers different "looks" based on the platform being used. By default, the IDE is set to the Java look and feel. If you prefer to use the Microsoft Windows or CDE Motif look and feel, you can edit the IDE's startup script to change the look and feel.

To change the IDE's look and feel on Windows machines:

1 In the installation directory, open the `bin` folder and select the `forte4j.cfg` file.

2 Open `forte4j.cfg` in a text editor and add one of the following switches to the startup script:

a. `-ui com.sun.java.swing.plaf.windows.WindowsLookAndFeel`, if you

want to change to the Windows look and feel.

b. `-ui com.sun.java.swing.plaf.motif.MotifLookAndFeel`, if you want to change to the Motif look and feel.

If you want to change back to the Java look and feel, open `forte4j.cfg` and delete the switch you previously added for the Windows or Motif look and feel.

To change the IDE's look and feel on Solaris and Linux machines:

1. In the installation directory, open the `bin` folder and select the startup script (`forte4j.sh` for single users or `forte4j_multiuser.sh` for multi-user installations)
2. Open the startup script in a text editor and add one of the following switches to the startup script:
   a. `-ui com.sun.java.swing.plaf.windows.WindowsLookAndFeel`, if you want to change to Windows look and feel.
   b. `-ui com.sun.java.swing.plaf.motif.MotifLookAndFeel`, if you want to change to Motif look and feel.

Configuring the Editor

Editor features such as keyboard shortcuts, abbreviations, code completion, and automatic code formatting are configurable. The following procedures show you how you can customize them to suit your needs.

Changing assignments for Editor keyboard shortcuts

If the key combinations for the shortcuts do not suit you, you can change them. By default, the Editor has a set of global keyboard shortcuts set. These shortcuts are available no matter what kind of file you are working on in the Editor.

In addition, it is possible to set keyboard shortcuts by individual type of editor (Java, Plain, HTML, and JSP). If you use assign a keyboard shortcut for one of these individual editors that has the same key combination as a global shortcut, the shortcut for the specific editor will override the global shortcut for that editor.

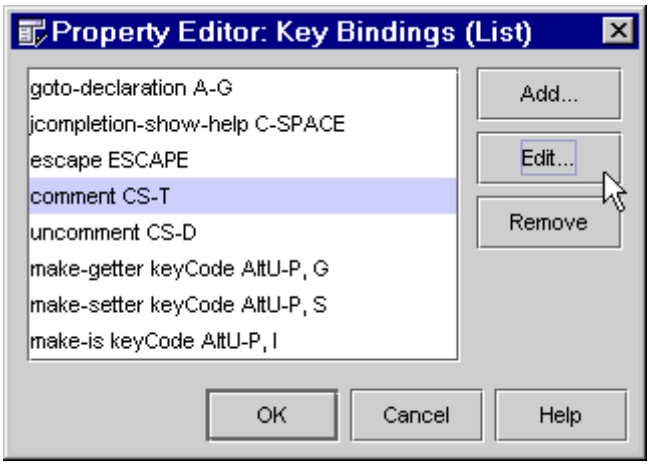
For example, the Alt+f key combination is set as a global keyboard shortcut for reformatting text. If you wanted to use this keyboard shortcut for something else when editing Java files (but keep it the same when editing any other types of files), you could reassign Alt+f to a different action on the property sheet for the `Java Editor` node.

Note: The Editor's shortcuts are distinct from other IDE shortcuts, which can be configured by selecting **Tools | Configure Shortcuts** from the main menu. See "Customizing IDE

shortcuts" on page 219 for information on configuring non-Editor shortcuts.

To change the Editor's global keyboard shortcut assignments:

1. Choose **Tools | Global Options...** from the main menu.
2. Select the `Editor Settings` node in the Global Options window.
3. Select the `Global Key Bindings` property and click on the **...** button. This will bring up the Key Bindings custom property editor.



4. In the custom property editor, select the shortcut assignment you want to change and press **Edit**.
5. In the Edit dialog box that appears, type the new Key for the shortcut. Use the following abbreviations for special keys and separate the special keys from the final character in the key combination with a hyphen (-). Press **OK** to confirm the change.

   The special keys should be entered in the following form:

   - Shift key – S
   - Ctrl key – C
   - Alt key – A
   - ENTER key – ENTER
   - SPACE key – SPACE
   - BACKSPACE key – BACK_SPACE
   - TAB key – TAB
   - INSERT key – INSERT
   - DELETE key – DELETE
   - LEFT arrow key – LEFT
   - RIGHT arrow key – RIGHT
   - UP arrow key – UP
   - DOWN arrow key – DOWN
   - HOME key – HOME

- END key – END
- Page up key – PAGE_UP
- Page down key – PAGE_DOWN

Multi-step keyboard shortcuts (where a combination of keys of pressed and released and then another key is pressed) are marked in the form `AltU-P,f` (meaning press ALT+u and then press f).

Note: You cannot set a keyboard shortcut for anything that is not in the **Action** combo box in the Edit dialog box.

To change keyboard shortcuts for a specific editor type:

1. Choose **Tools | Global Options...** from the main menu.
2. Expand the `Editor Settings` node in the Global Options window and select the subnode for the editor type you want to set the shortcut for.
3. Select the `Key Bindings` property and click on the **...** button. This will bring up the Key Bindings custom property editor.
4. Follow steps 4 and 5 of the previous procedure.

Customizing Editor abbreviations

For each type of editor, you can set and modify abbreviations for use in the Editor window. By default, abbreviations are already set for the Java and JSP editors.

To change the Abbreviation table:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, expand the `Editor Settings` node and select the subnode for the type of editor you want to configure abbreviations for.
3. Select the `Abbreviations` property and click on the **...** button. This will bring up a custom property editor for abbreviations.
4. Select the abbreviation that you want to change in the list box and click **Edit**.
5. Another dialog box will appear with a combo box list for the abbreviations (**Action**) with the selected one showing and a field showing the selected expansion (**Key**). Change either or both values and click **OK** to make the changes or **Cancel** to go back to the Abbreviations dialog box.

To add abbreviations:

◊ Follow the same steps as for changing abbreviations, but click **Add** in the Abbreviations dialog box instead of **Edit**.

Customizing Java code completion

Changing the key bindings for code completion

Just as you can with other Editor shortcuts, you can change the shortcut assignments for Java code completion.

To change the keyboard shortcut assignments for code completion:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, expand the `Editor Settings` node and select `Java Editor`.
3. On the property sheet, select the `Key Bindings` property and click on the **...** button. This will bring up the Key Bindings custom property editor.
4. In the custom property editor, select the shortcut assignment you want to change (`jcompletion-show-help` or `jcompletion-hide`) and press **Edit**.
5. In the Edit dialog box that appears, enter the new key for the shortcut and press **OK**.

Setting the code completion box delay

You can set the time it will take for the code completion box to automatically pop up.

To set the code completion delay:

◊ In the Global Options window, select `Editor Settings / Java Editor` and set the `Delay of Java Code Completion Popup` property to the delay you would like in milliseconds.

Disabling code completion

To disable code completion:

◊ In the Global Options window, select `Editor Settings / Java Editor` and set the `Auto Popup of Java Code Completion` property to `False`.

Java Editor formatting options

There are two options for the formatting of Java code generated by the Form Editor or through code completion:

- `Add Space before Parenthesis` – if `True`, whenever parentheses are used to enclose parameters, a space precedes the opening parenthesis.
- `Curly Brace on Next Line`– if `True`, the opening curly brace is placed on a new line.

These options can be accessed by choosing **Tools | Global Options...** and selecting the `Editor Settings / Java Editor` node.

Caret row highlighting

You can also have the current line in the Editor (the line with the insertion point) highlighted. This setting is configurable separately for each type of editor.

To switch on caret row highlighting for an editor:

1. Choose **Tools | Global Options...** from the main menu.
2. Expand the `Editor Settings` node and select the type or types of editor for which you would like to turn on the highlighting.
3. Click the **Expert** tab of the property sheet.
4. Change the `Highlight Caret Row` property to `True`.

The color of the caret row highlighting can be customized. Follow the procedure in "Setting fonts and text coloring in the Editor", choosing the **highlight-row** item in the Fonts and Colors dialog box.

Setting fonts and text coloring in the Editor

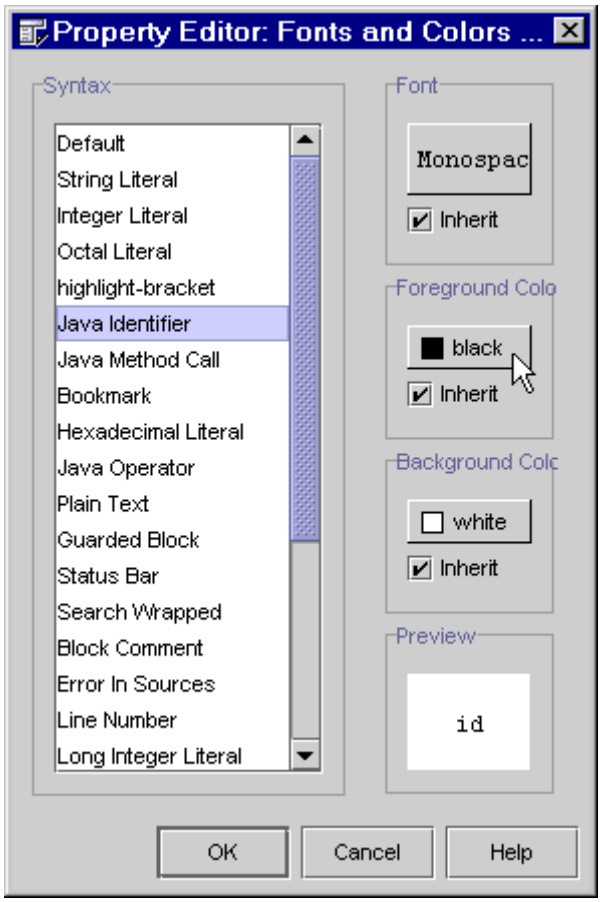
You can customize the fonts and background and foreground colors for various types of text (tokens) in the Editor using the `Fonts and Colors` property for each type of editor found under `Editor Settings` in the Global Options window. The following are some of the tokens you can set fonts and colors for:

- default text (meaning that the fonts and colors assigned here are the ones used for all types of text, except where overridden by configuration of the specific text types).
- different types of values such as strings and integers
- text blocks such as current caret row, selected text, editor bookmarks, and guarded (uneditable) text
- block comments and line comments
- tokens such as methods, identifiers, and keywords
- items found in incremental searches and items found in highlight searches
- status bar and status bar when highlighted

To change the font or color for a token:

1. Choose **Project | Settings...** from the main menu.

2. In the Global Options window, expand the `Editor Settings` node and choose an editor type (`Java`, `HTML`, `Plain Text`, or `JSP`).
3. Click the **Properties** tab of the property sheet.
4. Click in the value field for the `Fonts and Colors` and click the **...** button to bring up the Fonts and Colors property editor.



5. In the Syntax panel, click the token you would like to change the coloring of.
6. The right pane of the dialog consists of the following four panels:

   - **Font** – where you can modify the face, style, and size of the font
   - **Foreground Color** – where you can set the color of the text
   - **Background Color** – where you can set the background color for the text
   - **Preview** – which displays the font and color scheme you have chosen

7. The first three panels each have an **Inherit** checkbox. If you check this box in a category, the text type will inherit the font or color from the default text type (or from another token it is dependent on).

   These panels each have a button showing the current font or foreground or background color or font for the selected token in the left pane. Press one of these buttons if you want to change the value it displays.

a. If you have pressed the button in either the **Foreground Color** or **Background Color** panel, press the drop-down arrow that appears and choose a color or press the **...** button to bring up the Color Editor.

   If you are using the Color Editor, choose one of the six color palettes (Swatches, HSB, RGB, AWT palette, Swing palette, and System palette) by clicking its tab, choose the color you want within the palette, and then press **OK** to return to the Fonts and Colors property editor. See "Color Editor" on page 288 for more information on using the different color palettes.

b. If you have pressed the button in the **Font** panel, you can then press the **...** button to open the Font Property Editor, select a font face, style, and size, and then press **OK** to return to the Fonts and Colors property editor.

