

Getting Started with JBuilder®

JBuilder® 2005

Borland®
Excellence Endures™

Borland Software Corporation
100 Enterprise Way
Scotts Valley, California 95066-3249
www.borland.com

Refer to the file `deploy.html` located in the `redist` directory of your JBuilder product for a complete list of files that you can distribute in accordance with the JBuilder License Statement and Limited Warranty.

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1997–2004 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

For third-party conditions and disclaimers, see the Release Notes on your JBuilder product CD.

Printed in the U.S.A.

JB2005introjb 6E5R0804
0405060708-9 8 7 6 5 4 3 2 1
PDF

Contents

Chapter 1		
Getting started in JBuilder	1	
Topics	1	
Introducing JBuilder	1	
Working in JBuilder	2	
Using JBuilder's editor	2	
Customizing JBuilder	3	
Tutorials	3	
Part I		
Introducing JBuilder		
Chapter 2		
Introduction	7	
Chapter 3		
JBuilder's work environment	9	
JBuilder workspace	9	
JBuilder menus	11	
Main toolbar	12	
Content pane	14	
File view tabs	14	
Project pane	16	
Project pane title bar	16	
Project pane Project tab	17	
Project pane Files tab	17	
Project pane Classes tab	17	
Project pane toolbar	17	
Project pane context menu	18	
Structure pane	18	
Structure pane title bar	19	
Structure pane tool bar	19	
Visual design and the structure pane	19	
UML and the structure pane	20	
EJB designer and the structure pane	20	
DD Editor and the structure pane	20	
Struts Config Editor and the structure pane	20	
Tiles Designer and the structure pane	21	
Faces Config Editor and the structure pane	21	
Message pane	21	
Message pane title bar	22	
Message pane tool bar	22	
Status bars	22	
Main status bar	22	
Editor Status bar	23	
Message status bar	23	
JBuilder projects	23	
Wizards	24	
Designers	25	
Tools	26	
Command-line tools	26	
Java component libraries	27	
JBuilder samples	28	
Chapter 4		
Learning more about JBuilder	29	
JBuilder documentation set	29	
Documentation conventions	31	
Using JBuilder's online help	32	
How to get Help	32	
Getting help in JBuilder's IDE	32	
Getting help from your web browser	33	
The Help Viewer user interface	33	
Using the Help Viewer	34	
Using embedded or standalone help	35	
Using the table of contents	35	
Synchronizing Help Viewer contents	36	
Using the index	36	
Using the tasks list	36	
Using search in the Help Viewer	36	
Creating book subsets for searching		
in the Help Viewer	37	
Searching for text in the current topic	37	
Copying text from the Help Viewer	37	
Starting the Help Viewer in a		
separate VM	37	
Hiding Help Viewer panes	38	
Adding tabs to the Help Viewer		
content pane	38	
Zooming in the Help Viewer	38	
Using bookmarks in the Help Viewer	39	
Setting proxy configurations	39	
Navigating in the Help Viewer	40	
Using the keyboard for scrolling	40	
Using Dynamic Help	41	
Viewing class reference documentation	41	
The layout of the reference		
documentation	41	
Printing tutorials and other documentation	42	
Adaptive techniques supported in JBuilder	42	
JBuilder keyboard navigation	42	
Section 508 compliance	43	
Additional resources for learning about JBuilder	43	
Developer support and resources	43	
Contacting Borland Developer Support	43	
Online resources	43	
World Wide Web	44	
Borland newsgroups	44	
Usenet newsgroups	44	
Reporting bugs	44	

Part II

Working in JBuilder

Chapter 5

Introduction 49

Chapter 6

Working with projects 51

Creating projects	52
Creating a new project	52
Creating projects from existing code	52
Adding files to a project.	52
Updating paths	53
Adding libraries, projects, and archives	53
Removing and deleting files and directories	54
Renaming projects and files	54
Opening projects	54
Switching projects	54
Closing projects	55

Chapter 7

Developing code 57

Using wizards to generate code	57
Using the object gallery	58
Working with the visual design tools	59
Editing files	60
Using code audits	60

Chapter 8

Using the JBuilder workspace 63

Navigating the workspace	64
Keyboard navigation.	64
Using the history lists	64
Using View navigation buttons	65
Using favorite links	65
Adding favorite links	65
Organizing favorite links	65
Using bookmarks	66
Using window action commands	66
Searching in the workspace	66
Searching trees	66
Finding classes	67
Searching in the editor	67
Using the content pane	67
Using the content pane file tabs.	68
Using the file tab context menu	68
Resizing the content pane.	69
Using the project pane	69
Using the project pane toolbar	70
Using file tool tips	70
Using file status decorations	70
Navigating the project tree	71
Viewing files	71
Switching between open projects	71
Using the project pane context menu	72
Adding new files, folders, directories, or packages.	72
Removing files and classes	73

Creating new files or packages for a web application.	73
Running an application from the project pane	73
Using the files pane	73
Using the classes pane	74
Displaying class hierarchy	74
Navigating in the classes pane.	75
Using the structure pane	75
Sorting in the structure pane.	76
Filtering in the structure pane	76
Changing structure pane icons.	77
Viewing Javadoc information in the structure pane	77
Using the structure pane to navigate in the source code	78
Viewing structure pane error messages	78
Navigating Java code hierarchy in the structure pane	79
Getting Java help from the structure pane	79
Using the message pane	79
Copying message pane text	80
Formatting message pane text.	80
Wrapping message pane text	81
Hiding and showing the message pane	81
Undocking the message pane	81
Using the status bars	81
Main status bar.	81
Editor status bar	81
Using the editor status bar to display line numbers.	82
Using the editor status bar to navigate to lines	82
Changing keymaps from the editor status bar	82
Changing editor font size from the editor status bar	82
Message status bar	82
Monitoring memory use	82

Chapter 9

Compiling and building Java programs 83

Compiling Java projects	83
Building Java projects	84

Chapter 10

Running, testing, profiling, and debugging source code 85

Running programs in JBuilder.	85
Unit testing	86
Optimizing and profiling source code	86
Debugging your program	87

Chapter 11

Managing application development 89

Local version control	89
Version control system (VCS) integration	89
Refactoring code	90

Visualizing code with UML	90
Managing project requirements	91
Internationalizing programs	91

Chapter 12

Developing applications 93

Developing web applications	93
Developing enterprise applications	94
Combining J2EE technologies into an enterprise application	94
Developing Enterprise JavaBeans	94
Working with XML	95
Developing web services	95
Developing mobile applications	96
Developing database applications	97

Chapter 13

Archiving and documenting applications 99

Archiving applications	99
Developing runnable application archives	100
Documenting source files	100

Part III

Using JBuilder's editor

Chapter 14

Introduction 105

Chapter 15

Working in the editor 107

Using the editor's context menus, file tabs, gutters, and actions	108
Using editor context menus	108
Using file tabs	108
Using the editor's gutter	108
Using editor actions	109
Navigating in the editor	109
Selecting, formatting, and finding text	109
Selecting text	110
Selecting text by line	110
Selecting text blocks	111
Adding characters to text blocks	111
Deleting characters from text blocks	111
Dragging and dropping text	112
Copying text and adding imports	112
Resizing the font	112
Formatting code	113
Accessing format settings	113
Applying formatting to your code	115
Customizing screen elements	115
Wrapping curly braces around code blocks	116
Finding text in the editor	117
Finding and replacing text	117
Finding and replacing in the path	117
Finding text with the Find In Path command	118
Navigating to line numbers	119

Finding code elements and definitions	119
Finding a symbol's definition	119
Finding references to a symbol	120
Finding the overridden method	120
Finding unused code elements and overriding methods	121
Applying Javadoc shortcuts	121
Using Javadoc code templates	121
Adding and editing Javadoc tags	122
Using JavadocInsight	122
Accessing JavadocInsight	123
Setting JavadocInsight pop-up timing	123
Setting JavadocInsight color options	123
Creating custom Javadoc tags	123
Resolving Javadoc conflicts	123
Using Javadoc QuickHelp	124
Folding Javadoc comments	124
Adding todo tags in the editor	124
Viewing todo comments	125
Using bookmarks in the editor	125
Adding bookmarks	125
Editing bookmarks	126
Viewing bookmarks	126
Navigating to bookmarks	126
Moving, opening, and adding files	127
Adding files to a project	127
Coding shortcuts	127
Printing support in the editor	128

Chapter 16

Using code shortcuts 129

CodeInsight	130
MemberInsight	131
Smart MemberInsight	131
ClassInsight	131
ParameterInsight	132
Tool tip expression evaluation	132
ExpressionInsight	132
Configuring CodeInsight	133
ScopeInsight	133
Enabling or disabling ScopeInsight	133
Locating code members	134
Code folding	134
Assigning code folding shortcuts	134
Accessing code folding options	135
Folding code and comment blocks and displaying tool tips	135
Disabling code folding	135
Navigating to class members	136
Navigating to class members using the main menu	136
ErrorInsight	136
Code templates	138
Adding code templates to code by typing the code template name	139
Adding code templates to code using the cursor	139
Adding code template file types	139

Adding import statements to your code	
using auto import	140
Adding template macros to code templates . . .	140
Creating code templates	140
Creating code templates from source code . . .	141
Surrounding source code with	
code templates	141
Importing or exporting code templates	142
Editing code templates	142
Setting tabs within code templates	143
SyncEdit	143
Editing with SyncEdit	143
Customizing SyncEdit screen elements . . .	144
TagInsight	145
EntityInsight	146
XML TagInsight	147
Tag inspector	147
Tag component palette	147
Correcting tag errors	148
Customizing TagInsight	148
Customizing markup language tags	149
Customizing markup language editing	149
Customizing markup language formatting . .	149