8 Click **OK** in the Fonts and Colors property editor to record your changes.

Using an external editor

It is possible to edit files from outside of the IDE, while the file is open in the IDE. Periodically, the IDE will check to see if any changes have been made to the file from outside of the IDE and prompt you whether to load them or not. If you want the IDE to immediately reflect changes you have made externally, select the root node for the file system in the Explorer and choose **Refresh Folder** from the contextual menu.

Additional Editor Settings

The property sheets for the different editor types (found under `Editor Settings` in the Global Options window) provide many preferences that you can set, including type of caret (insertion point), insets, and status bar caret delay. See "Editor Settings reference" on page 265 for more information.

Customizing menus and toolbars

You can organize all menu and toolbar commands in the IDE to your preferences so that you have quick and easy access to them. From a customization point of view there are only minor differences between the menu and toolbars, because they are both used as generic containers to visually display a list of commands.

Changing commands in menus and toolbars

You can add and delete existing commands on the menus and toolbars using clipboard-style operations.

To add a new command to a menu or toolbar:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, expand the `Actions` node and then the appropriate category node.
3. Select the action you want to add and choose **Copy** from the **Edit** menu or contextual menu or by keyboard shortcut.
4. Under the `Menu` (or `Toolbars`) node in the same window, select the subnode for the menu or toolbar under which you would like to place the command.
5. Choose **Paste | Copy** from either the **Edit** menu or contextual menu.

To move a command to a different menu or toolbar:

1. In the Global Options window, expand the `Menu` or `Toolbars` node and navigate to the command you want to move.
2. Select the action you want to move and choose **Cut** from the **Edit** menu or contextual menu or by keyboard shortcut.
3. Under the `Menu` (or `Toolbars`) node, select the subnode for the menu or toolbar under which you would like to place the command.
4. Choose **Paste** from either the **Edit** menu or contextual menu or by keyboard shortcut.

Note: If you delete a command supplied with Forte for Java from the default menus or toolbars, it is not completely deleted from the IDE. You can restore a command to a menu or toolbar by copying it from the `Actions` hierarchy in the Global Options window (where all available commands are stored) and pasting it to a menu (listed under `Menu`) or toolbar (`Toolbars`) in the same window.

To group related menu commands with a separator:

1. Choose **Tools | Global Options...** from the main menu.
2. In the Global Options window, navigate to the node of the desired parent container under `Menu`.
3. Right-click on the node and choose **New | Menu Separator** from the contextual menu.
4. Enter a name for the separator in the dialog box that appears and click **OK**.

   The menu separator will appear as the last item in the menu.

To change the order of commands within their parent container:

- In the `Menu` (or `Toolbars`) hierarchy of the Global Options window, select the command you want to move, and move it using the **Move Up** or **Move Down** command in the contextual menu; or
- In the `Menu` (or `Toolbars`) hierarchy of the same window, select the parent container and choose **Change Order** from its contextual menu. A dialog box will appear which will enable you to rearrange the commands by dragging and dropping with the mouse or by using the **Move Up** and **Move Down** buttons on them.

Creating new menus and toolbars

You can also create new menus and toolbars in the IDE.

To create a new menu:

1. Choose **Tools | Global Options...** from the main menu.
2. Right-click on the `Menu` node, and choose **New | Menu** from the contextual menu.
3. In the New Menu dialog box that appears, type the name of the new menu and click **OK**.

To create a new toolbar:

1. Choose **Tools | Global Options...** from the main menu.
2. Right-click on the `Toolbars` node and choose **New | Toolbar** from the contextual menu.
3. In the New Toolbar dialog box that appears, type the name of the new toolbar and click **OK**.

Dragging toolbars

The Main Window toolbars can be managed with mouse operations and contextual menus directly in the Main Window.

To move a single toolbar:

◊ Click on the "handle" on the left side of the toolbar (marked by two light-colored vertical bars) and drag it with the mouse.

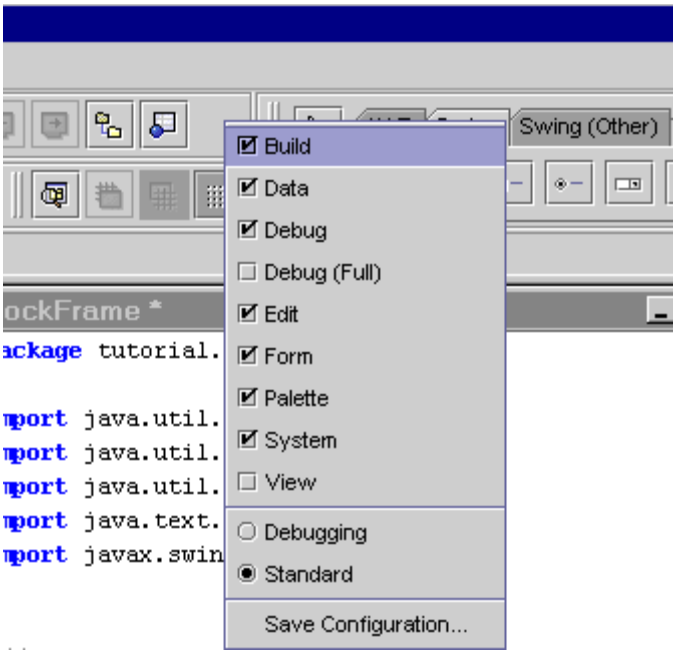
To move consecutive toolbars simultaneously:

◊ Right-click on the "handle" of the left-most of the toolbars and drag it with the mouse. That toolbar and all toolbars to its right will be moved.

Toolbar contextual menu and toolbar configurations

There is also a contextual menu on the Main Window which you can use to manage toolbars. You can access this contextual menu by right-clicking on any empty space in the toolbar area

of the Main Window.



The contextual menu is divided into three parts:

- a list of available toolbars
- a list of toolbar configurations
- the **Save Configuration** command

The available toolbars are listed with checkboxes. By checking and unchecking these items, you can control which toolbars are displayed in the Main Window.

The toolbar configurations are shown with radio buttons, enabling you to check the configuration (group of toolbars and their positions) you want. By default, there are two configurations. The **Standard** configuration includes the **Debug** toolbar, which contains only a selection of debugging commands, and is for all workspaces except Debugging, by default. The **Debug** configuration has the **Debug (Full)** toolbar with all Debugger commands and is used in the Debugging workspace by default.

The **Save Configuration** command enables you to save the current configuration of toolbars and create a name for it (thus creating a new configuration).

You can set the toolbar configuration separately for each workspace.

To set a toolbar configuration for a workspace:

1. Choose **Project | Settings...** from the main menu.
2. Select the node for the workspace under `Workspaces` in the Project Settings window.

3 On the workspace's property sheet, choose the configuration from the drop-down list on the `Toolbar Configuration` property.

Changing commands in contextual menus

You can also change commands that appear in various contextual menus in the Actions property editor. The items in the various contextual menus are determined by the type of object the menu is for.

To open the Actions property editor for an object type:

1 Open the Global Options window by choosing **Tools | Global Options** from the main menu.

2 Expand the `Object Types` node and select the subnode for the object type you would like to change the contextual menu for. (You can determine which object type to change by matching the icon of one of the nodes you want a different menu for and finding the same icon in the list of object types.)

3 Select the `Actions` property and then press the **...** button that appears.

A two-pane custom property editor will appear. The left pane displays available commands that you can add to the contextual menu. The right pane displays the commands currently in the contextual menu.

4 Change the contents of the menus in the following ways

- To add a command to the contextual menu, select the command in the left panel and press **Add**.
- To add a separator to the contextual menu, press **New Separator**.
- To change the order of menu items, select the item you want to move in the right panel and then press **Move Up** or **Move Down** until it is in the position you want.
- To delete a command or separator, select the item and press **Remove**.

5 Press **OK** to confirm changes and close the property editor.

Customizing IDE shortcuts

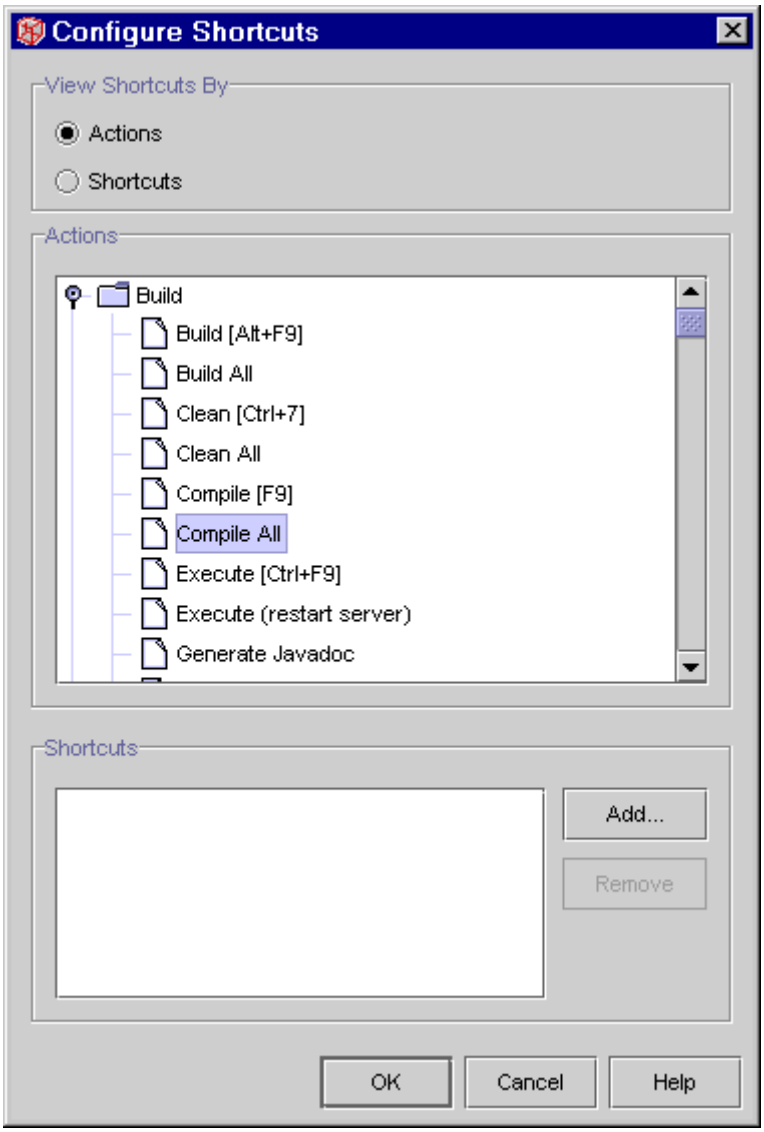
Keyboard shortcuts offer an alternative method of performing operations for users who prefer the keyboard to the mouse. They are particularly useful for frequently-used commands, where it is easier to use a combination of keystrokes than to lift fingers from the keyboard to choose the operation from a menu using the mouse.

Keyboard shortcuts can be configured in the Configure Shortcuts dialog box, where you can view a list of current keyboard shortcuts, either by the shortcuts assigned or by the list of all IDE actions to which keyboard shortcuts can be applied.

The Editor has its own set of shortcuts that can be configured separately. See "Changing assignments for Editor keyboard shortcuts" on page 209.

To add a keyboard shortcut:

1. Select **Tools | Configure Shortcuts...** from the main menu.
2. In the Configure Shortcuts dialog box that appears, select the **Actions** radio button to view a list of all actions by category and the keyboard shortcuts applied to them.
3. Select one of the listed actions.



4. In the Shortcuts pane, press the **Add...** button.
5. In the Add Shortcut dialog box that appears, type the key combination that you wish to be the shortcut. For example, for CTRL+ALT+7, hold down the CTRL and ALT keys and type 7.
6. Close the two dialog boxes by pressing the **OK** button in each of them.

To remove a keyboard shortcut:

1. Select **Tools | Configure Shortcuts...** from the main menu.
2. In the Configure Shortcuts dialog box that appears, select the **Shortcuts** radio button to view a list of all shortcuts currently assigned.
3. In the Shortcuts pane, select the shortcut you would like to remove and then press the **Remove** button.
4. Close the Configure Shortcuts dialog by pressing the **OK** button.

Note: You can have multiple keyboard shortcuts for the same action.

Customizing the Component Palette

You can also customize the Component Palette. In the Global Options window under `Component Palette` you can find the current Component Palette categories. By expanding a category node, you can view a list of all components in that category.

To create a Palette category:

1. Choose **Tools | Global Options...** from the main menu.
2. Right-click on the `Component Palette` node and choose **New Palette Category** from the contextual menu.
3. Enter the new category name in the dialog box and click **OK**.

   The new category will appear as the last item in the component category listing, and a tab for the new category will appear in the Component Palette.

Standard clipboard operations are available at each level of the Explorer's `Component Palette` hierarchy. Individual components can be cut, copied, and pasted between component categories. For example, you can copy the components you use most to a new category for quick and easy access. You can also cut, copy and paste component categories.

Component categories can be renamed—either by choosing **Rename** on the contextual menu or using in-place renaming in the Explorer.

It is also possible to customize the ordering of component categories and the components within a category.

To move a component category:

- In the Global Options window under `Component Palette`, right-click on the category node and choose **Move Up** or **Move Down** from the contextual menu; or
- Right-click on the `Component Palette` node and choose **Change Order** from the contextual menu to bring up the Customize Order dialog box, which lists all of the categories, has **Move Up** and **Move Down** buttons, and also enables you to re-order items by

drag-and-drop.

To change the order of a component within a category:

◊ In the Global Options window under `Component Palette`, right-click on the node for the component and choose **Move Up** or **Move Down** from the contextual menu; or

◊ Right-click on the node of the component's category and choose **Change Order** from the contextual menu to bring up the Customize Order dialog box.

To remove a component or component category from the Component Palette:

◊ In the Global Options window under `Component Palette`, right-click on the node for the component or the category and choose **Delete** from the contextual menu.

For information on adding JavaBeans components to the Component Palette, see "Adding JavaBeans components to the IDE" on page 107.

Customizing workspaces

Workspaces are entirely customizable through the `Workspaces` node in the Project Settings window. Choose **Project | Settings...** from the main menu and expand `Workspaces` to see a list of current workspaces. These workspace nodes can in turn be expanded to see a list of currently open windows on each workspace.

Each level of this hierarchy can be cut, copied and pasted, using commands on either the contextual menu or the **Edit** menu of the Main Window. Existing workspaces can be renamed, again through the contextual menu, the workspace's property sheet, or using in-place editing of the name in the Explorer.

Workspace clipboard operations act recursively: copying a workspace copies all windows present on that workspace. If you then paste the new workspace to the workspace hierarchy, these windows are copied to that new workspace.

On the property sheet for each workspace node is the `Toolbar Configuration` property, where you can choose from among different sets of toolbars to display in the workspace. See "Toolbar contextual menu and toolbar configurations" on page 217 for information on creating your own toolbar configurations.

To create a new workspace:

1 Choose **Project | Settings...** from the main menu.

2 Right-click on the root `Workspaces` node and choose **New Workspace** from the contextual menu.

A dialog box will open enabling you to specify the name of the new workspace.

3 Type the workspace name and click **OK**.

The new item will appear in the given tree in the Project Settings window and immediately be displayed on the Main Window.

If you need to expand the workspace tab area on the Main Window to see your new workspace, click and drag the workspace tab divider to the right. If there are more workspaces than are visible in the workspace tab area, left and right scroll buttons will be displayed enabling you to scroll through all available workspaces.

To delete a workspace:

◊ Right-click on the workspace's tab in the Main Window and choose **Delete *WorkspaceName*** from the contextual menu.

To add windows to the new workspace, you can either:

◊ Choose the workspace by clicking on its tab in the Main Window and then open up the windows that you want to be on that workspace from the **Windows** menu or toolbar; or

◊ Under the `Workspaces` tab in the Project Settings window, copy windows from other workspace nodes and paste them to the new workspace.

Note: Only the first option works if you want to add a window not already in a workspace to the new workspace.

You can also customize which workspace the IDE automatically switches to when you start execution or debugging or open a form.

To set the workspace the IDE switches to when you open a form object:

1 Choose **Tools | Global Options...** from the main menu to open the Global Options window and select the `Form Objects` node.

2 Click on the value of the `Workspace` property to access the drop-down list of available workspaces, and then choose a workspace. You can also set this property to `None` if you do not want the workspace to change when you open a form object.

To set the workspace the IDE switches to for execution:

1 Choose **Tools | Global Options...** from the main menu to open the Global Options window and select the `Execution Settings` node.

2 Click on the value of the `Workspace` property to access the drop-down list of available workspaces, and then choose a workspace. You can also set this property to `None` if you do not want the workspace to change when you run a class.

To set the workspace the IDE switches to for debugging:

1 Choose **Tools | Global Options...** from the main menu to open the Global Options window and select the `Debugger Settings` node.

2 Click on the value of the `Workspace` property to access the drop-down list of available workspaces, and then choose a workspace. You can also set this property to `None` if you don not want the workspace to change when you start the Debugger.

For more information on workspaces, see “Workspaces” on page 188.

Opening files from outside of the IDE

Many programs have the ability to associate specific commands with file types (generally by extension or MIME type), so that the user may select a program to view or edit that type of file. Examples include the Microsoft Windows desktop and Explorer; assorted file browsers available for Unix and other operating systems, as well as alternatives on Windows machines; web browsers; version control systems; mail programs; and so on. If you wish, you can associate the IDE with certain file types, such as Java sources or property bundles, so that the IDE's editor will be opened when you double-click on the file.

In Windows installations of Forte for Java, the installer gives you an option to associate .java files with Forte for Java. If you select this option, you can double-click on `.java` files outside of the IDE to open them in the IDE, when the IDE is running.

Using the remote launcher manually

To make this work, first try opening a file in the IDE using the remote launcher manually. Open a command prompt appropriate for your system (for example, MS-DOS console or Unix shell). Find the JAR file for the Open File module – probably included in your Forte for Java installation. Now decide on a file to open and type something like this:

```
C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
     Files\forte4j\modules\utilities.jar"
 "C:\My Development\com\mycom\MyClass.java"
```

If the IDE is running, you should see `MyClass.java` displayed in the Editor, or the usual mount dialog box may appear. See “Adding a file to the IDE” on page 77. Please specify complete paths to all files to make sure the same command will work later. The quotes are desirable, especially in the Windows environment, in case you have any directories with spaces in their names.

Associating the launcher with a type of file

Now you are ready to associate the Open File launcher with some file type in your application. The details of how to do this will of course vary depending on the system. Generally, you must specify everything before the filename in the above example command as the “external editor”.

As an example, to associate the IDE with `*.java` files on Windows machines:

1. Open any Windows Explorer window and choose **Tools | Global Options...**. Go to the **File Types** tab.
2. See if there is some kind of entry for Java source files (the name may vary, depending on what software has been installed) – the entry should specify the file extension `java`. Click **New Type...** if necessary; otherwise select the entry and click **Edit...**.
3. In the **Actions** box, you need at least one action which opens the file in the IDE – you can name it whatever you like, usually `Open`. Click **New...** or **Edit...** as appropriate.
4. In the resulting action dialog box, supply whatever name you like, and give as the application the command you typed before, but replacing the filename with `%1`:

```
C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
      Files\forte4j\modules\utilities.jar" "%1"
```

5. Normally you will also click **Set Default** (it will be bold if it is the default) to make it the default action for double-clicks.
6. If desired, you may similarly associate other file types – recommended are `*.properties`, `*.ser`, `*.form`, `*.class`, `*.jar`, `*.zip`, and so on.

   Now you should just be able to double-click any of these files in the Windows environment and the IDE will open them for you. For other systems such as Solaris and Linux, if you use a file browser (such as from KDE or Gnome, or a separate browser) you can similarly set up file associations.

Customizing file opening

The Open File launcher has several options which can be used to customize the process of opening files remotely. You may pass these on the command line to the launcher, for example:

```
C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
      Files\forte4j\modules\utilities.jar" -port 2121 "C:\My
      Development\com\mycom\MyClass.java"
```

Note that you may also pass multiple files at once to the launcher. The available options are:

`-host` ***`hostname-or-IP`*** – select an alternate machine to open files on. The specified machine should be the one running the IDE. You may want to also turn on `-nocanon` (below). The Open File server must be configured to accept requests from the machine running the command (below).

`-port` ***`port-number`*** – Select an alternate port to send the Open File requests to (via UDP). By default, port 7318 is used. The Open File server must be configured to listen on that port (below).

- `-canon` (default) or `-nocanon` – when using `-canon`, any relative file names passed to the launcher are first converted to an absolute name. If using `-host` to open files remotely, you may want to use `-nocanon` and always specify the correct absolute file name for the machine running the IDE.

- `-wait` and `-nowait` (default) – by default, the launcher sends a message to the IDE asking it to open a file, and then immediately exits as soon as the IDE has opened it. (The command will return a success code (zero on Unix) normally, or an error code (nonzero on Unix) if there was a problem opening the file.)

  Some applications demand that an editor be used to make a specific set of changes to a file and then finish with it. If this is the case, specify `-wait` to the launcher. For example, many version control systems on Unix provide the option to edit a change log message with an external editor, or to clean up the results of a merge before continuing with a check-in. If you use such a system, and set the Open File launcher with `-wait` as your external editor, then the file will open in the IDE as usual, but the launcher will wait to exit. The launcher will not exit until you have made some changes to the file and saved them. Currently it is not sufficient to just close the Editor window – you must actually save changes. If you decide you do not want to make any changes, simply type any character, delete it, and then ask to save (to let the launcher know you are finished with the file).

- `-help` – display a brief usage message.

- `-line` – open the file at the specified line number. Line numbers start at one.

Open File Server properties

When using the IDE as an external viewer or editor, you can control the behavior of the server that listens to open requests. Choose **Tools | Global Options...** from the main menu and select the `Open File Server` node in the Global Options window. You can configure how the server should behave by adjusting the following properties:

- **Access Restriction** – By default, only open requests from the same machine are honored, so that other people cannot open files in your IDE. If you set this property to `Any Host`, any incoming requests will be honored. (Many local area networks prohibit arbitrary traffic from entering the network from the Internet, so this is not necessarily as unsafe as it sounds. Also note that the request is sent through UDP, which most firewalls will not pass.)

- **Port** – You may configure the port the server runs on, which might be necessary if another program is using port 7318.

- **Running** – You may set this property to `False` to stop the server, or `True` to turn it back on.

Stopping the server only affects using the launcher – it does not affect opening files from within the IDE using the toolbar button or menu item.

Appendix A

Default Keyboard Shortcuts

Table 9: Naming Convention for Function Keys

Name	Used For
LEFT, UP, RIGHT, DOWN	Insertion point arrow keys
BACKSPACE, SHIFT, ENTER, DELETE	These keys correspond to common special keys on most keyboards.
END, HOME, PgUp, PgDown	Other insertion point repositioning keys
F1, F2, ... F12	Numbered function keys (across the top of the keyboard)

Modifier keys

Most keyboard shortcuts use one or more modifier keys such as ALT, CTRL, META, and SHIFT, which you must hold down while pressing the other key in the combination.

Note: These names are conventional, but some systems (especially the X Window System) may use different names. For example, META is often equivalent to ALT.

Global Shortcuts

Context-Sensitive Help

You can get context help for a component in the GUI by placing the mouse cursor over that item and pressing F1. Components that have their own context help include windows, dialog boxes, nodes, and window tabs.

Clipboard

Table 10: Clipboard shortcuts

Press	*To*
CTRL+c	Copy current selection into the clipboard.
CTRL+x	Cut current selection into the clipboard.
CTRL+v	Paste current content of the clipboard.
DELETE	Delete selection.

Note: In some cases, it is better to call the Paste command from the contextual menu for a node in the Explorer, because more specific options are provided. For example, when copying a file from one directory to another, you have the option of creating a duplicate file or a link to the original file.