Chapter 17

Customizing the editor 151

Setting keymapping preferences	151
Selecting a keymapping for the editor	152
Editing keymaps	152
Constructing new keymaps	153
Creating keymap reference copies	154
Searching for keymap actions	155
Customizing the editor's look and actions . . .	155
Customizing editor display	156
Customizing editor actions	156
Splitting the source view	157
Displaying line numbers	158
Displaying white space characters	158
Positioning file tabs	159
Limiting open files	159
Customizing unused variables	159
Setting Editor preferences	160

Part IV

Customizing JBuilder

Chapter 18

Introduction 165

Chapter 19

Configuring your workspace 167

Using JBuilder's "classic" configuration	168
Managing configurations	168
Editing configurations	169
Maximizing and restoring panes	169
Revealing and hiding panes	169

Iconifying, docking, and undocking panes	170
Iconifying and undocking	170
Docking	170
Positioning tabs	171
Viewing multiple files	171
Redocking files	172

Chapter 20

Setting JBuilder preferences 173

Importing settings 176

Chapter 21

Specifying default project settings 179

Setting paths	179
Setting general properties	180
Specifying runtime configurations	180
Specifying build properties	181
Setting code format properties	181
Specifying file status decorations	182
Setting server and service properties	182
Configuring personality properties	182
Viewing file information	182
Configuring code audits	183

Chapter 22

Extending JBuilder with OpenTools 185

Part V

Tutorials: Basic

Chapter 23

Introduction 189

Chapter 24

Tutorial: Building an application 191

Step 1: Creating the project	192
Step 2: Generating your source files	194
Step 3: Building and running your application . . .	197
Step 4: Customizing your application's	
user interface	198
Step 5: Adding a component to your	
application	202
Step 6: Editing your source code	204
Step 7: Compiling, building, and running your	
application	204
Step 8: Running your application from	
the command line	205
Step 9: Adding an event to a button	206
Step 10: Completing your UI	207
Step 11: Preparing your application	
for deployment	209
Step 12: Running your deployed application	
from the JAR file	210
Step 13: Running your deployed application	
from the command line	211

HelloWorld source code.	211
Source code for HelloWorldFrame.java	212
Source code for HelloWorldClass.java.	215
Source code for HelloWorldFrame_AboutBox.java	216

Chapter 25

Tutorial: Building an applet **219**

Overview	220
Good Evening applet	220
Step 1: Creating the project.	221
Step 2: Generating your source files	223
Step 3: Building and running your applet	227
Step 4: Customizing your applet's user interface	228

Step 5: Adding AWT components to your applet	232
Step 6: Editing your source code	234
Step 7: Deploying your applet	238
Deploying your applet with the Archive Builder	239
Step 8: Modifying the HTML file	242
Step 9: Running your deployed applet from the command line	244
Testing your deployed applet on the Web	245
Applet source code	245
Applet HTML source code for GoodEveningApplet.html	246
Applet class source code for GoodEveningApplet.java	246

Index **249**

Tables

3.1	Main toolbar icons	12	15.1	Commands for finding text	117
3.2	File view tabs	14	15.2	Page header variables	128
4.1	Typeface and symbol conventions	31	16.1	Shortcut keystrokes	130
4.2	Platform conventions	32	16.2	ErrorInsight menu options for source code	137
8.1	Structure pane filters	76	17.1	Editor Preferences.	160
9.1	Build features available in all JBuilder editions	84	20.1	Global	174
9.2	Build features available in JBuilder Developer and Enterprise	84			

Figures

3.1	JBuilder workspace	10	15.1	JBuilder's editor	107
4.1	JBuilder's Help Viewer	33	16.1	CodeInsight pop-up window	130
7.1	Object gallery	58	16.2	Example of SyncEdit text selection	143
7.2	JBuilder in design view	59	16.3	ElementInsight pop-up window	146
8.1	Content pane showing the source view (provides editor)	67	16.4	EntityInsight pop-up window	146
8.2	Project pane context menu	72	17.1	CUA Keymap Editor	153
8.3	Structure View Properties dialog box	76	24.1	JBuilder's IDE elements	196
8.4	Structure pane icons	77	24.2	UI designer elements	198
8.5	Message pane	79	25.1	UI designer	229

Tutorials

Building an application	191	Building an applet	219
-----------------------------------	-----	------------------------------	-----

Getting started in JBuilder

Getting Started with JBuilder is the place to start if you are new to JBuilder. It introduces JBuilder, describing JBuilder's graphical user interface and the major components that comprise the integrated development environment (IDE). It describes the resources that are available to you to learn about JBuilder, and shows you how to view and use online help with JBuilder's Help Viewer.

This book contains information to help you use the JBuilder work environment to perform common tasks. It also includes comprehensive information about JBuilder's editor, which includes numerous coding tools and shortcuts. You'll also learn how to customize JBuilder to suit your specific needs. This book includes basic tutorials to help you further explore JBuilder as you build your own application and applet.

Getting Started with JBuilder is intended to give you an overall understanding of what JBuilder is and how to use it. Topics that describe concepts and features at a very high level include references to more detailed information in other parts of the documentation set. This book provides the information you need to get started with JBuilder, and directs you to resources and additional documentation to get the most out of JBuilder.

Topics

Getting Started with JBuilder contains the following parts and chapters:

Introducing JBuilder

- [Chapter 3, "JBuilder's work environment"](#)

Introduces the JBuilder integrated development environment (IDE), and describes the components of which it consists, including the workspace, projects, wizards, designers, tools, component libraries, and samples.

- [Chapter 4, "Learning more about JBuilder"](#)

Describes the resources Borland provides to help you learn more about JBuilder, including the documentation set and JBuilder's online help.

Working in JBuilder

- [Chapter 6, “Working with projects”](#)
Explains how to create a JBuilder project, describes what a JBuilder project is, and explains how to perform basic project management tasks.
- [Chapter 7, “Developing code”](#)
Provides an overview of the methods for generating and editing code in JBuilder, including the use of visual design tools, JBuilder’s many wizards, and the editor.
- [Chapter 8, “Using the JBuilder workspace”](#)
Describes how to use the different elements of the JBuilder workspace, such as panes, context menus, and tabs, and how to perform common tasks, such as using bookmarks, navigating with keyboard shortcuts, searching for classes or text, and viewing files.
- [Chapter 9, “Compiling and building Java programs”](#)
Gives an overview of how JBuilder is used to compile and build.
- [Chapter 10, “Running, testing, profiling, and debugging source code”](#)
Gives an overview of how you can run, test, profile, optimize, and debug your source code from within JBuilder.
- [Chapter 11, “Managing application development”](#)
Describes the many ways that JBuilder helps you manage your code, from local version control to integrated version control systems. Includes information about refactoring code, visualizing code with UML, and internationalizing your applets and applications.
- [Chapter 12, “Developing applications”](#)
Discusses some of the types of applications that you can develop with JBuilder, and how JBuilder speeds the development process with supplied tools and features.
- [Chapter 13, “Archiving and documenting applications”](#)
Describes how JBuilder can help you prepare your application for deployment with archiving and documentation features.

Using JBuilder’s editor

- [Chapter 15, “Working in the editor”](#)
Explains how to use JBuilder’s code editor features to perform basic operations, such as selecting, formatting, and finding text, finding code elements and definitions, applying Javadoc shortcuts, using bookmarks, working with files, and printing your source code.
- [Chapter 16, “Using code shortcuts”](#)
Describes how to use the powerful code shortcut tools and features available in the editor. Some of these include CodeInsight, ScopeInsight, code folding, ErrorInsight, SyncEdit, and code templates.
- [Chapter 17, “Customizing the editor”](#)
Provides information for modifying the editing environment in JBuilder. Includes instructions for setting keymapping preferences, and describes how to change the appearance and behavior of the editor.

Customizing JBuilder

- [Chapter 19, “Configuring your workspace”](#)

Describes how you can rearrange the panes and tabbed pages of your workspace in a way that works best for you. You can save specific workspace configurations that you find useful for different kinds of work.

- [Chapter 20, “Setting JBuilder preferences”](#)

Introduces the Preferences dialog box (Tools|Preferences), which you can use to customize JBuilder, with settings that allow you to add file types, set options for running and debugging, configure audio options, and configure web, XML, UML, and EJB designer options. Describes how you can import prior JBuilder settings.

- [Chapter 21, “Specifying default project settings”](#)

Explains how you can specify default project properties, such as source paths, JDK version to compile against, and required libraries. The default project properties can be applied whenever you create a new project.

- [Chapter 22, “Extending JBuilder with OpenTools”](#)

JBuilder’s OpenTools API provides a means for you to modify or extend the features and function of JBuilder to best suit your needs.

Tutorials

These tutorials will give you hands-on experience to help you learn how to use JBuilder.

- [Chapter 24, “Tutorial: Building an application”](#)

Explains how to create a simple “Hello World” application using JBuilder.

- [Chapter 25, “Tutorial: Building an applet”](#)

Explains how to create an AWT applet using JBuilder.

P a r t I

Introducing JBuilder

Introduction

This section of *Getting Started with JBuilder* introduces the various panes, tabs, toolbars, status bars, icons, and menus of JBuilder's workspace, as well as the other major components of JBuilder, including wizards, designers, tools, and component libraries. This section describes the resources that are available to you to learn more about JBuilder, and shows you how to view and use online help with JBuilder's Help Viewer.

This section contains the following chapters:

- [Chapter 3, "JBuilder's work environment"](#)

Introduces the JBuilder integrated development environment (IDE), and describes the components of which it consists, including the workspace, projects, wizards, designers, tools, component libraries, and samples.