Form Editor Shortcuts

These shortcuts are useful when you are designing visual forms using the Form Editor.

Table 11: Form Editor shortcuts

Press	*To*
CTRL+F10	Switch to Form Editor window.
CTRL+F11	Switch to Editor window.
CTRL+F12	Switch to Component Inspector.

Explorer Shortcuts

The following items are associated with the Explorer (though the Explorer does not have to be open to use all of them).

Table 12: Explorer shortcuts

Press	*To*
CTRL+f	Open the Customize Criteria dialog box to search Filesystems (or, if the Editor window is active, open the Find dialog box to search for text in the Editor).
CTRL+n	Create New Object.
CTRL+o	Open Explorer or activate currently open Explorer window.
CTRL+s	Save .
CTRL+SHIFT+s	Save All (unsaved documents).
DELETE	Delete (selected node).

Window Shortcuts

The following are shortcuts for opening or activating specific windows and other window-related functions.

Table 13: Window shortcuts

Press	*To*
ALT+0	Explore From Here.
ALT+1	Open the selected object's property sheet in a separate window.
ALT+2	Open/activate the Explorer window (same as CTRL+o).
ALT+3	Open/activate the Editor window (works only if a text or source file is already open).
ALT+4	Open/activate the Output Window.
ALT+5	Open Debugger Window.
ALT+6	Open/activate the Execution View.

Press	To
ALT+7	Open/activate the Web Browser.
ALT+8	Open/activate the Component Inspector.
CTRL+F4	Close the active window (or tab in multi-tab window).
CTRL+ALT+LEFT	Switch to previous workspace.
CTRL+ALT+RIGHT	Switch to next workspace.
ALT+LEFT	Switch to previous tab (in multi-tab window).
ALT+RIGHT	Switch to next tab (in multi-tab window).
CTRL+u	Undock window (undocks currently activated tab in multi-tab window into a single window) .
CTRL+p	Print the file selected in the Editor, Explorer, or Object Browser.
CTRL+F1	Do Javadoc index search (if Javadoc module is installed).

Project Shortcuts

These shortcuts keys mirror commands available in the **Project** menu.

Table 14: Project shortcuts

Press	To
SHIFT+F9	Compile project.
CTRL+SHIFT+F9	Execute project.
CTRL+SHIFT+F5	Debug project.

Build Shortcuts

These shortcuts keys mirror commands available in the **Build** menu.

Table 15: Compile/Build shortcuts

Press	*To*
F9	Compile all out-of-date files under selected node
ALT+F9	Build (compile all files) under selected node
ALT+c	Stop Compilation
ALT+F7	Go to Previous Error Line
ALT+F8	Go to Next Error Line
CTRL+F9	Run

Debugger Shortcuts

These shortcuts mirror commands available in the **Debug** menu.

Table 16: Debugger shortcuts

Press	*To*
F5	Start Debugging.
ALT+F5	Continue Debugging.
SHIFT+F5	Finish Debugging.
CTRL+F8	Add/Remove Breakpoint.
SHIFT+F8	Add Watch.
F7	Trace Into.
F4	Go to Cursor.
F8	Trace Over.
CTRL+F7	Trace Out.
ALT+5	Open Debugger Window.

Editor Shortcuts

Keyboard shortcuts in the Editor can be loosely divided into three categories: navigation, edit, and Java. Navigation shortcuts encompass everything from using the arrow keys on the keyboard to finding the brace, bracket, or parenthis matching the one the insertion point follows. Edit shortcuts include the cutting, copying, and deleting of various segments of text. Java shortcuts only apply to Java files and include Java code completion and applying Java prefixes to identifiers.

Navigation shortcuts

Table 17: Shortcuts with standard navigation keys

Keyboard shortcut	With no selected text	With selected text
RIGHT	Move the insertion point one character to the right.	Deselect the text and move the insertion point one character to the right.
LEFT	Move the insertion point one character to the left.	Deselect the text and move the insertion point one character to the left.
DOWN	Move the insertion point to the next line.	Deselect the text and move the insertion point to the next line.
UP	Move the insertion point to the previous line.	Deselect the text and move the insertion point to the previous line.
SHIFT+RIGHT	Select the character to the right of the insertion point.	Extend the selection one character to the right.
SHIFT+LEFT	Select the character to the left of the insertion point.	Extend the selection one character to the left.
SHIFT+DOWN	Create a text selection and extend it to the next line.	Extend the selection to the next line.
SHIFT+UP	Create a text selection and extend it to the previous line.	Extend the selection to the previous line.

Keyboard shortcut	With no selected text	With selected text
CTRL+RIGHT	Move the insertion point one word to the right.	Deselect the text and move the insertion point one word to the right.
CTRL+LEFT	Move the insertion point one word to the left.	Deselect the text and move the insertion point one word to the left.
CTRL+SHIFT +RIGHT	Create a text selection and extend it one word to the right.	Extend the selection one word to the right.
CTRL+SHIFT +LEFT	Create a text selection and extend it one word to the left.	Extend the selection one word to the left.
PgDown	Move the insertion point one page down.	Deselect the text and move the insertion point one page down.
PgUp	Move the insertion point one page up.	Deselect the text and move the insertion point one page up.
SHIFT+ PgDown	Create a text selection and extend it one page down.	Extend the selection one page down.
SHIFT+PgUp	Create a text selection and extend it one page up.	Extend the selection one page up.
HOME	Move the insertion point to the beginning of the line.	Deselect the text and move the insertion point to the beginning of the line.
END	Move the insertion point to the end of the line.	Deselect the text and move the insertion point to the end of the line.
SHIFT+HOME	Create a text selection and extend it to the beginning of the line.	Extend the selection to the beginning of the line.
SHIFT+END	Create a text selection and extend it to the end of the line.	Extend the selection to the end of the line.

Keyboard shortcut	With no selected text	With selected text
CTRL+HOME	Move the insertion point to the beginning of the document.	Deselect the text and move the insertion point to the beginning of the document.
CTRL+END	Move the insertion point to the end of the document.	Deselect the text and move the insertion point to the end of the document.
CTRL+SHIFT +HOME	Create a text selection and extend it to the beginning of the document.	Extend the selection to the beginning of the document.
CTRL+SHIFT +END	Create a text selection and extend it to the end of the document.	Extend the selection to the end of the document.
ALT+e	Move the insertion point to the end of the current word.	Move the insertion point to the end of the last word in the selection.

Table 18: Insertion point/screen position shortcuts

Press	To
CTRL+UP	Scroll the text one line up while holding the insertion point in the same position.
CTRL+DOWN	Scroll the text one line down while holding the insertion point in the same position.
ALT+t	Scroll the text up so that the insertion point moves to the top of the window while remaining at the same point in the text.
ALT+m	Scroll the text so that the insertion point moves to the middle of the window while remaining at the same point in the text.
ALT+b	Scroll the text down so that the insertion point moves to the bottom of the window while remaining at the same point in the text.
SHIFT+ALT+t	Move the insertion point to the top of the window.

Press	To
SHIFT+ALT+m	Move the insertion point to the middle of the window.
SHIFT+ALT+b	Move the insertion point to the bottom of the window.

Table 19: Jump list shortcuts

Press	To
ALT+k	Go to previous entry in the jump list.
ALT+l	Go to next entry in the jump list.
SHIFT+ALT+k	Go to the previous jump list entry not in the same file.
SHIFT+ALT+l	Go to the next jump list entry not in the same file.

Table 20: Miscellaneous navigation shortcuts

Press	To
F2	Go to next bookmark.
CTRL+F2	Toggle bookmark.
CTRL+b	Find matching bracket.
CTRL+SHIFT+b	Select block between current bracket and matching one.

Table 21: Find shortcuts

Keyboard shortcut	With no selected text	With selected text
CTRL+f	Show Find dialog.	Show Find dialog and show selected text as the text to find.
CTRL+r	Show Replace dialog.	Show Replace dialog and show selected text as the text to find.
F3	Search for the next occurrence.	
SHIFT+F3	Search for the previous occurrence.	

Keyboard shortcut	With no selected text	With selected text
CTRL+F3	Search for the next occurrence of the word that the insertion point is on.	Search for the next occurrence of the selected text.
ALT+SHIFT+h	Switch highlight search on or off.	
CTRL+g	Show Goto Line dialog.	

Edit shortcuts

Table 22: Standard Edit shortcuts

Press	To
INSERT	Switch between insert mode and overwrite mode.
CTRL+a	Select all.
CTRL+y	Undo the previous command.
CTRL+z	Redo the undone command.
CTRL+x	Delete the selected text from the file and copy it the clipboard.
SHIFT+DELETE	Delete the selected text from the file and copy it the clipboard.
CTRL+c	Copy the selected text to the clipboard.
CTRL+INSERT	Copy the selected text to the clipboard.
CTRL+v	Insert the clipboard text into the file.
SHIFT+INSERT	Insert the clipboard text into the file.
CTRL+e	Remove the current line.
CTRL+h	Remove character preceding the insertion point.
CTRL+u	Delete text in the following cycle (when using the shortcut successive times): first the text preceding the insertion point on the same line, then the indentation on the line, then the line break, then the text on the previous line, and so on.

Press	To
CTRL+w	Remove the current word or the word preceding the insertion point
CTRL+k	Word Match - find the previous word that begins like the current word and complete the current word so that they match
CTRL+l	Word Match - find the next word that begins like the current word and complete the current word so that they match
DELETE	Remove character after the insertion point
SHIFT+SPACE	Insert space without expanding abbreviation
ALT+j	Select the word the insertion point is on or deselect any selected text

Table 23: Indentation shortcuts

Keyboard shortcut	With no selected text	With selected text
TAB	Insert tab	Shift selection right
SHIFT+TAB		Shift selection left
CTRL+t	Shift line right	Shift selection right
CTRL+d	Shift line left	Shift selection left
Alt+f	Reformat the entire file	Reformat the selected lines

Table 24: Capitalization shortcuts

Keyboard shortcut	With no selected text	With selected text
ALT+u then u	Make the character after the insertion point uppercase	Make the selection uppercase
ALT+u then l	Make the character after the insertion point lowercase	Make the selection lowercase

Keyboard shortcut	With no selected text	With selected text
ALT+u then r	Reverse the case of the character after the insertion point	Reverse the case of the selected text
ALT+u then f	Reverse the case of the first character of the identifier the insertion point is on.	Reverse the case of the first character of the selected identifier
Alt+f	Reformat the entire file	Reformat the selected lines

Special Java Shortcuts

Table 25: Java-only shortcuts

Press	To
CTRL+SPACE	Open the Java code completion list box with matching entries. (Press TAB to enter the longest common substring. Press ENTER to complete the word and close the list box.)
ALT+u then g	Prefix the identifier with `get`
ALT+u then s	Prefix the identifier with `set`
ALT+u then i	Prefix the identifier with `is`
ALT+g	Go to the variable declaration. For this shortcut to work properly, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source.
ALT+o	Open the source based on where the insertion point is. For this shortcut to work properly, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source.

Press	To
ALT+F1	Open the web browser on the Javadoc file pertaining to the item that the insertion point is on. For this shortcut to work properly, the source file must be in a directory mounted in Filesystems, and the parser database must be updated for that source. In addition, the Javadoc documentation for that source must be mounted under the **Javadoc** tab in the Explorer.
CTRL+SHIFT+t	Comment out the current line or selected lines of code.
CTRL+SHIFT+d	Remove comment from the current line or selected lines.

Appendix B

Default Editor Abbreviations

Editor abbreviations can be defined by opening the Global Options window, expanding the `Editor Settings` node, selecting the subnode for the type of Editor, selecting the `Abbreviations` property, and opening its custom property editor. You can add, delete, and change abbreviations. The following tables list the default Java Editor abbreviations and JSP Editor abbreviations.