- [Chapter 4, "Learning more about JBuilder"](#)

Describes the resources Borland provides to help you learn more about JBuilder, including the documentation set and JBuilder's online help.

JBuilder's work environment

Available features
vary by JBuilder
edition

JBuilder is an award-winning integrated development environment (IDE) for the Java programming language. JBuilder includes a comprehensive collection of tools, designers, and wizards to help you design, develop, test, deploy, and manage your Java applications throughout their lifecycle. JBuilder also includes many ready-made components and templates to further increase consistency and productivity.

The following sections describe the major components of the JBuilder's work environment:

- [“JBuilder workspace” on page 9](#)
- [“JBuilder projects” on page 23](#)
- [“Wizards” on page 24](#)
- [“Designers” on page 25](#)
- [“Tools” on page 26](#)
- [“Java component libraries” on page 27](#)
- [“JBuilder samples” on page 28](#)

See also

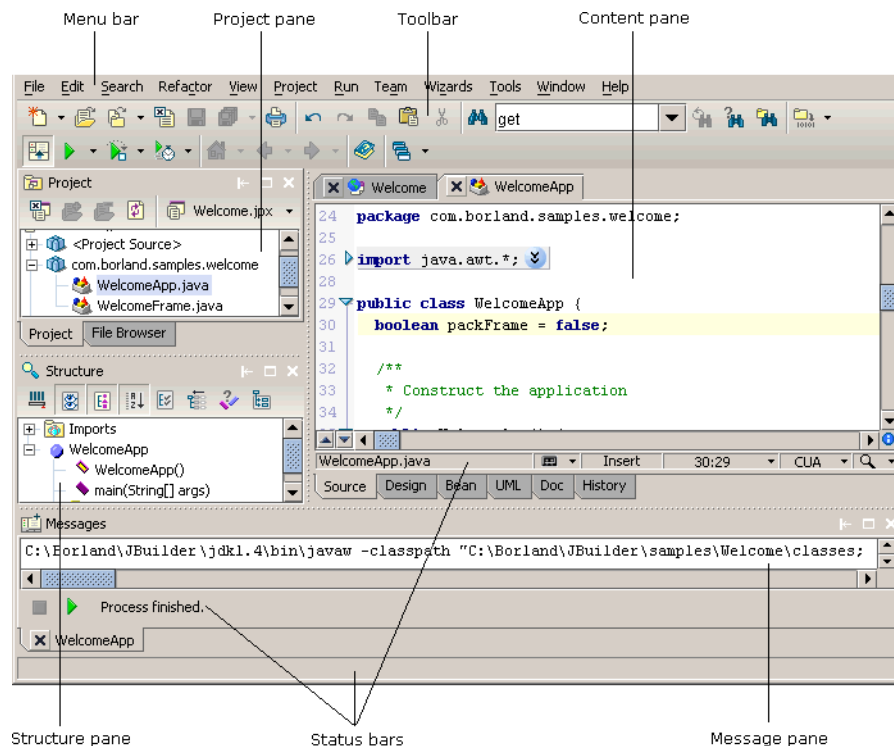
- [“Using the JBuilder workspace” on page 63](#) for information on how to use the different elements of the JBuilder workspace to perform common tasks

JBuilder workspace

JBuilder uses one window to perform most of the development functions: editing, visual designing, navigating, browsing, compiling, debugging, and other operations. This window is JBuilder's workspace, and it contains several panes for performing these development functions. You can undock the panes that comprise the JBuilder

workspace. The tabbed panes that are available in the content pane depend on what kind of file you have selected in the project pane.

Figure 3.1 JBuilder workspace



Workspace element	Description
Menu bar	The menu bar includes several menus of commands for designing, developing, testing, deploying, and managing your Java applications. The menu bar includes the following menus: File, Edit, Search, Refactor, View, Project, Run, Team, Enterprise, Tools, Window, and Help. The commands available in each menu vary by JBuilder edition.
Main toolbar	The main toolbar is displayed at the top of the JBuilder workspace under the menu bar. It is composed of smaller toolbars grouped by functionality: File, Edit, Search, Build, Run/Debug, Navigate, Help, and Workspaces. The buttons available for each of the smaller toolbars varies by JBuilder edition.
Content pane	The largest pane, hosting the main views of the open files. Many files can be open in the content pane, but only one file at a time can be active. You can change the view for the active file by clicking the file tabs at the bottom of the content pane. Possible views include the Source, Design, Bean, UML, Doc, and History views, depending on the type of file that is active. The availability of views and their features vary by JBuilder edition.
Project pane	Displays the contents of the active project and provides right-click access to project-wide commands. Provides a file browser, opened from the Files tab, and a class browser, opened from the Classes tab. The tabs can be turned on and off from the View menu (View/Panes). The Classes tab is off by default.
Structure pane	Displays the structure of the file currently active in the content pane. The structure pane hosts different views, so what appears in the structure pane is pertinent to the view tab selected in the content pane. Structure pane filter settings also affect the structure pane view.

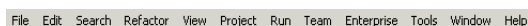
Workspace element	Description
Message pane	Displays the output of various processes, such as compiler messages, search results, debugger UI, refactoring, Javadoc, unit testing, viewing todos, version control commands, and WS-I testing tools functions for web services. It can also host modeless components. For example, when you start a debugging session, the debugger appears in the message pane. The CaliberRM plug-in (JBuilder Enterprise) is also displayed here.
Status bars	JBuilder's status bars display messages and features relevant to the frame and view they pertain to, and to what's happening in the active file.

See also

- [Chapter 8, “Using the JBuilder workspace”](#)
- [Chapter 19, “Configuring your workspace”](#)
- [Chapter 20, “Setting JBuilder preferences”](#)
- [Chapter 15, “Working in the editor”](#)
- [“Setting Editor preferences” on page 160](#)

JBuilder menus

The JBuilder menu bar is at the top of the JBuilder workspace, below the title bar.



The menus are described in the following table. The availability of some menus and menu commands depends upon the personalities selected in the project properties. The availability of menu commands also varies with JBuilder edition. The individual commands for each menu are described in detail in the online help.

Menu	Description
File	The File menu contains commands for creating and opening projects, creating and opening files, closing and saving files and projects, printing files, and exiting JBuilder.
Edit	The Edit menu contains commands for editing source code, formatting, folding code and code comments, accessing wizards, completing code with CodeInsight, and accessing context-sensitive help for APIs.
Search	The Search menu contains commands for finding and replacing text, searching incrementally, finding classes and definitions in source files, navigating to lines, methods, and class members, for adding and managing bookmarks, and for viewing todos in the message pane.
Refactor	The Refactor menu contains commands for refactoring and optimizing your program. Available commands will depend what's selected in the editor.
View	The View menu contains commands for viewing and working with workspaces, toolbars, and browser panes.
Project	The Project menu contains commands for project management.
Run	The Run menu contains commands for running and debugging your program.
Team	The Team menu contains version control commands for the selected version control system (VCS) integrations for the active project.
Enterprise	The Enterprise menu contains commands for configuring and deploying servers, enterprise-level setups, and EJB processes.
Tools	The Tools menu contains commands for setting preferences, configuring the JBuilder environment, and displaying various internal and external tools.

Menu	Description
Window	The Window menu contains commands for switching between open files and projects and for navigating through the history of opened files.
Help	Commands on the Help menu access the JBuilder online documentation set, link to the borland.com web site, display the sample Welcome Project, and display the JBuilder About box.

In addition to the menus on the menu bar, context menus provide quick access to commands that pertain to a specific part of the JBuilder workspace. Context menus are described in the documentation where applicable.

See also

- [Chapter 21, “Specifying default project settings”](#)

Main toolbar

The main toolbar is displayed at the top of the IDE under the menu bar. It is composed of smaller toolbars grouped by functionality: File, Editing, Search, Build, Run/Debug, Navigate, Help, and Workspaces. You can modify the toolbar display by checking or unchecking selections on the View/Toolbars menu. You also can view the toolbar button name and a brief description of the button by moving your cursor over the button. The button name appears below the button, and a brief description of the button appears in the main status bar.

Available features vary slightly by JBuilder edition. For details on exactly which features are available from a toolbar icon in your edition of JBuilder, see [“JBuilder menus” on page 11](#).

The toolbar shown here is wrapped; it probably will look slightly different on your monitor depending on the width of your browser window.





















The toolbar provides shortcut buttons for the following menu commands:

Table 3.1 Main toolbar icons

Icon	Menu equivalent	Description
	File New	Opens the object gallery where you can select from a variety of wizards. The New arrow opens a drop-down menu that displays the New menu command which opens the object gallery. It also displays the entire object gallery options in menu format.
	File Open	Opens a project, file, or package.
	File Reopen	Reopens a project, file, or package. Select from a history list.
	File Close	Closes the current file.
	File Save File	Saves the current file.
	File Save All	Saves all open projects and files, using the current names. The Save All icon arrow opens a drop-down menu that includes the Save All command and an option to save any open, modified open file.
	File Print	Prints selected file or text.

Table 3.1 Main toolbar icons (continued)

Icon	Menu equivalent	Description
	Edit Undo	In the editor, reinserts any characters you deleted, deletes any characters you inserted, replaces any characters you overwrite, or moves your cursor back to its prior position. Undoes actions in the designers. There are multiple levels of undo.
	Edit Redo	Reverses the effects of an Undo. There are multiple levels of redo.
	Edit Copy	Copies selected text in the editor or selected objects in the UI designer to the clipboard.
	Edit Paste	Pastes the contents of the clipboard to the location of the cursor.
	Edit Cut	Cuts selected text in the editor or selected objects in the UI designer to the clipboard.
	Search Find	Searches for text within the currently active file. Select previously used searches from the drop-down menu.
	Search Search Again	Finds the next occurrence of a search string in the currently active file.
	Search Replace	Replaces specified text with other specified text in the active file.
	Search Find Classes	Loads the specified class into JBuilder's browser. Must be on the class path of the active project.
	Project Make	Compiles archives, resources, and any .java files within the selected project that have outdated or nonexistent .class files. Also compiles any imported files that the project depends on and which have outdated or nonexistent .class files. The arrow provides a drop-down menu allowing you to choose Make (the default), Rebuild, or a user-defined build target.
	View Messages	Toggles the visibility of the message pane.
	Run Run Project	Runs your application using the default configuration specified on the Run page of the Project Properties dialog box. Click the Run Project arrow to select a different configuration from the drop-down list.
	Run Debug Project	Debugs your program using the default runtime configuration selected on the Run page of the Project Properties dialog box. Click the New or Edit button on the Run page to add or change a configuration or click the Debug Project arrow to select a different configuration from the drop-down list.
	Run Optimize Project	Evaluates the project's code using Optimizeit. This is only available when you have Optimizeit installed. Click the Optimize Project arrow to select a different configuration from the drop-down list.
	View Back	Takes you back to the previous location in the history list. Click the arrow to access the back history list.
	View Forward	Takes you forward to the next location in the history list. Click the arrow to access the forward history list.
	Help Help Topics	Opens the help viewer.
	View Workspace Chooser	Saves, resets, manages, or sets your workspace to the default setting.

Content pane

The content pane displays all opened files as a set of tabs. To *open* a file in the content pane, either double-click it in the project pane or select it in the project pane and press *Enter*.

The name of each open file is displayed on its tab at the top of the file in the content pane. When you click on a file tab, that file becomes the *current* or *active* file and can be edited directly.

To customize the content pane tabs, choose Tools|Preferences and change the settings on the Content pane page. You can also customize the configuration of the panes in JBuilder's IDE.

You can open files from different projects at the same time, but only one project at a time appears in the project pane. To switch between projects, select the desired project from the project pane toolbar's drop-down list or from the Window menu. When you switch to another project, that project appears in the project pane and its open files become accessible in the content pane.

You can split the source view of a file into two or more vertical or horizontal panes. This setting applies to an individual file, not to all the opened files in the content pane. You can have a different configuration for every opened file. To split the view into panes, click and drag a file's tab when the Source tab is selected.

File view tabs

The content pane gives you access to various file views and operations by way of the file view tabs shown at the bottom of each file window. The tabbed pages that are available in the content pane depend on what kind of file you have selected in the project pane. Only tabs appropriate to the opened file will appear in its window in the content pane.

Table 3.2 File view tabs

Tab name	Description
Source	The Source tab displays the source code of the active file in the editor.
Class	Displays the read-only stub source generated by decompiling the class file if the Java source isn't available. For example, when you search for a class in the Find Classes dialog box (Search Find Classes) and the Java source isn't available, the generated stub source for the class file opens in the class pane. Also, when you're viewing a UML diagram of a class file and choose Go To Source from the context menu, the generated stub source displays in the class pane.
Design	The Design tab displays the appropriate designer for the active file. Several types of designer are available. In the designer, you can create and modify components, use drag and drop design, manipulate layouts, and see how code renders in the user interface.
Bean	The Bean tab exposes the BeansExpress Property, Event, BeanInfo, and Property Editor designers. Use them to add properties and events to your bean, choose what properties are exposed, and create custom property editors. The Bean tab is read-only in JBuilder Foundation.
UML	The UML tab displays a UML browser for viewing and navigating UML diagrams of the selected class or package. This is a feature of JBuilder Enterprise.
Doc	The Doc tab shows the API reference documentation (if available) for the active class in the content pane. This is a feature of JBuilder Developer and Enterprise.
History	The History tab displays information that helps you manage revisions of files. It displays the source code of the active file and a revision list of previous versions of the file. The features of the history pane vary by JBuilder edition.

Table 3.2 File view tabs (continued)

Tab name	Description
Screens	The Screens tab displays the Screen Manager which is used to visually manage a MIDlet's <code>Displayable</code> screens and their relationships.
StarTeam	The StarTeam tab provides access to information and commands pertaining to StarTeam version control and configuration management. This is a feature of JBuilder Developer and Enterprise.
View	The View tab displays the opened HTML, XML, or image file in a browser or image viewer in the content pane. To view XML files, the Enable Browser View option must be enabled on the XML page of the Preferences dialog box (Tools Preferences XML). Viewing XML files is a feature of JBuilder Developer and Enterprise.
Web View	The Web View tab displays the output from your running web file, such as JSPs, servlets, SHTML, and HTML. For JSPs, the Web View tab displays the output from your running JSP. For servlets, this tab displays the output from the running servlet in parsed HTML. This is a feature of JBuilder Developer and Enterprise.
Web View Source	The Web View Source tab displays raw markup of the active HTML file. This is a feature of JBuilder Developer and Enterprise.
Flow	The Flow tab displays the JSF Flow Designer. This is a feature of JBuilder Developer and Enterprise.
Web Module DD Editor	Displays an editor for the <code>web.xml</code> deployment descriptor file of a web module. This is a feature of JBuilder Developer and Enterprise.
Faces Config Editor	Displays an editor for the <code>faces-config.xml</code> deployment descriptor file of a JSF web module. This is a feature of JBuilder Developer and Enterprise.
Struts Config Editor	Displays an editor for the <code>struts-config.xml</code> deployment descriptor file of a Struts web module. This is a feature of JBuilder Developer and Enterprise.
Tiles Editor	Displays an editor for the <code>tiles-def.xml</code> deployment descriptor file of a Struts web module. This is a feature of JBuilder Developer and Enterprise.
Transform View	The Transform View tab displays a browser view of the transformed XML document with an XSL stylesheet applied if available. This is a feature of JBuilder Developer and Enterprise.
Transform View Source	The Transform View Source tab displays the source code of the transformed XML document. This is a feature of JBuilder Developer and Enterprise.
EJB Designer	The EJB Designer tab displays the EJB designer, which you use to create EJB components. This is a feature of JBuilder Enterprise.
EJB Module DD Editor	The EJB Module DD Editor tab displays the EJB Module DD editor. Use the DD editor to make changes to an EJB module's deployment descriptors. This is a feature of JBuilder Enterprise.
Application DD Editor	Displays an Application DD editor, which you can use to edit the deployment descriptors of an Application module. This is a feature of JBuilder Enterprise.
Application Client DD Editor	Displays an Application Client DD editor, which you can use to edit the deployment descriptors of an Application Client module. This is a feature of JBuilder Enterprise.
Connector DD Editor	Displays a Connector DD editor, which you can use to edit the deployment descriptors of a Connector module. This is a feature of JBuilder Enterprise.
Web Services Designer	Displays the Web Services designer, which you use to import and export web services. When the Web Services designer is open, the Flow designer is also for editing the web services deployment file. The Flow designer is a feature of the server designer module. For web services created with the Apache Axis toolkit, you use the Axis Flow designer to edit the deployment file. For web services created with the WebLogic toolkit, you use the WebLogic Flow designer to edit the deployment file. This is a feature of JBuilder Enterprise.

Switch from one available view to another by clicking the tab containing the view you want or by choosing View|Switch Viewer To "<Viewname>". View multiple files by dragging a file tab to split the viewer or create another viewable pane.

Project pane

The project pane, in its default position, is displayed to the upper left of the content pane. It shows the open project and the contents of the project. This pane also hosts the file browser by default.

Structure pane

The structure pane, in its default position, is displayed below the project pane. It shows the structure of the file that's active in the content pane. The way it displays structure depends on which view is being used in the content pane.

Message pane

Various processes are displayed in the message pane, such as search results, the debugger, build results, compiler errors and warnings, unit testing, refactoring code, version control, and WS-I testing tools functions for web services.

See also

- [Chapter 19, “Configuring your workspace”](#)
- [“Searching trees” on page 66](#)
- [Chapter 8, “Using the JBuilder workspace”](#)

Project pane

The project pane is, by default, in the top left of the JBuilder workspace. It is a movable pane. The project pane shows the current open project and all the nodes (files, packages, and descendants) it contains. It also displays a project group node, when the open project is a member of a project group.

When your workspace is configurable, the project pane can be docked (attached to the workspace), undocked (free floating), or iconified (reduced to an icon on the left border of the workspace).

Project pane title bar

When your workspace is configurable, the project pane has a title bar with buttons for window action commands that let you configure the pane in the workspace. A context menu of Window action commands is also available by choosing View|Window Actions, pressing *Alt+F10*, or right-clicking the title bar.

Double-clicking the title bar maximizes the pane. If the pane is maximized, double-clicking the title bar restores the pane to its original size and location. Click and drag the title bar to undock the pane. Click and drag the title bar while holding the *Ctrl* key to dock an undocked pane to the workspace.

If your workspace is not configurable, the pane has no title bar, and window action commands are not available.