Table 26: Java Editor abbreviations

Abbreviation	*Expands To*
En	Enumeration
Ex	Exception
Gr	Graphics
Ob	Object

Abbreviation	Expands To
Psf	public static final
Psfb	public static final boolean
Psfi	public static final int
Psfs	public static final String
Re	Rectangle
St	String
Ve	Vector
ab	abstract
bo	boolean
br	break
ca	catch (
cl	class
cn	continue
de	default:
ex	extends
fa	false
fi	final
fl	float
fy	finally
ie	interface
im	implements
impS	import com.sun.java.swing.
impa	import java.awt.
impb	import java.beans.
impd	import com.netbeans.developer
impj	import java.
impq	import javax.sql

Abbreviation	Expands To
imps	import javax.swing.
impx	import.com.netbeans.developerx
iof	instanceof
ir	import
pe	protected
pr	private
psf	private static final
psfb	private static final boolean
psfi	private static final int
psfs	private static final String
pst	printStackTrace();
pu	public
re	return
serr	System.err.println ("
sh	short
sout	System.out.println ("
st	static
sw	switch (
sy	synchronized
tds	Thread.dumpStack();
th	throws
tr	transient
tw	throw
twn	throw new
twne	throw new Error()
twni	throw new InternalError();
wh	while

Table 27: JSP Editor abbreviations

Abbreviation	*Expands To*
ag	application.getValue("
ap	application.putValue("
ar	application.removeValue("
cfgi	config.getInitParameter("
jg	<jsp:getProperty name="
jspf	<jsp:forward page="
jspg	<jsp:getProperty name="
jspi	<jsp:include page="
jspp	<jsp:plugin type="
jsps	<jsp:setProperty name="
jspu	<jsp:useBean id="
oup	out.print("
oupl	out.println("
pcg	pageContext.getAttribute("
pcgn	pageContext.getAttributeNamesInScope(
pcgs	pageContext.getAttributesScope("
pcr	pageContext.removeAttribute("
pcs	pageContext.setAttribute("
pg	<%@ page
pga	<%@ page autoFlush="
pgb	<%@ page buffer="
pgc	<%@ page contentType="
pgerr	<%@ page errorPage="
pgex	<%@ page extends="
pgie	<%@ page isErrorPage="
pgim	<%@ page import="

Abbreviation	Expands To
pgin	<%@ page info="
pgit	<%@ page isThreadSafe="
pgl	<%@ page language="
pgs	<%@ page session="
rg	request.getParameter("
sg	session.getValue("
sp	session.putValue("
sr	session.removeValue("
tglb	<%@ taglib uri="

Appendix C

Main Window Menus and Toolbars

Menus

The following sections provide lists and short descriptions each of the entries on the Main Window menus.

File Menu

- **New From Template** – create a new object. The Templates folder will open, enabling a predefined template to be used as the basis for the new object. Available template categories are: AWT Forms, Classes, Dialogs, Other, and Swing Forms. See "Using templates" on page 200 for more information on templates.

- **Open Explorer** – open a new Explorer window.

- **Object Browser** – open the Object Browser window.
- **Open File** – open a file that is currently in the IDE, or mount a file system to the IDE and open a file in it.
- **Save** – save the current object. If there is more than one object currently open in the multi-tab Editor window, the source in the currently selected tab is saved. If there is more than one Editor window open (that is, one or more sources have been undocked), the object in the currently selected Editor is saved. Save is disabled when no modified objects are currently open or the selected object is unmodified.
- **Save All** – save all current unsaved open objects.
- **Save Settings** – save all current unsaved Project Settings. This command is available only when the Projects module is disabled or uninstalled (and replaces **Save Project**).
- **Print** – print the file active in the Editor or selected in the Explorer or Object Browser.
- **Exit** – exit the IDE. You will be prompted to selectively save any currently open unsaved objects, discard their changes, or cancel. Workspace configurations and other settings changes are saved.

Edit Menu

- **Undo** – undo last action.
- **Redo** – redo last action.
- **Cut** – cut selected object or text to clipboard.
- **Copy** – copy selected object or text to clipboard.
- **Paste** – paste contents of clipboard.
- **Delete** – delete selected object.
- **Find...** – find file or find text in file.
- **Replace...** – replace in text.
- **Goto...** – go to line number in Editor window.

View Menu

- **Explore from Here** – open a new Explorer, with the selected node as the root.

- **Properties** – make the Property Sheet window for the selected object the active window (or open the property sheet).
- **Explorer Window** – make the currently open Explorer the active window (or open it if it is not already open).
- **Editor Window** – make the currently open Editor the active window (or open it if it is not already open).
- **Output Window** – make the currently open Output Window the active window (or open it if it is not already open).
- **Debugger Window** – make the currently open Debugger Window the active window (or open it if it is not already open).
- **Execution Window** – make the currently open Execution Window the active window (or open it if it is not already open).
- **Web Browser** – make the currently open web browser window the active window (or open it if it is not already open).
- **Component Inspector** – make the currently open Component Inspector window the active window (or open it if it is not already open).
- **Workspaces** – list of all available workspaces as a submenu with the current workspace indicated by selected the radio button. Selecting one in the submenu switches the current workspace.

Project Menu

- **New Project...** – close the current project and create a new one.
- **Open Project...** – close the current project and open another existing project.
- **Save Project** – save the current IDE state, including the project settings. The files themselves are not saved.
- **Compile Project** – compile all of the sources currently listed under the **Project** tab in the Explorer.
- **Build Project** – force compilation of all classes currently listed under the **Project** tab in the Explorer., whether current or not.
- **Set Main Class...** – set the main class to be executed when executing the project as a whole.

- **Execute Project** – execute the project.
- **Debug Project** – start the Debugger on the main class of the project
- **Import Project...** – import a project developed in another IDE.
- **Settings** – open the Project Settings window.

Build Menu

- **Compile** – compile selected object. If a folder is selected, compile all sources in that folder that are uncompiled or have been modified since the last compilation.
- **Compile All** – recursively compile (compile all uncompiled or out-of-date files in a folder and all its sub-folders).
- **Build** – force compilation of all classes, whether current or not.
- **Build All** – recursively build; build a folder and all its subfolders.
- **Clean** – delete all `.class` files in the selected package.
- **Clean All** – recursively delete all `.class` files in the selected package and its sub-packages.
- **Stop Compilation** – stop the current compilation process.
- **Next Error** – jump to the next error in the Output Window.
- **Previous Error** – jump to the previous error in the Output Window.
- **Set Arguments** – set command line arguments to be passed to an executed application. (May also be set in the `Arguments` property under the **Execution** tab of the property sheet of the class).
- **Execute** – execute the selected object.

Debug Menu

- **Start Debugging** – initiate a debugging session and run the program.
- **Continue** – resume a debugging session that has been halted.
- **Connect** – connect the Debugger to an already running process.

- **Finish Debugging** – end the current debugging session.
- **Suspend All** – Suspend all threads in the Debugger.
- **Trace Into** – trace into the method the debugger has halted at.
- **Go To Cursor** – execute the current statement and all ensuing statements until it reaches the line that the insertion point is on.
- **Trace Over** – trace over the method the debugger has halted at.
- **Step Out** – halt execution after the current method finishes and control passes to the caller.
- **Add/Remove Breakpoint** – switch a breakpoint on or off at the current line. The line will be highlighted in blue when a breakpoint is set.
- **New Breakpoint** – add a new breakpoint. A dialog will appear prompting you to type where it should it be set.
- **Add Watch...** – watch a variable.
- **Debugger Window** – open the Debugger Window.
- **Resume All** – Resume debugging of all threads in the Debugger.

Tools Menu

- **Add Directory** – mount a new directory under the **Filesystems** tab in the Explorer.
- **Add JAR** – mount a new JAR archive as a file system under the **Filesystems** tab in the Explorer.
- **Remove Filesystem** – unmount the selected file system from the Explorer.
- **JSP Wizard** – run a wizard to create a Java Server Page.
- **Search Filesystems** – search for a file in a file system mounted in the Explorer, by file name, file text, or date.
- **Install New JavaBean** – install a new JavaBeans component to the Component Palette.
- **Tools | Create Group...** – create a group (object composed of links to one or more files, enabling you to access them from the same place in the IDE).
- **Tools | Auto Comment...** – enable you to comment your source code automatically and view all parts of your source code (methods, constructors, inner classes, variables, and so

on) and document them individually.

- **Tools | Generate Javadoc...** – generate Javadoc documentation for the selected classes or elements and place it in the directory that you specify.
- **Tools | Add to Component Palette...** – Add the selected object to the Component Palette.
- **Tools | Synchronize** – force synchronization of the selected source file with the interfaces it implements.
- **Tools | Set As Project Main Class** – set the selected project as the class to be executed when you choose the **Execute Project** command.
- **Tools | Update Parser Database...** – update the Java parser database with the classes of the selected package, thus making those classes available in addition to the standard Java 2 Platform SDK classes when using the Java code completion feature in the Editor.
- **Tools | Add to Project** – add the selected class to the project, making it subject to compilation when you choose the Compile Project command.
- **Configure Shortcuts...** – display the Configure Shortcuts dialog box in order to add, delete, or change keyboard shortcuts.
- **Global Options...** – display the Global Options window, enabling you to change various IDE settings.
- **Settings...** – display the Project Settings window. This command is available only when the Projects module is disabled or uninstalled.

Window Menu

- **Clone View** – clone the current window (opens a second view of the current window as a new tab on the multi-tab window, or in a separate window if the original window is not tabbed).
- **Undock Window** – undock the current window from the parent multi-tab window. Opens in a completely separate and independent multi-tab window.
- **Dock Into...** – dock the selected single-pane window or the current pane of the selected multi-tab window to one of the windows listed in the submenu.
- **Next Tab** – flip to the next tab in the multi-tab window.
- **Previous Tab** – flip to the previous tab in the multi-tab window.
- **Windows** – open a submenu of workspaces, each of which has a submenu of windows in

that workspace. If you choose one of these windows, it will open in the current workspace.

Help Menu

- **Documentation** – view the Forte for Java User's Guide in the JavaHelp viewer. If extension modules with their own documentation are installed in the IDE, this menu item will have a submenu that enables you to choose which documentation to view.
- **Javadoc Index Search** – open the Javadoc Search Tool window to conduct a search on Javadoc directories mounted in the Explorer.
- **Getting Started** – open the "Forte for Java (Community Edition) QuickStart Guide" in the web browser.
- **Tutorial** – open the Forte for Java tutorials in the web browser.
- **Forte For Java Home on the Web** – open the web browser and connect to the Forte for Java web site at http://www.netbeans.com/ If you are using a machine which connects to the internet through a dial-up modem, this may mean your modem attempts to dial and connect to your ISP. This depends on your local system configuration.
- **Forte For Early Access Program** – open the web browser and connect to the web site for the Forte for Java's Early Access Program.
- **Forte For Java Open APIs On-line** – open the web browser and connect to the web site for the Forte for Java's Open APIs On-line.
- **Bookmarks** – display a submenu with bookmarks. If you select a bookmark, the web page connected with it will open in the web browser.
- **Tip of the Day...** – display the Tip of the Day dialog which opens by default when you run Forte for Java.
- **Update Center** – connect with the Forte for Java web site to automatically install new or updated modules to your IDE.
- **About** – display the About Forte for Java, Community Edition dialog.

Toolbars

There are toolbar items for many of the menu commands as well. You can identify the commands for each toolbar item by holding the mouse over the icon to bring up its tool tip

label.

Appendix D

Reference Guide to Project Settings and Global Options

Filesystems Settings reference

The following settings are available for each file system under the `Filesystems Settings` node (the last four appear under the **Capabilities** tab).

Table 28: File System properties

Property	*Description*
Extensions of Backup Files	Extensions of files that should be backed up during modifications. This option is only available when the file system is mounted using **Add Directory** from the **Tools** menu or the context menu for the `Filesystems` node in the Explorer. Backup files will be named with a tilde (~).
Archive File (for archive files only)	The mounted JAR or ZIP archive. You can use the custom property editor to change which JAR is mounted.
Hidden	If `True`, the file system is not displayed under the **Filesystems** tab in the Explorer.
Read Only	If `True`, you may not modify any files in the file system.
Root Directory (for directory-based file systems only)	The root directory of the mounted file system (should be the top of the package hierarchy). You can use the custom property editor to change which file system is mounted.
Valid	If `True`, the file system is valid and usable.
Use in Compiler	If `True`, files in the directory should be included in the compiler's source path.
Use in Debugger	If `True`, files in the directory should be made available when debugging.
Use as Documentation	If `True`, files in the directory are made available for Javadoc index searches (via **Help | Javadoc Index Search** or the CTRL+F1 shortcut) and for access from the Editor using the ALT+F1 shortcut.
Use for Execution	If `True`, files in the directory should be made available (meaning put in the class path) when executing.