For more information, see [Chapter 19, “Configuring your workspace,”](#) and [“Using window action commands” on page 66.](#)

Project pane Project tab

The project pane Project tab displays the contents of the active project. It consists of the following items:

- A small toolbar with five buttons and a pull-down menu (see [“Main toolbar” on page 12](#) for descriptions)
- A tree view of the contents of the active project
- A context-sensitive menu for each *node* (for example, file, package, or directory nodes) in the project pane, accessed by right-clicking the node

Project pane Files tab

The project pane Files tab displays your home and project directories and computer drives. The directory icons have the same appearance as the icons in the File Selection dialog box. The files pane contains the following items:

- A small toolbar with three buttons and a pull-down menu (see [“Main toolbar” on page 12](#) for descriptions)
- A tree view of your home and project directories and computer drives
- A context-sensitive menu for each *node* (for example, drive, directory, folder, favorites, or file nodes) in the project pane, accessed by right-clicking the node





Project pane Classes tab







The Classes tab appears in the project pane when you choose View|Panels|Classes. The Classes pane displays a tree view of the inheritance structure of the selected class (target class), including the child-parent relationship of any class within the inheritance hierarchy. The classes pane contains the following items:

- A small toolbar with three buttons and a pull-down menu (see [“Main toolbar” on page 12](#) for descriptions)
- An initial view of the top-level Object class with a link to open the Select Class dialog box to select a class
- A hierarchical class tree view

Project pane toolbar

The project pane includes a files pane and a classes pane (also known as file and class browsers). Each pane has its own unique buttons, as well as shared buttons. Depending on the pane on display, the associated buttons will vary. The project, files, and classes panes have toolbars with the following buttons:

Button	Description
	Close Project — closes the active project. Before closing any changed files, JBuilder prompts you to save changes. Check the files you want to save and click OK.
	Add Files/Packages/Classes — opens the Add File Or Packages To Project dialog box where you choose the name of an existing file or package to add to the current project. The file is added to the list of files in the project pane.
	Remove From Project — removes the selected file from the current project. A prompt appears, asking whether to remove the file. When you click Remove, the selected file is removed from the current project. The file is not deleted from the drive.
	Refresh — refreshes the node list in the project, files, and classes panes.

Button	Description
	Show Decorated Only — shows only the files or packages that include decorations. The decorations signify modified files, VCS files, files not in the repository or project, and read-only files.
	Select An Open Project To Work With — displays a pull-down menu of open projects. Choose a project file name from the menu to make the corresponding project active.
	Favorites — adds the selected drive, directory, or folder to the top of the tree view of the Files page, identified by the Favorites icon. Adds your selected Favorites directory to the File Selection dialog box. Allows you to organize the added Favorites drive, directory, or folder.
	Find Class — opens the Find Class dialog box. Search or browse for a class to appear in the classes pane (displays the class inheritance structure).
	Select Class in Class Browser — selects the class file (open in the editor), to appear in the classes pane (displays the class inheritance structure).
	Show Interfaces — displays interfaces within the class inheritance structure nodes in the classes pane.

Project pane context menu

Right-click a node in the project pane to display the project pane context menu. The commands on this menu vary depending on the type of node you selected and the relative state of the node.

See also

- [Chapter 6, “Working with projects”](#)
- [Chapter 19, “Configuring your workspace”](#)
- “Managing paths” in *Building Applications with JBuilder*

Structure pane

The structure pane is located, by default, in the lower left of JBuilder’s IDE. It is a movable pane. When a file type with appropriate content is open in a viewer that can provide structural information, this pane displays a structural analysis of the contents of the file. You can think of the structure pane as a table of contents for the file.

You can also use the structure pane to navigate the file. For example, if you have a `.java` file open in the Source view, you see classes, variables, methods, and interfaces in the structure pane. You can then click any of these elements in the structure pane and the editor moves to and highlights that element in the source code. This gives you a much faster way to browse and find the elements of a `.java` file than scrolling through it or searching for a word. You can also search the structure pane for a particular element by moving focus to the tree in the structure pane and beginning to type. Right-click in the structure pane to see the context menu.

With certain types of files open in certain views, you can,

- Change the display order of the structure elements by choosing Properties from the context menu
- Filter what appears in the structure pane by using the buttons on the structure pane’s toolbar

Structure pane title bar

When your workspace is configurable, the structure pane has a title bar with buttons for window action commands that let you configure the pane in the workspace. A context menu of Window action commands is also available by choosing View|Window Actions, pressing *Alt+F10*, or right-clicking the title bar.

Double-clicking the title bar maximizes the pane. If the pane is maximized, double-clicking the title bar restores the pane to its original size and location. Click and drag the title bar to undock the pane. Click and drag the title bar while holding the *Ctrl* key to dock an undocked pane to the workspace.









If your workspace is not configurable, the pane has no title bar, and window action commands are not available.

For more information, see [Chapter 19, “Configuring your workspace”](#) and [“Using window action commands” on page 66](#).

Structure pane tool bar

The toolbar on the structure pane includes buttons for changing the appearance of the tree displayed in the pane. The toolbar includes buttons for filtering, grouping, and sorting the objects that appear in the tree. You can also choose to show method return types and field types.

The Structure pane has a toolbar with the following buttons:

Button	Description
	Adjust visibility filter: opens a slider pop-up to set structure pane visibility to display a combination of public, protected, package, or private class elements.
	Group by visibility: groups the class elements by visibility.
	Group Classes/Methods/Fields: groups the class elements by class, method, and field.
	Sort alphabetically: sorts elements alphabetically from a to z.
	Group properties: groups elements by individual properties.
	Flatten view of inner classes: displays all class elements, including inner class elements.
	Show types: toggles visibility of method return types and field types.
	Group by inheritance: groups elements by inherited classes.

See also

- [“Using the structure pane” on page 75](#)
- [Chapter 19, “Configuring your workspace”](#)

Visual design and the structure pane

When you select a visually designable `.java` file and then select the Design tab at the bottom of the content pane, the *component tree* appears in the structure pane. The tree includes folders for specific designer types, including the UI, Menu, Data Access, and Default designers. The component tree displays the designable objects in the file and how they are nested and classified. The component tree lets you select, move, and rename components.

See also

- [“Using the component tree” in *Designing Applications with JBuilder*](#)

UML and the structure pane

When you view a UML diagram, the structure pane provides a tree view of relationships by category. The structure pane can also be used for selection and navigation to a class or package.

See also

- “UML and the structure pane” in *Building Applications with JBuilder*

EJB designer and the structure pane

When you are working the with EJB designer, the structure pane displays the schema of a data source and allows you to import and add new data sources. By right-clicking elements that appear in the structure pane, you can modify the schema of a data source, then use that schema to create entity beans. You can also export a modified schema back to the data source.

See also

- “Importing a data source” in *Developing Applications with Enterprise JavaBeans*
- “Modifying the imported data source schema” in *Developing Applications with Enterprise JavaBeans*

DD Editor and the structure pane

When the DD Editor is open, the elements of a module’s deployment descriptor(s) are shown in the structure pane. Double-clicking an element opens the page of the editor for that element.

The DD Editor can edit these types of modules:

- EJB modules
- Web modules
- Application modules
- Application client modules
- Connector modules

See also

- “Editing J2EE deployment descriptors” in *Developing Applications for J2EE Servers*
- “Editing EJB deployment descriptors” in *Developing Applications with Enterprise JavaBeans*
- “Editing the web.xml file” in *Developing Web Applications*

Struts Config Editor and the structure pane

When the Struts Config Editor is open, the elements of the `struts-config.xml` file are shown in the structure pane. Double-clicking an element opens the page of the editor for that element. Double-clicking an Action element opens the Struts Action Designer. Double-clicking a Form Bean element displays the Form Bean Designer. The structure pane doesn’t change when either of the Struts designers is opened. It still shows the structure of the `struts-config.xml` file.

See also

- “Editing the struts-config.xml file” in *Developing Web Applications*
- “Using the Struts Designers” in *Developing Web Applications*

Tiles Designer and the structure pane

When the `tiles-def.xml` file is selected in the project pane, the elements of the file are shown in the structure pane. Double-clicking an element opens the page of the editor for that element. Double-clicking a Definition element opens the Tiles Designer. The structure pane doesn't change when the Tiles Designer is opened. It still shows the structure of the `tiles-def.xml` file.

See also

- “Editing the `tiles-def.xml` file” in *Developing Web Applications*
- “Tiles Designer” in *Developing Web Applications*

Faces Config Editor and the structure pane

When the Faces Config Editor is open, the elements of the `faces-config.xml` file are shown in the structure pane. Double-clicking an element opens the page of the editor for that element. Double-clicking a Navigation Rule element opens the Navigation Designer. The structure pane doesn't change when the Navigation Designer is opened. It still shows the structure of the `faces-config.xml` file.

See also

- “Editing the `faces-config.xml` file” in *Developing Web Applications*
- “Using the JSF Designers” in *Developing Web Applications*

Message pane

The tabbed message pane appears in the lower part of JBuilder's IDE, below the structure and content panes. The message pane displays various processes, including:

- Building, including building Javadoc
- Compiling
- Running
- Debugging
- Searching
- Unit testing
- Version control
- Refactoring
- Viewing `@todo` tags
- Requirements management with CaliberRM
- TCP Monitor

If your code has errors when you compile, error messages are listed in the message pane. To go to the line of code with the error, click the error message.

As you run an application, the message pane displays a tab with the name of the active process. For more information, see the status bar.

When debugging, the debugger user interface and debugging process are in the message pane.

Output from the selected server is also displayed in the message pane. Additionally, refactoring details are displayed in the message pane.

The message pane can also display the results of a search for all instances of a specified expression. For example, choose SearchFind, specify a search string, and click the Find All button. All strings in the current file that match the search string appear in the message pane on a Search Results page. Click a string on the Search Results page to go its instance in the file.

The Search Results tab will also display the results of a search for all references to a specified symbol or definitions of a symbol. For example, select a symbol in the editor, right-click, and choose Find References or Find Definition.

Message pane title bar







When your workspace is configurable, the message pane has a title bar with buttons for window action commands that let you configure the pane in the workspace. A context menu of Window action commands is also available by choosing View/Window Actions, pressing *Alt+F10*, or right-clicking the title bar.

Double-clicking the title bar maximizes the pane. If the pane is maximized, double-clicking the title bar restores the pane to its original size and location. Click and drag the title bar to undock the pane. Click and drag the title bar while holding the *Ctrl* key to dock an undocked pane to the workspace.

If your workspace is not configurable, the pane has no title bar, and window action commands are not available.

Message pane tool bar

When you are debugging, the Message pane includes a toolbar at the top with the following buttons:

Button	Description
	Copy: Copies the selected content or the whole content if none is selected.
	Clear All: Deletes all output content.
	Search: Opens the Find/Replace dialog box to search the output content.
	Search Again: Searches again in the output content.
	Auto Scroll: Automatically scrolls down when new output is generated.
	Word Wrap: Activates word wrapping, otherwise the message pane wraps by the line.

For more information, see [Chapter 19, “Configuring your workspace,”](#) and [“Using window action commands” on page 66.](#)

Status bars

There are three status bars in JBuilder’s IDE:

- Main status bar
- Editor status bar
- Message status bar

Main status bar

The Main status bar is displayed at the bottom of the IDE window. It keeps you updated on any processes and their results. It also provides tool tip help for context menus: place your cursor over a context menu item to see that item described in this status bar.



At the far right of the Main status bar is the garbage collection icon. Clicking it forces garbage collection. The drop-down arrow displays a context menu that has two commands: Force Garbage Collection and Properties, which displays a Status Bar Properties dialog box.

Editor Status bar

The Editor status bar is displayed at the bottom of each opened file window in the content pane. It displays information specific to the current file, such as the name of the file, the cursor location (line number and column), and the insertion mode in a text file, the current keymap settings, and the magnifying glass icon you can use to resize text in the editor.

Clicking the line number and column numbers displays the Go To Line Number dialog box, in which you can enter a line number and, when you click OK, the specified line appears highlighted in the editor. The down-arrow next to the line number/column numbers displays a menu that displays the Go To Line Number dialog box or lets you select or unselect the Show Line Numbers option.

The down-arrow next to the current keymap setting presents a menu of keymap choices. The down-arrow next to the magnifying glass icon presents a menu of choices to allow you to zoom or unzoom the editor. The ScopeInsight icon on the far right invokes ScopeInsight when you click it.



Message status bar

The Message status bar is displayed at the bottom of the message pane during such processes as running, debugging, unit testing, refactoring, and version control. It informs you of the status of the current process.

JBuilder projects

JBuilder projects help you organize, manage, and share your work. The concept of a JBuilder project is integral to understanding and working with JBuilder. When you work with multiple files or applications in JBuilder, you can group the files and related resources into a project. Using projects, you can easily specify properties to apply to all the files you are working on. For example, you can set the version of the Java Development Kit (JDK) to use when building and running your project, and you can add files or directories to the class path.

Projects also make it easier for JBuilder to be used in a team environment. JBuilder's integrations with popular version control systems, such as StarTeam and CVS, allow you to check in and control entire projects, including the project files. The ability to apply local labels (Project!Manage Local Labels) lets you manage local revisions of the source files in your project.

Note the following features and concepts related to JBuilder projects:

- Project file

Each project is administered by a project file. The JBuilder project (`.jpx`) file is an XML file that contains the shared project settings and a list of all the files added to the project. Each project also includes a `.jpx.local` file that contains the local project's history list of the user's editor settings and some user actions, such as debugging.

Project settings vary according to the type of project you create, but can include directory structure, paths to libraries, sources, JDK versions, and relevant deployment settings. JBuilder records paths relative to the project parent directory whenever possible.

JBuilder uses the list of files and the project settings whenever you load, save, change, build, or deploy a project. A project file is not edited directly by the user, but it is modified whenever you use JBuilder to add or remove files or set project properties. You can see the project file as the top node of the project tree in the project pane.

- Project properties

Project properties determine how the project is compiled and run. Using the Project Properties dialog box (Project|Project Properties), you can specify project path settings, set up a run configuration for your project, specify how a project is built, customize the display of UML diagrams, specify server options, and much more. You can specify default project properties (Project|Default Project Properties) to use when you create new projects.

- Project pane

When you open a project, the project pane displays a hierarchical view of the project's contents, such as files, packages, and descendants. It also displays a project group node when the open project is a member of a project group. The project pane is, by default, in the top left of the JBuilder workspace, but it can be moved.

- Project menu

The Project menu contains commands for project management, including commands for compiling your project or project group, as well as commands for opening the Project Properties and Default Project Properties dialog boxes.

- Project group

Project groups are containers for projects and can be useful when working with related projects. Project groups provide advantages, such as ease of navigation between projects and building projects as a group. The ability to group projects is a feature of JBuilder Developer and Enterprise.

- Project page in the object gallery

The Project page of the object gallery (File|New|Project) contains shortcuts to wizards for creating new projects, pulling projects from a version control system (VCS), importing a VisualCafé project, or creating a project group.

See also

- [Chapter 6, “Working with projects”](#) for information about basic project management tasks
- “Creating and managing projects” in *Building Applications with JBuilder*
- “Managing paths” in *Building Applications with JBuilder*

Wizards

Wizards provide numerous options for quickly creating code elements or performing tasks. You can do anything from copying files with the Import Source wizard accessed from the object gallery or from the Project menu, to creating a new interface that includes Javadoc comments with the Implement Interface wizard (Edit|Wizards|Implement Interface).

The Archive Builder wizards create JAR files for numerous file types, including a runnable JAR file for an application. For more information on a specific wizard, open the wizard and click the Help button at the bottom of the wizard steps.

See also

- [“Using wizards to generate code” on page 57](#) for additional information on accessing and using wizards

Designers

JBuilder provides a variety of visual design tools to help you build applications quickly and consistently. JBuilder includes designers to help you develop Ant or Swing user-interfaces, JavaBeans, database applications, web services, struts web modules, mobile applications, and EJBs. In most cases, you click the Design tab of the content pane to access the designer for the active file. The designers provide helpful tool tips as you move over the designer surface and components.

In general, when you activate a designer in the content pane, the structure pane displays a component tree. The component tree allows you to view and manage the components in a visually designable file. It shows all of the components in the active file and their relationships, the layout managers associated with UI containers, and the type of designer each component uses. It provides access to commands and controls for the designers and the components. Changes made in the component tree are immediately reflected in the Inspector, the design surface, and the source code.

JBuilder includes the following designers:

- UI designer — use the UI designer to build user interfaces that contain visual elements, such as list boxes and buttons. When you first click the Design tab after opening a file in the content pane, JBuilder displays the UI designer by default. UI components appear in the UI folder of the component tree.
- Menu designer — the Menu designer is used to create menus. Menu components appear in the Menu folder of the component tree. To access the menu designer, choose the Menu tab.
- Data access and column designers — designers for database development provide a design surface for adding and implementing data access components. The Column designer lets you view and modify column properties in a navigable table.
- Default designer — the Default designer contains generic non-UI and non-menu pop-up components and actions.
- Web services designers: the Web Services designer provides a design surface for importing and exporting web services, and visually creating and implementing web services. The flow designers let you visualize and construct the message flow of the web service, and edit the web services deployment file. (JBuilder Enterprise)
- Struts designers — the Struts designers allow you visually design elements of your Struts web module. These designers allow you to manipulate Actions, Form Beans and Tiles elements with drag-and-drop actions. (JBuilder Developer and Enterprise)
- EJB designer — the EJB designer provides a rapid application development (RAD) environment for EJB 2.0 and 1.1 development. The EJB designer provides *two-way* editing, allowing you to design your enterprise bean visually as JBuilder generates the code from your design. (JBuilder Enterprise)
- Mobile designer for designing MIDP and DoJa™ (i-mode™) applications. (JBuilder Developer and Enterprise)

See also

- [“Working with the visual design tools” on page 59](#) for an overview of JBuilder’s visual design tools
- “Introducing the designer” in *Designing Applications with JBuilder*
- “Working with columns” in *Developing Database Applications*
- “Using the Struts designers” in *Developing Web Applications*
- “Working with the EJB designer” in *Developing Applications with Enterprise JavaBeans*

- “Working in the Web Services designer” in *Developing Web Services*
- Creating a MIDP user interface in *Developing Mobile Applications for MIDP*
- Creating a user interface for i-mode applications in *Developing Mobile Applications for i-mode*

Tools

JBuilder includes several tools to help you build, test, and deploy your applications.

Supplied tools include:

Tool	Description
Database Pilot	A hierarchical database browser that also allows you to view and edit data.
JDBC Monitor	A window that you use to monitor the activity of database drivers while they are in use by JBuilder.
JDataStore Explorer	A tool for browsing, and performing database operations on JDataStore databases. The JDataStore Explorer also performs administrative tasks, such as creating or deleting databases, changing passwords, and administering users.
JDataStore Server	A graphical interface for the JDataStore server. Lets you monitor server usage information, such as connected users, server status logs, and database connections. You can also specify JDataStore Server options and start and stop the server.
Web Services Explorer	A tool to help you search for and publish web services. Assists you in searching Web Services Description Language (WSDL) servers, Universal Description, Discovery and Integration (UDDI) sites, and Web Services Inspection Language (WSIL) documents for available web services, and publishing web services to UDDI registries.
Web Services Console	A console that lets you view the WSDL that describes the service, test the web service, and view the Simple Object Access Protocol (SOAP) messages between the client and the server for a deployed web service.
TCP Monitor	A tool for monitoring the SOAP messages that are passed between the client and the server using Transmission Control Protocol (TCP).

Note You can add and remove tools from the Tools menu with the Configure Tools dialog box (Tools|Configure External Tools). Tools you add can run externally from JBuilder or they can run as a service within JBuilder.