Workspaces reference

Of the following properties, the first two apply to workspace nodes, and the remaining apply to the mode (meaning the programmatic construct that allows several different types of windows to appear in a multi-tab window).

Table 29: Workspace properties

Property	Description
Name	Name of the workspace.
Toolbar Configuration	Choice of available toolbar configurations to display for the workspace.
Bounds	Set the X and Y coordinates of the top left corner and the height and width of the window.
Display Name	Display the name of the mode.
Name	Programmatic name of the mode.
Visible	If `True`, the window is currently visible (if the workspace is open).

Compiler Types reference

Table 30: External Java Compiler service properties

Property	Description
Debug	If `True`, the compiler produces code with debug information. Must be turned on to debug with source tracking.
Deprecation	If `True`, the compiler treats the use of deprecated methods as errors.
Encoding	Character encoding is used in Java sources.
Error Expression	A regular expression in POSIX format describing the format of the error output of the specified compiler. You can also define a custom expression, if using a different compiler, and save that setting. Select one of the pre-defined sets for error descriptors: `Sun javac`, `Microsoft jvc`, or `IBM jikes + E`.
External Compiler	Path to executable compiler. You can use the custom property editor to define the process and arguments, using a set of substitution codes.

Property	Description
Identifying Name	The name for the compiler type by which classes reference it.
Optimize	If `True`, the compiler optimizes generated bytecode.
Debug tag replace	The command-line option to include debugging information (for example, `-g`).
Deprecation tag replace	The command-line option to show deprecations (for example, `-deprecation`).
Optimize tag replace	The command line option to optimize (for example, `-O`).

Table 31: Internal Java Compiler service properties

Property	Description
Debug	If `True`, the compiler produces code with debug information. Must be turned on to debug with source tracking.
Deprecation	If `True`, the compiler treats the use of deprecated methods as errors.
Identifying Name	The name for the compiler type by which classes reference it.
Encoding	If `True`, character encoding is used in Java sources.
Optimize	If `True`, the compiler optimizes generated bytecode.

See http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javac.html for more information on Javac.

Execution Types reference

Table 32: External Execution service properties

Property	Description
External Process	Path to Java launcher. You can use the custom property editor to define the process and arguments, using a set of substitution codes. See "Process Descriptor property editor" on page 206.
Identifying Name	The name for the execution type by which classes reference it.
Boot Class Path	Boot class path of Java, including installed extensions.
Class Path	The startup class path used by the IDE that is transferred to the external process.
Environment variables	A list of environment variables in the form `Name=Value`. If the value is `null`, the IDE's default settings are used.
Filesystems Path	Read-only value showing the current class path from Filesystems.
Library Path	List of modules and special libraries the IDE uses.
Working Directory	A working directory to start the process in. If the value is `null`, the IDE's own working directory is used.

Table 33: Applet Execution service properties

Property	Description
External viewer	Path to Java launcher. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Identifying Name	The name for the applet execution type by which classes reference it.

Table 34: JSP Execution service properties

Property	Description
Allow unregistered servlets	When `False`, links to servlets in the form *`URL/servlet/package.ClassName`* are disallowed.
External Process	Path to Java launcher. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Identifying Name	The name for the applet execution type by which classes reference it.
MIME types	Specifies mapping of file extensions to MIME types.
Port	Port on which the server runs.
Welcome files	Files to look for when there is a request to a directory.
Boot Class Path	Boot class path of Java, including installed extensions.
Class Path	The class path used by the IDE that is transferred to the external process.
Environment Variables	A list of environment variables in the form *`Name=Value`*. If the value is `null`, the IDE's default settings are used.
Filesystems Path	Read-only value showing the current class path from Filesystems.
Library Path	List of modules and special libraries the IDE uses.
Working Directory	A working directory to start the process in. If the value is `null`, the IDE's own working directory is used.

Debugger Types reference

Table 35: Applet Debugging and Default Debugging properties

Property	*Description*
Classic Switch	Set to `True` if HotSpot is installed.
External Process	Path to the debugger. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Identifying Name	The name for the debugger type by which classes reference it.
Boot Class Path	Boot class path of Java, including installed extensions.
Class Path	The class path used by the IDE that is transferred to the external process.
Library Path	List of modules and special libraries the IDE uses.
Filesystems Path	Read-only value showing the current class path from Filesystems.

Search Types reference

Table 36: Search Types properties

Property	*Description*
Identifying Name	The name for the search criterion, by which you can access it in the Customize Criteria dialog box.
Regular Expression	A POSIX-style regular expression which can be used as a search criterion.
Substring	A literal string of text to be used as a search criterion.
Modified After	The search will match files that were last modified on or after the date entered here.
Modified Before	The search will match files that were last modified on or before the date entered here.

Property	Description
Modified Within Previous Days	The search will match files that were last modified within the previous (number entered here) days.

Java Sources settings reference

Table 37: Java Sources properties

Property	Description
Automatic parsing delay	Parser auto-start time-out in milliseconds.
Default compiler type	The compiler type that serves as the default for new sources.
Default debugger type	The debugger type that serves as the default for new sources.
Default executor	The execution type that serves as the default for new sources.
Parse class files	If `True`, class files are parsed for their Source property. This is useful if you are using sources that have package private top-level classes (Java 1.0-style) and you want such class files to be correctly associated to the source file. However, setting this property to `True` slows down the IDE.
Strings table	Table of substitution keys for templates.

Source synchronization

Table 38: Source synchronization properties

Property	*Description*
Return generation mode	Generate either nothing, an exception, or the string "return null" when creating a new method declared to return a value.
Synchronization enabled	If `False`, all synchronization is turned off.

Object Browser settings reference

Table 39: Object Browser properties

Property	*Description*
Package Filter	Custom property editor available to add, delete, and rename package filters to the Object Browser.

Project Options

Table 40: Project Options

Property	*Description*
Add to project	If set to `Ask`, each time you create a class from template, you will be prompted whether or not to add the class to the current project. If set to `Never`, you will not be prompted, and the class will not be added to the current project when you create it. If set to `Always`, you will not be prompted, and the class will automatically be added to the current project when you create it.

Debugger Settings reference

Table 41: Debugger properties

Property	*Description*
Action on Trace Into	The action that will occur when you use the Trace Into command on a method without available source. Choose between `Trace Out` and `Stop`.
Run Compilation	Compile sources before debugging.
Workspace	The name of the workspace to which the IDE switches when the debugger starts, or None if the workspace should not be switched.

Documentation settings reference

Table 42: Internal Javadoc options

Option	*Description*
1.1 style	If `True`, comments are generated in JDK 1.1 style.
Encoding	The source file encoding name such as `EUSJIS/SJIS`. If this option is not specified, the platform default converter is used.
Extdirs	The directories where extension classes reside.
Locale	The locale that Javadoc uses when generating documentation. Use the name of the locale, as described in `java.util.Locale` documentation, such as `en_US` (English, United States).
Members	Determines which members will be included in the documentation. If `public`, shows only public classes and members. If `protected`, shows only protected and public classes and members. If `package`, shows only package, protected, and public classes and members. If `private`, shows all classes and members.

Option	Description
Overview	File containing text to be used on the Overview page (`overview-summary.html`).
Verbose	If True, Javadoc provides more detailed messages when running.

Table 43: Standard doclet options

Option	Description
Author	If `True`, includes the `@author` tag in the documentation.
Bottom	The HTML text to be placed at the bottom of each output file.
Charset	Character set for cross-platform viewing.
Destination	Destination directory for the output files.
Doctitle	The title to be placed near the top of the overview summary file.
Footer	The footer text to be placed at the bottom of each output file.
Group	Enables you to specify groupings of packages on the overview page. Enter in the format `"groupheading" "packagepattern"`. You can use an asterisk (*) as a wildcard, and you can include multiple items in `"packagepattern"` using a colon (:) to delimit them.
Header	The header text to be placed at the top of each output file.
Helpfile	The path of an alternate help file `path/filename` that the HELP link in the top and bottom navigation bars link to. Without this option, Javadoc automatically creates a help file `help-doc.html` that is hard-coded in Javadoc.
Link	A URL link to already existing javadoc-generated documentation of external referenced classes.
No Deprecated	If `True`, prevents the generation of any deprecated API at all in the documentation.
No Deprecated List	If `True`, prevents the generation of the file containing the list of deprecated APIs (`deprecated-list.html`) and the link in the navigation bar to that page.

Option	Description
No Help	If `True`, omits the HELP link in the navigation bars at the top and bottom of each page of output.
No Index	If `True`, omits the HELP link in the navigation bars at the top and bottom of each page of output.
No Navbar	If `True`, prevents the generation of the navigation bar, header and footer, otherwise found at the top and bottom of the generated pages.
No Tree	If `True`, omits the class/interface hierarchy from the generated docs. The hierarchy is produced by default.
Split Index	If `True`, splits the index file into multiple files, alphabetically, one file per letter, plus a file for any index entries that start with non-alphabetical characters.
Stylesheetfile	Path to an alternate HTML style sheet file. Without this option, Javadoc automatically creates a style sheet file `stylesheet.css` that is hard-coded in Javadoc.
Use	If `True`, includes one "Use" page for each documented class and package. The page describes what packages, classes, methods, constructors and fields use any API of the given class or package.
Version	If `True`, includes the `@version` text in the generated docs.
Window Title	The title to be placed in the HTML `<title>` tag.

Editor Settings reference

There are both global Editor settings and settings for each individual type of editor.

Global Editor settings

This setting applies to editing of all types of files (for example, Java, HTML, plain text, and JSP) in the Editor.

Table 44: Global Editor Settings property

Property	Description
Global Key Bindings	Enables you to use the custom property editor for key bindings to set the key combinations for the available shortcuts. These do not include the keyboard shortcuts that can be configured separately for each type of editor or which apply to the IDE as a whole.

Editor settings by type of editor

Most Editor Settings are divided by type of editor – Plain, HTML, Java and JSP editors in Forte for Java, Community Edition (with other types added by extension modules). Even though the different editors have many of the same properties in common, most of the properties for the different types of editor are configured separately. The only exception is Global Key Bindings which is configured globally for all types of editors.

Table 45: Standard properties (available separately for each editor type)

Property	Description
Abbreviations	Enables you to use the Abbreviations (Map) custom property editor to set abbreviations that the Editor will automatically expand to longer strings when you type them.
Expand tabs to spaces	If `True`, tabs in the document are converted to spaces padded to the same column.
Font Size	If you change this property, the font for all tokens will be set to that size, regardless of sizes set in the `Fonts and Colors` property.
Fonts and Colors	Provides access to a custom property editor where background and foreground colors as well as fonts can be set for various syntax tokens.
Key Bindings	Clicking on the property and then clicking on the **...** button brings up the Key Bindings property editor, which enables you to add, remove, and edit key bindings.
Line Numbers	If `True`, the lines are numbered.
Number of Spaces per Tab	Number of spaces a block is indented when you enter TAB.

Property	Description
Tab size	Number of spaces between each tab stop in the editor. The tab stops are used when importing files into the editor.