Command-line tools

JBuilder has a command-line interface that includes arguments for:

- Building projects
- Comparing files
- Displaying configuration information
- Displaying the license manager
- Disabling the splash screen
- Enabling verbose debugging mode for OpenTools authors
- Updating projects from a version control system

See also

- “Using command-line tools” in *Building Applications with JBuilder*
- “Database administration tasks” in *Developing Database Applications*
- “JDataStore Administration” in *JDataStore Developer’s Guide*
- “Browsing and publishing web services” in *Developing Web Services*
- “Monitoring SOAP messages” in *Developing Web Services*

Java component libraries

JBuilder includes libraries of value-added classes and components that can be used to quickly create consistent, reliable applications. This includes classes and components available on the component palette, as well as those you can use programmatically.

JBuilder includes the following component libraries:

Library	Description
DataExpress	Includes packages that provide basic data access. The <code>com.borland.dx.dataset</code> package provides general routines for data connectivity, management, and manipulation. The <code>com.borland.dx.sql.dataset</code> package provides data connectivity functionality that is JDBC specific. The <code>com.borland.dx.text</code> package contains classes and interfaces that control alignment and formatting of objects, and the formatting of data and values. This package also handles formatting and parsing exceptions and input validation. (all editions of JBuilder)
dbSwing	Includes the <code>com.borland.dbswing</code> package, which contains components that allow you to make Swing components capable of accessing database data through DataExpress <code>DataSets</code> . (all editions of JBuilder)
JDataStore	Includes packages for connecting to and performing transactions with JDataStore databases. The <code>com.borland.datastore</code> package provides basic connectivity and transaction support for local JDataStore databases. The <code>com.borland.datastore.jdbc</code> package contains the JDBC interface for the DataStore, including the JDBC driver itself, and classes for implementing your own DataStore server for multi-user connections to the same DataStore. The <code>com.borland.datastore.javax.sql</code> package provides functionality for distributed transaction (XA) support. The classes in this package are used internally by other Borland classes. You should never use the classes in this package directly. (all editions of JBuilder)
DataExpress for EJB	Contains the <code>com.borland.dx.ejb</code> package, which contains DataExpress for EJB components that allow you to use entity beans with DataExpress <code>DataSets</code> to resolve and provide data. Some of these components can be added from the EJB page of the component palette in the UI designer. (JBuilder Enterprise)
InternetBeans Express	Contains the <code>com.borland.internetbeans</code> package, which provides components for generating and responding to the presentation layer of a web application. Many of these components can be added from the InternetBeans page of the component palette in the UI designer. (JBuilder Developer and Enterprise)

Library	Description
XML Database Components	Contains the <code>com.borland.jbuilder.xml.database.xmldbms</code> , <code>com.borland.jbuilder.xml.database.template</code> , and <code>com.borland.jbuilder.xml.database.common</code> packages, which contain model-based and template-based components for transferring data between XML documents and a database, as well as exception classes for XML database components. Many of the components in these packages can be added from the XML page of the component palette in the UI designer. (JBuilder Developer and Enterprise)
Javax Classes	Contains the <code>com.borland.javax.sql</code> package, which provides implementations of JDBC 2.0 <code>DataSource</code> and connection pooling components. These classes can be used with any JDBC driver, but have additional functionality that is specific to the <code>JDataStore</code> JDBC driver. (all editions of JBuilder)
SQL Adapter Classes	Contains the <code>com.borland.sql</code> package, which contains the <code>SQLAdapter</code> interface. This interface can be implemented by any JDBC class and adapted for improved performance. (all editions of JBuilder)
SQL Tools Classes	Contains the <code>com.borland.sqltools</code> package, which contains classes for retrieving report output using SQL queries specified in XML format. (JBuilder Developer and Enterprise)
I/O and Utility Classes	Contains packages that provide basic string handling functions and other utilities. The <code>com.borland.jb.io</code> package provides basic input and output string handling functions. The <code>com.borland.jb.util</code> package provides diagnostic, searching, conversion and string handling utilities. (all editions of JBuilder)
Unit Test Support Classes	Contains the <code>com.borland.jbuilder.unittest</code> package, which contains classes for creating test cases and test suites, running tests, and recording results. (JBuilder Developer and Enterprise)
CORBA Express	Contains the <code>com.borland.cx</code> package, which contains CORBA connection classes for CORBA-based distributed applications. (JBuilder Enterprise)

Note When you use JBuilder designers to add a component from one of the supplied libraries, JBuilder automatically adds the library to the project.

See also

- *DataExpress Component Library Reference* in the online documentation
- “Redistribution of classes supplied with JBuilder” in *Building Applications with JBuilder*

JBuilder samples

JBuilder includes many sample projects which demonstrate use of the rich component libraries provided with JBuilder, as well as use of Java base classes. Samples are located in the `samples` directory of your JBuilder installation. For additional information, and a description of individual samples, choose **Help|Learning About JBuilder|JBuilder Samples** from the JBuilder menu bar.

Learning more about JBuilder

JBuilder provides numerous documentation resources to help you learn about its features and tools. This section describes JBuilder's extensive documentation set and explains how to use JBuilder's online Help system. The online Help section describes the Help Viewer's main parts and how to use them, as well as how to view class reference documentation. Learn how to use the Help Viewer tools, including the navigation, search, tab, and bookmarking tools, to make viewing and finding documentation easier. Read through this section to ensure you are making the best use of the Help system's capabilities.

The section also explains printing documentation, keyboard navigation, viewing options, and other adaptive techniques JBuilder supports. Finally, it lists other places you might go to find additional information about using JBuilder.

JBuilder documentation set

You have several options to access the JBuilder documentation. You can view the documentation in JBuilder's Help Viewer by choosing from the main Help menu selections: Help Topics, Learning About JBuilder, JBuilder On The Web, and Reference Documentation. See ["Using JBuilder's online help" on page 32](#) for complete information about viewing help in the Help Viewer. The documentation is also available on your JBuilder CD in HTML format so that you can use any web browser to view it.

If you prefer to see the documentation in printed format, you can download the books in PDF format for printing; you can also download the documentation in HTML format for viewing online. For either option, go to the Borland JBuilder documentation web site at <http://info.borland.com/techpubs/jbuilder>.

Document	Description
All JBuilder editions	
<i>Getting Started with JBuilder</i>	Introduces you to JBuilder's integrated development environment and describes its components and learning resources. Explains how to use the workspace, wizards, tools, and code editor to perform common tasks for developing and managing your programming projects. Describes code shortcuts and how to customize the editor, workspace, and projects, and how to extend JBuilder with OpenTools. Includes tutorials for building a simple application and applet.

Document	Description
<i>Getting Started with Java</i>	Provides an overview of the Java programming language, class libraries, and editions.
<i>Building Applications with JBuilder</i>	Discusses how to manage projects in JBuilder and explains how JBuilder uses paths. Explains how to compile, debug, deploy, and internationalize programs in JBuilder. Discusses JBuilder's Javadoc, code visualization, code refactoring, and unit testing features. Also includes information on command-line tools, error messages, and how to migrate applications from other Java development tools.
<i>Designing Applications with JBuilder</i>	Explains how to design user interfaces and use layout managers. Includes step-by-step tutorials for creating user interfaces using JBuilder's visual design tools.
<i>Managing Application Development in JBuilder</i>	Discusses version management and explains how to use JBuilder's interfaces with three major version control systems as well as JBuilder's own version handling tools.
<i>JDataStore Developer's Guide</i>	Explains how to make effective use of JDataStore functionality. JDataStore is a high-performance, small-footprint, all Java database.
<i>Developing Database Applications</i>	Provides information on using JBuilder's DataExpress database functionality to develop database applications.
<i>DataExpress Component Library Reference</i>	Provides detailed information online about all the JBuilder value-added, data-aware components, classes, properties, methods, and events.
<i>OpenTools API Reference</i>	Provides reference documentation for the JBuilder OpenTools API.
<i>JDK documentation</i>	API reference documentation for the Sun Java Development Kit (JDK) You can access this documentation in one of two ways: <ul style="list-style-type: none"> ■ Choose HelpJava Reference. ■ Choose the Doc tab in the content pane when viewing a JDK file.
<i>Java Language Specification</i>	Contains version 2.0 of the Java Language Specification.
Context-sensitive online help	Provides information related specifically to the current JBuilder user interface element.
Developer and Enterprise editions	
<i>Developing Web Applications</i>	Discusses the technologies used for web development in JBuilder.
<i>Working with XML</i>	Discusses the technologies used for XML development in JBuilder.
<i>Developing Mobile Applications for MIDP</i>	Explains how to use the JBuilder integrated development environment to create and archive MIDP applications for J2ME enabled hand-held devices. Discusses how to manage projects and files, visually design a user interface, and compile, run, debug, and archive MIDP applications.
<i>Developing Mobile Applications for i-mode</i>	Explains how to use the JBuilder integrated development environment to create and archive J2ME applications for i-mode™ mobile phones using the DoJa SDK. Discusses how to manage projects and files, visually design a user interface, and compile, run, debug, and archive i-mode applications.
Enterprise edition	
<i>Developing Web Services</i>	Explains how to use the JBuilder web services features to create, browse, consume, and publish web services.
<i>Developing Applications for J2EE Servers</i>	Presents an overview of J2EE development and explains how to use J2EE servers with JBuilder.
<i>Developing Applications with Enterprise JavaBeans</i>	Describes how to create Enterprise JavaBeans using JBuilder's EJB tools and designers.

Documentation conventions

The Borland documentation for JBuilder uses the typefaces and symbols described in the following table to indicate special text.