Table 46: Expert properties (available separately for each editor type)

Property	Description
Caret Blink Rate	Rate in milliseconds that the insertion point (caret) blinks.
Display Text Limit Line	If `True`, shows a vertical line to mark suggested maximum line width (for example, for printing).
Highlight Caret Row	If `True`, the row where the insertion point is highlighted.
Highlight Matching Bracket	If `True`, whenever the insertion point is immediately after a brace, bracket, or parenthesis, the matching brace/bracket/parenthesis is highlighted.
Insert Caret	The type of insertion point that appears when in insert mode. From the drop-down list you can choose between line, thin line, and block.
Insert Caret Color	Choose a color by clicking on the property and then either selecting a color from the drop-down list or invoking the custom property editor for colors by pressing **...**.
Italic Insert Caret	If `True`, insertion point is italic when in insert mode.
Italic Overwrite Caret	If `True`, insertion point is italic when in overwrite mode.
Line Height Correction	Multiplier to adjust height of lines.
Line Number Margin	Brings up a custom property editor to set up placement of line numbers.
Margin	Brings up a custom property editor to set top, bottom, left, and right margins.
Overwrite caret	The type of insertion point that appears when in overwrite mode. From the drop-down list you can choose between line, thin line, and block.
Overwrite caret color	Insertion point color when in overwrite mode.
Scroll Find Insets	Specify how much space should be reserved on each side of text located with the Find command.

Property	Description
Scroll Jump Insets	Specify for each four directions how much the view should jump when the scrolling goes off of the screen.
Status Bar Caret Delay	The delay in milliseconds between the time the insertion point stops moving and its position is updated in the status bar.
Status Bar Visible	If True, the status bar (which shows information such as current line number, whether the Editor is in insert or over-write mode, and so on) is displayed at the bottom of the window.
Text Limit Character Count	The number of characters right of the left margin that the text limit line is displayed.
Text Limit Line Color	The color of the text limit line.

Table 47: Java-only Editor Settings

Property	Description
Add Space before Parenthesis	If `True`, a space is added before the opening parenthesis in the generated code.
Auto Popup of Java Completion	If `True`, the Java code completion box automatically appears when appropriate.
Curly Brace on Next Line	If `True`, the curly braces generated automatically are put on the line following the previous code.
Delay of Java Auto Completion Popup	Delay in milliseconds before the Java Completion popup appears.

Execution Settings reference

Table 48: Execution properties

Property	*Description*
Clear Output Tab	If `True`, output is cleared from the Output Window before reuse.
Reuse Output Tab	If `False`, the IDE creates new tabs in the Output Window for each execution.
Run Compilation	If `True`, the IDE first compiles before executing a file.
Workspace	The window to activate when executing a file (Running, Browsing, Editing, Debugging, or None).

Form Objects reference

The last six properties on this table are listed under the **Expert** tab on the property sheet.

Table 49: Form Objects properties

Property	*Description*
Event Variable Name	The name of the variable generated in the signature of the event handler method for the event object. For example, `evt` is the variable name in `private void button1ActionPerformed (java.awt.event.ActionEvent evt)`.
Generate Null Layout	If `True`, forms using the Absolute Layout manager will generate null layouts instead of Absolute layouts.
Grid X	Size of grid for AbsoluteLayout in the X axis.
Grid Y	Size of grid for AbsoluteLayout in the Y axis.
Indent AWT Hierarchy	If `True`, the code generated in `initComponents ()` is indented for child components of a container.

Property	Description
Output Detail Level	Determines how detailed information written to the Output Window is. Choice between `Normal`, `Minimum`, and `Maximum`.
Property Editor Search Path	List of packages that are searched for property editors that are to be used in the Form Editor.
Property Editors	Explicitly registered editors for certain property types.
Show Grid	If `True`, a grid is displayed in the Form Editor when using AbsoluteLayout.
Variables Modifier	The access modifier of variables generated for components on the form (private, package private, protected, or public).
Workspace	The workspace the IDE jumps to when a form object is opened.
Apply Grid to Position	If `True`, the position of components is snapped to grid (if a grid is used).
Apply Grid to Size	If `True`, the size of components is snapped to grid (if a grid is used).
Connection Border Color	Color of components' selection border during connection mode.
Drag Border Color	Color of components' drag border during dragging.
Selection Border Color	Color of the boxes around a component which mark it as "selected".
Selection Border Size	Size (in pixels) of the boxes around a component which mark it as "selected".

HTML Browser reference

Table 50: HTML Browser properties

Property	*Description*
Default Background	Background color of HTML pages.
Fixed font	The base fixed-width font used in HTML pages.
Proportional Font	The base proportional font used in HTML pages.

HTTP Server reference

The last two properties in the following table can be found under the **Expert** tab of the property sheet.

Table 51: HTTP Server properties

Property	*Description*
Grant access to	Specifies machines which enable access to the HTTP server. Machines are entered as a comma-separated list of IP addresses.
Host	Host has two possible settings: `Any Host` or `Selected Hosts`. `Selected Hosts` restricts access so that only the machine on which Forte for Java is running and machines specified in `Grant access to` are allowed access. (default=`Selected Hosts`).
Port	The port number on which the HTTP server operates. (default=8081).
Running	If `True`, the HTTP server is running.
Base Class Path URL	Gives access to internal IDE resources.
Base Filesystems URL	Gives access to the contents of the **Filesystems** tab of the Explorer.

JSP and Servlets reference

Table 52: Servlets and JSP

Property	*Description*
External browser	Path to the external web browser.
Web browser	Choice between using the default web browser and the external web browser specified in the `External browser` property.

Java Elements reference

Table 53: Java element properties

Property	*Description*
Classes	Display name of classes (using a combination of plain text and substitution codes).
Constructors	Display name of constructors (using a combination of plain text and substitution codes).
Fields	Display name of fields (using a combination of plain text and substitution codes).
Initializers	Display name of initializers (using a combination of plain text and substitution codes).
Interfaces	Display name of interfaces (using a combination of plain text and substitution codes).
Methods	Display name of methods (using a combination of plain text and substitution codes).

Table 54: Substitution Codes for the Java Elements properties

Substitution Code	Type of substituted text
{m}	Element's modifiers (for all elements except initializers).
{n}	Element's name (for all elements except initializers).
{C}	Name of class with all outerclasses (for classes and interfaces only).
{f}	Full name of element with package (for all elements except initializers).
{t}	Type (for fields only).
{r}	Return type (for methods and constructors).
{s}	Superclass (for classes only).
{c}	Static (for initializers only).
{p}	Parameters with types but not variable names (for constructors and methods).
{a}	Parameters with types *and* names (for constructors and methods).
{i}	Interfaces implemented or extended (for classes and interfaces only).
{e}	Exceptions (for constructors and methods only).
<initializer>	Initializer.

Advanced substitution formats

You can also use advanced substitution formats with these substitution codes to create a more informative display name for the elements in the Explorer and Object Browser.

For simple arguments (arguments that do not contain lists of items), you can create a code in the format:

`{SubstitutionCode,prefix,suffix}`

where ***prefix*** represents a string to appear before the element name and ***suffix*** represents a string to appear after it. If there is nothing to replace the substitution code with, the prefix and suffix parameters are ignored. If you want to use a comma in the prefix or suffix, enclose the string in double quote (") marks.

For example, to display information about exceptions on method nodes, you could use the expression

```
{n}{e, throws ,}
```

If the method does not contain exception code, only the method name is displayed. If there is exception code, the following is displayed:

methodName throws *exceptionName*

You can display the `{p}`, `{a}`, `{i}`, and `{e}` codes in the following array format:

`{`*SubstitutionCode*`,`*prefix*`,`*suffix*`,`*delimiter*`}`

where *delimiter* represents the text that separates the parameters.

For example, for `Methods`, you could enter

```
{n}{p, (,),", "}
```

to display the method name and then the parameters of the method in parentheses. If there is more than one parameter, the parameters will be delimited with commas. If there are no parameters, only the method name will be displayed.

Open File Server reference

Table 55: Open File Server properties

Property	*Description*
Access Restriction	If set to `Any Host`, people on other machines can open files in your IDE. Set to `Local Host Only` by default.
Port	The port the server runs on - by default 7318.
Running	If `True`, the server is on.

Output Window reference

Table 56: Output Window properties

Property	*Description*
Background	Background color of the Output Window.
Cursor Background	Background color of highlighted text.
Cursor Foreground	Color of highlighted text.
Font Size	Size of the characters in the Output Window.
Foreground	Default text color.
Jump Cursor Foreground	Text color for lines of text in the Output Window that are linked to lines in the Editor.
Jump Cursor Background	Background color for lines of text in the Output Window that are linked to lines in the Editor.
Tab Size	The number of spaces represented by a TAB stroke.

Print Settings reference

Table 57: Print Settings properties – general

Property	*Description*
Line Ascent Correction	A multiplier to adjust the spacing between lines.
Page Footer Alignment	Options: LEFT, CENTER, and RIGHT.
Page Footer Font	Has a custom property editor available enabling you to set font face, style, and size.
Page Footer Format	You can set the footer with a combination of text and the following tags – {0} for page number, {1} for date, and {2} for file name.
Page Header Alignment	Options: LEFT, CENTER, and RIGHT.
Page Header Font	Has a custom property editor available enabling you to set font face, style, and size.
Page Header Format	You can set the header with a combination of text and the following tags – {0} for page number, {1} for date, and {2} for file name.
Wrap Lines	If `True`, lines are wrapped.

Table 58: Print Settings properties – by individual Editor type

Property	*Description*
Print Fonts and Colors	Provides access to a custom property editor where background and foreground colors as well as fonts can be set for various syntax tokens.
Print line numbers	If `True`, line numbers appear in printouts.

Property Sheet reference

Table 59: Property Sheet properties

Property	*Description*
Disabled Property Color	The text color of read-only properties.
Display Editable Only	Whether to show only properties that are editable or that have a custom property editor.
Painting Style	Indicates if values are shown always as text, preferably as text, or preferably as graphics.
Plastic	Whether to enable animation of buttons in the property sheet to make the active property more visible.
Sorting Mode	Sets the criteria for sorting properties on the property sheet – by name, by type, or unsorted.
Value Color	The text color of property values in the property sheet.

System Settings reference

Table 60: System Settings

Property	*Description*
Confirm Delete	If `True`, a dialog appears to confirm any deletions you make.
Home Page	Home page for the internal web browser.
Proxy Host	Host of the proxy server.
Proxy Port	Port number of the proxy server.
Show Tips on Start-tup	If `True`, a dialog box showing a user tip will appear each time you start the IDE.
Use Proxy	If `True`, the proxy server is used.

Update Center reference

Table 61: Update Center

Property	*Description*
Ask Before Check	If `True`, you are prompted before the IDE checks for updates.
Check Period	Frequency that the IDE checks for updates.
Last Check	Date of the last check.
Registration Number	Update Center registration number.
Show Negative Results	If `True`, the results of the update inquiry are shown even if there are no modules to be updated.

Appendix E

Actions

This table lists and briefly describes each of the actions available in the actions pool under `Global Options / Actions`.

Table 62: Build Actions

Action	*Description*
Build	Force compilation of all objects in selected folder, whether current or not.
Build All	Build selected folder and all sub-folders, recursively.
Clean	Delete all `.class` files in the selected package.
Clean All	Recursively delete all `.class` files in the selected package and its sub-packages.
Compile	Compile the selected object.
Compile All	Compile the selected folder and all sub-folders recursively.

Action	Description
Execute	Execute the selected object.
Execute (restart server)	Execute the selected JSP with a restart of the server (the server is not automatically restarted when you re-execute a JSP).
Generate Javadoc	Generate Javadoc documentation for the selected class or package.
NextError	Jump to the next error in the Output Window.
PreviousError	Jump to the previous error in the Output Window.
Set Arguments	Set command line arguments to pass to an application.
Set Request Parameters	Specify query parameters for a JSP to be passed in the request from the browser to the server.
StopCompile	Halt the current compilation process.

Table 63: Debugger Actions

Action	Description
Add Watch	Add a watch.
Add-Remove Breakpoint	Add a new breakpoint at the current line in the Editor or remove the breakpoint at the current line.
Connect	Connect Debugger to a process in an already running virtual machine.
Debugger Window	Make the Debugger Window the active window.
Finish Debugger	Terminate debugging session.
Go To Cursor	Go to the current line in the Editor.
New Breakpoint	Add a new breakpoint.
Resume All	Resume debugging of the selected threads.
Start Debugger	Start debugging session.
Suspend All	Suspend the selected threads in the Debugger.
Trace Into	Trace into the method the debugger has halted at.

Action	Description
Trace Out	Halt execution after the current method finishes and control passes to the caller.
Trace Over	Trace over the method the debugger has halted at.

Table 64: Edit Actions

Action	Description
Copy	Copy selected object to the clipboard.
Cut	Cut the selected object, keeping a copy in the clipboard.
Delete	Delete the selected object.
Find	Find specified text in Editor.
Goto	Go to specified line number in the Editor.
Paste	Paste from the clipboard.
Redo	Redo undone action.
Replace	Replace in text.
Undo	Undo last action.

Table 65: Form Actions

Action	Description
Add to Component Palette	Add the selected class to the Component Palette.
Component Inspector	Go to the selected component in the Component Inspector.
Component Palette	Install the Component Palette.
Customize Layout	Bring up the customizer dialog for GridBag Layout.
Design Mode	Put the Form Editor in design mode.
Events	List events for selected component.
Goto Form	In the Form Editor window, go to the selected component.

Action	Description
Goto Inspector	In the Component Inspector, go to the node for the selected component.
Goto Source	Go to the line in the Editor corresponding to the selected component.
Install New JavaBean	Install a JavaBeans component into the Component Palette.
Set Layout	Set the layout.
Show Grid	Display a grid in the Form Editor window (for Absolute layout).
Test Mode	Put the Form Editor in test mode.

Table 66: Help Actions

Action	Description
About	Display the About dialog box.
Bookmarks	Show the bookmarks submenu.
Documentation	Open up the available documentation, including the Forte for Java User's Guide, in the internal web browser.
Forte4J EAP	Connect to the Forte for Java Early Access Program web site.
Forte4J Home	Connect to the Forte for Java home page.
Forte4J Open APIs	Connect to the Forte for Java Open APIs web site.
Getting Started	Open the Forte for Java tutorial in the internal web browser.
Help	Show the context help for the item that the cursor is pointing to.
Javadoc Index Search	Conduct a search the Javadoc documentation mounted in Filesystems.
Tip Of The Day	Display the Tip of the Day dialog box.
Tutorial	Open the Forte for Java tutorials in the internal web browser.

Action	Description
Update Center	Connect to the Update Center and check for new and updated modules.

Table 67: Project Actions

Action	Description
Add New Class To Project	Add a new class to the current project.
Add Existing Class To Project	Add an existing class to the current project.
Add To Project	Add the selected class to the current project.
Build Project	Force compilation of all objects in the current project.
Compile Project	Compile all objects in the project, whether current or not.
Debug Project	Start the Debugger on the main class of the project.
Execute Project	Execute the main class of the project.
Import Project	Import a project into the IDE.
New Project	Create a new project.
Open Project	Open an existing project.
Save Project	Save the project settings and current IDE state into the current project file.
Set As Project Main Class	Designate the selected class as the class to be executed when the Execute Project action is called.
Set Main Class	Designate the class to be executed when the Execute Project action is called.
Settings	Open the Project Settings window.

Table 68: System Actions

Action	Description
Add Directory	Mount a new file system under Filesystems.
Add JAR Archive	Mount a JAR archive under Filesystems.

Action	Description
Configure Shortcuts	Configure general IDE keyboard shortcuts.
CustomizeBean	Customize properties of a JavaBeans component.
Edit	For a file that has a visual and textual editing mode, open the text editor.
Exit	Exit the IDE.
Filesystem Action	Submenu of operations specific to a file system.
Garbage Collect	Garbage Collect. By default, this item is not installed in any menus or toolbars.
Global Options	Open the Global Options window.
Instantiate	Instantiate a template.
Move Down	Moves current item down among the parent's children.
MoveUp	Moves current item up among the parent's children.
New	Create a new object.
New From Template	Create a new object using an existing template.
Open	Open an object.
Open Explorer	Open a new instance of the Explorer.
Open File	Open a file in the IDE.
Open In New Window	Open a bookmark in a new web browser window.
Print	Print the file active in the Editor or selected in the Explorer.
Refresh	Refresh state of a component.
Remove Filesystem	Unmount a mounted file system.
Rename	Rename selected object.
Reorder	Change order of subnodes (subcomponents) of selected item (container).
Save	Save current object.
Save All	Save all open objects.

Action	Description
Save As Template	Save a copy of the object as a template in the Templates hierarchy.
Set Default Value	Set the default value for the property.
Save Settings	Save the current Project Settings. This action is only available if the Projects module has been disabled (replacing Save Project).
Settings...	Display the Project Settings window. This command is available only when the Projects module is disabled or uninstalled.
Tools	Show the Tools submenu (menu-only action).
View	View an object (for example, by launching the HTML browser).
View Servlet	View the servlet code generated from the JSP.

Table 69: Tools Actions

Action	Description
Auto Comment	Open the Auto Comment dialog box to add and edit Javadoc comments for a class.
Create Group	Create a group object.
Global Options	Open the Global Options window.
JSP Wizard	Run the JSP wizard to create a new Java Server Page.
Search Filesystems	Conduct a search of mounted file systems by date, text in file name, or text in file.
Synchronize	Force synchronization of the selected source file with the interfaces it implements.
Update Parser Database	Update the Java parser database with the classes of the selected package, thus making those classes available in addition to the standard Java 2 Platform SDK classes when using the Java code completion feature in the Editor.

Table 70: View Actions

Action	Description
CloseView	Close the active window.
Customize	Bring up a JavaBeans component's customizer.
Editor Window	Open a new instance of the Editor Window.
Execution Window	Open a new instance of the Execution Window.
Explore From Here	Open a new instance of the Explorer, with the selected node as the root.
Explorer Window	Open a new instance of the Explorer.
Global Properties	Open a Property Sheet window to display the property sheet for any subsequently selected object.
Next Workspace	Switch to the next workspace.
Object Browser	Open the Object Browser.
OutputWindow	Open a new instance of the Output Window.
Previous Workspace	Switch to the previous workspace.
Properties	Open the property sheet in a separate window for the selected object.
Status Line	Install the IDE's status line (available only for toolbars).
Web Browser Window	Open a new instance of the Web Browser window.
Workspaces	List and switch Workspaces.

Table 71: Window Actions

Action	Description
Clone View	Clone the current view.
Dock Into	Dock the current window into a separate window or into a multi-tab window.
Next Tab	Switch to the next tab in a multi-tab window.
Opened Windows	List opened windows in the Windows menu (available only for menus).

Action	Description
PreviousTab	Switch to the previous tab in a multi-tab window.
Undock Window	Undock the current tab from the MultiWindow.

Appendix F

Custom Property Editors

The following property editors are available both for modifying properties of components in your applications and for adjusting various settings and options in the IDE.

Color Editor

The Color Editor contains several tabs, each of which represents a different way to select a color.

Table 72: Panes in the Color Editor

Tab	*Description*
Swatches	Contains the Swing spectrum of colors and an additional color chart of the most recently selected colors. Click on a color in the chart to select it. The Preview panel demonstrates how your color selection looks as either foreground or background color. Click OK to apply the color.

Tab	Description
HSB	Displays a color square that enables you to change the hue, saturation, and brightness to create your own shade of color. Click on the H, S, or B radio button, then drag the circle in the square to adjust that color aspect. As you drag the circle, the current values for hue, saturation, brightness, are displayed as well as the values for red, green, and blue amounts.
RGB	Provides slider controls that you use to set the individual values for red, green, and blue. You can also enter values for each in the text fields provided. You can check your color setting in the Preview panel. Click OK to apply the color.
AWT palette	Contains a scrolling list of standard AWT colors. Click on a color to select it, then click OK to apply the color.
Swing palette	Contains a scrolling list of preset default colors for each state of a Swing component that uses a color indicator or has a color background or foreground. You can apply any of these colors to any item with a color property.

Fonts Editor

You can select the font family, type, and size from the Font Editor for any of the editors available under Editor Settings in the Global Options window and for any text property of a component. Select the family, style, and size from the scrolling lists. You can view your font selections in the Preview pane. Click OK in the Font Editor to close the dialog box with your selections set. Click OK in the Fonts and Colors Editor to set the change for the selected editor or window component.

Border Editor

Visually separate areas of a window or panel by adding a border. The Border Editor provides several styles of borders that you can apply to any component that has a border property. Select the border type from the scrolling list and click OK to add the border to the component. You modify the properties for the border in the properties pane below the scrolling list.

Cursor Editor

Assign a cursor style to an item with the Cursor Editor. The Cursor Editor provides a list of standard cursor styles to denote such things as text insertion, resizing ability, relocation, and a wait period. For items that take a cursor property, click on the property and choose the cursor style either from the combo box or from the Cursor Editor dialog box.

String Editor

Create or change the content of a tool tip or other text string with the String Editor. Click on a text property value to get the **...** box button. Click the button to open the String Editor. In the textfield, type the text for the string. Click OK to establish the change.

Icon Editor

Assign icons to represent an object or to denote an object's state with the Icon Editor. For items with an icon property, click on the property name, then click on the **...** button to open the Icon Editor. Click the radio button that corresponds to the image source, a URL, file, or classpath. In the `Name` field, enter the path of the image or click the dialog box to open a file chooser. The preview pane below the `Name` field displays the image. Click **OK** to assign the icon to the property.

Dimension Editor

Set the maximum, minimum, or preferred size for a selected component with the Dimension Editor. Click an item's size property to get the **...** button. Click the button to open the Dimension Editor. Enter the desired width and height values and click **OK** to set the property values.

Bounds Editor

Change the default size and location of individual top-level windows with the Bounds (or

Rectangle) Editor. For example, the Explorer window has default settings that place it to the far left of the screen. You can change the default setting to place the window to the far right by changing the X value of its `Bounds` property. For items with a `Bounds` property, click on `Bounds` in the property list. Click on the **...** button that appears to open the Bounds (Rectangle) Editor. Use the `X` and `Y` fields to set the default location on the screen where you want the window to open. Use the `Width` and `Height` fields to set the default size of the window.

Insets Editor

Determine the position of components within a cell in a layout manager with the Insets Editor. For items with an Insets property, click the property name to get the **...** button. Click the button to open the Insets Editor. Specify a value for each field to position a component within its cell in the layout.

Modifiers Editor

Change a class or method modifier with the Modifiers Editor. Click the `Modifiers` property of a class or method and click on the **...** button to open the Modifiers Editor. Only those modifiers that are permissible for the selected item are active. Choose the accessibility type from the **Access** list and choose the field modifier you want to apply from the **Other Modifiers** panel. Click **OK** to confirm the selection.

Parameters Editor

Add, edit, or remove a method parameter with the Parameters Editor. Click the `Parameters` property and click the property editor dialog box button to open the Parameters Editor. The default method parameter is displayed in the parameter list.

To add a parameter, click **Add**. A dialog box opens in which you can choose the parameter type and enter the parameter name. Click **OK** to add the parameter to the list of parameters.

To edit a parameter, select a parameter in the parameters list and click **Edit**. Change the parameter type or name in the dialog box that opens. Click **OK** to establish the change.

To remove a parameter, select it in the list of parameters and click **Remove**. Click **OK** to confirm the deletion.

Exceptions Editor

Add, edit, or remove an exception with the Exceptions Editor. Click the `Exceptions` property of a method, then click the **...** button to open the Exceptions (Identifiers) Editor.

To add an exception, click **Add**. A dialog box opens in which you can enter the name of an exception. Click **OK** to add it to the list of exceptions.

To modify an exception name, select the exception in the exceptions list and click **Edit**. Edit the name in the dialog box that opens. Click **OK** to establish the change.

To remove an exception, select it in the list of exceptions and click **Remove**. Click **OK** to confirm the deletion.

Error Expression Editor

In the Error Expression Editor, you can create a regular expression in POSIX format describing the format of the error output of the specified compiler. You can also define a custom expression, if using a different compiler, and save that setting. Select one of the pre-defined sets for error descriptors: `Sun javac`, `Microsoft jvc`, or `IBM jikes + E`.

Customizer dialog box

The Customizer is a dialog box which can be used for advanced customization of a whole object at once. It is available by selecting **Customize...** from an object's contextual menu or by pressing the **Customize** icon on an object's property sheet.

Index