Table 4.1 Typeface and symbol conventions

Typeface	Meaning
Bold	Bold is used for java tools, <code>bmj</code> (Borland Make for Java), <code>bcj</code> (Borland Compiler for Java), and compiler options. For example: <code>javac</code> , <code>bmj</code> , <code>-classpath</code> .
<i>Italics</i>	Italicized words are used for new terms being defined, for book titles, and occasionally for emphasis.
<i>Keycaps</i>	This typeface indicates a key on your keyboard, such as "Press <i>Esc</i> to exit a menu."
Monospaced type	Monospaced type represents the following: <ul style="list-style-type: none"> ■ text as it appears onscreen ■ anything you must type, such as "Type <code>Hello World</code> in the Title field of the Application wizard." ■ file names ■ path names ■ directory and folder names ■ commands, such as <code>SET PATH</code> ■ Java code ■ Java data types, such as <code>boolean</code>, <code>int</code>, and <code>long</code>. ■ Java identifiers, such as names of variables, classes, package names, interfaces, components, properties, methods, and events ■ argument names ■ field names ■ Java keywords, such as <code>void</code> and <code>static</code>
[]	Square brackets in text or syntax listings enclose optional items. Do not type the brackets.
< >	Angle brackets are used to indicate variables in directory paths, command options, and code samples. JDK 5.0 uses angle brackets to denote generics. For example, <code><filename></code> may be used to indicate where you need to supply a file name (including file extension), and <code><username></code> typically indicates that you must provide your user name. When replacing variables in directory paths, command options, and code samples, replace the entire variable, including the angle brackets (<code>< ></code>). For example, you would replace <code><filename></code> with the name of a file, such as <code>employee.jds</code> , and omit the angle brackets. See "Using command-line tools" in <i>Building Applications with JBuilder</i> for more information. Note: Angle brackets are used in HTML, XML, JSP, and other tag-based files to demarcate document elements, such as <code></code> and <code><ejb-jar></code> . The following convention describes how variable strings are specified within code samples that are already using angle brackets for delimiters.
<i>Italics, serif</i>	This formatting is used to indicate variable strings within code samples that are already using angle brackets as delimiters. For example, <code><url="jdbc:borland:jbuilder\\samples\\guestbook.jds"></code>
...	In code examples, an ellipsis (...) indicates code that has been omitted from the example to save space and improve clarity. On a button, an ellipsis indicates that the button links to a selection dialog box.

JBuilder is available on multiple platforms. See the following table for a description of platform conventions used in the documentation.

Table 4.2 Platform conventions

Item	Meaning
Paths	Directory paths in the documentation are indicated with a forward slash (/). For Windows platforms, use a backslash (\).
Home directory	The location of the standard home directory varies by platform and is indicated with a variable, <home>. <ul style="list-style-type: none"> ■ For UNIX, Linux, and OS X, the home directory can vary. For example, it could be /user/<username> or /home/<username> ■ For Windows NT, the home directory is C:\Winnt\Profiles\<username> ■ For Windows 2000 and XP, the home directory is C:\Documents and Settings\<username>
Screen shots	Screen shots reflect the Borland Look & Feel on various platforms.

Using JBuilder's online help

JBuilder displays online help topics in the Help Viewer. Topics can also be displayed in JBuilder's IDE or in a web browser.

Choose one of the following topics for detailed information about JBuilder's Help system:


- ["How to get Help" on page 32](#)
- ["The Help Viewer user interface" on page 33](#)
- ["Using the Help Viewer" on page 34](#)
- ["Navigating in the Help Viewer" on page 40](#)
- ["Using Dynamic Help" on page 41](#)
- ["Viewing class reference documentation" on page 41](#)
- ["Printing tutorials and other documentation" on page 42](#)

How to get Help

There are two ways to get help: in the JBuilder IDE and in a web browser. Note that if you're viewing documentation in a web browser, you won't have access to full-text search.

Getting help in JBuilder's IDE

To get help in JBuilder's IDE,

- 1 Choose Help|Help Topics from the JBuilder menu bar to open the Help Viewer.
-  2 Or, click the Help Topics button on JBuilder's main toolbar, the Help button in any JBuilder dialog box, or press *F1*.

There are several more ways to get help on all topics from JBuilder's IDE.

- Choose Search|Find Classes, choose a class, and click OK to open the class in the content pane. Click the Doc tab to see if any documentation is available for that class.
- Double-click a Java class name in the structure pane or project pane to open it and select the Doc tab in the content pane to view the documentation for the class, if available. You can also select a Java class name and press *Enter* to open the class.

- Press *Ctrl+Q* or choose **Edit|CodeInsight|Javadoc QuickHelp** to open Javadoc reference documentation.
- Press *F1* to get help for these topics:
 - Java keyword — position the cursor on a Java keyword in the editor.
 - Code member — position the cursor on any code member in the editor to access Javadoc information.
 - Property or event in the Inspector in the UI designer.
 - In the DD Editor or a designer.
 - In an inspector, such as in the Web Services Designer.

Getting help from your web browser

To get help from your own web browser, follow these steps:

- 1 Unjar all of the .jar files in your JBuilder doc directory.

To do so, use your favorite tool, such as WinZip, or, from the command line, go to the doc directory and enter the following for each JAR file:

```
<JBuilder>/<jdk>/bin/jar xf <docjarfile.jar>
```

- 2 After all of the JAR files are extracted, open `index.html` with your browser and choose your JBuilder edition from the drop-down list in the HTML Books section to load the documentation in a frameset. Choose a book from the drop-down list at the top of the page.

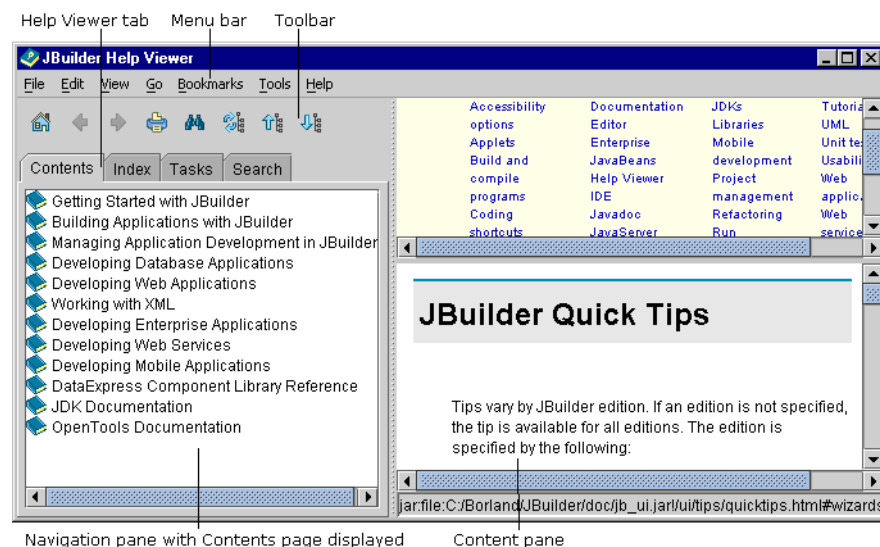
Tip If you want to view PDFs, copy the PDFs from the JBuilder CD to the pdf directory created when you extracted the JARs.

Note There is no full-text search capability across the online documentation set when you access it from your web browser.

The Help Viewer user interface











The JBuilder Help Viewer provides an easy-to-use user interface with many convenient features to assist you in finding and viewing documentation. The user interface elements include the menu bar, main toolbar, and navigation and content panes.

Figure 4.1 JBuilder's Help Viewer



The JBuilder Help Viewer includes the following user interface elements:

- Menu bar
- Navigation pane containing the following pages accessed from the Help Viewer tabs:
 - Contents page — displays an expandable table of contents for all books in a tree view
 - Index page — shows the index entries for all books
 - Tasks page — presents a list of helpful, common tasks
 - Search page — allows you to use complex search delimiters to search in all books or a subset of books
- Content pane — displays the text of the selected topic
- Content pane context menu — displays menu commands for navigation, printing, and so on (Right-click in the content pane to view the available commands.)
- Main toolbar with the following buttons:

Button	Name	Description
	Home	Goes to the first topic in the history list.
	Back	Goes to the previous topic in the history list.
	Forward	Goes to the next topic in the history list.
	Print	Prints the current topic.
	Find In Page	Finds text in the current topic.
	Synchronize Table Of Contents	Synchronizes the topic selection in the table of contents on the Contents page with the current topic viewed in the content pane.
	Previous Topic	Opens the preceding topic in the table of contents.
	Next Topic	Opens the next topic in the table of contents.
	Refresh button	Refreshes the documentation view of the embedded Help Viewer in the content pane.
	Show Navigation Panel	Opens the Navigation Pane of the embedded Help Viewer in the message pane.

Using the Help Viewer

The Help Viewer provides numerous resources for searching, customizing, and increasing usability. Choose one of the following topics for information on using some of the Help Viewer features:

- [“Using embedded or standalone help” on page 35](#)
- [“Using the table of contents” on page 35](#)
- [“Synchronizing Help Viewer contents” on page 36](#)
- [“Using the index” on page 36](#)
- [“Using the tasks list” on page 36](#)
- [“Using search in the Help Viewer” on page 36](#)
- [“Creating book subsets for searching in the Help Viewer” on page 37](#)
- [“Searching for text in the current topic” on page 37](#)
- [“Copying text from the Help Viewer” on page 37](#)
- [“Starting the Help Viewer in a separate VM” on page 37](#)

Choose one of the following topics for information on customizing the Help Viewer for easier use:

- [“Hiding Help Viewer panes” on page 38](#)
- [“Adding tabs to the Help Viewer content pane” on page 38](#)
- [“Zooming in the Help Viewer” on page 38](#)
- [“Using bookmarks in the Help Viewer” on page 39](#)
- [“Setting proxy configurations” on page 39](#)

Using embedded or standalone help

You can choose to view help from within JBuilder or from the standalone Help Viewer. Standalone help is turned on by default; however, embedded help can easily be activated. Both views give you access to the main Help Viewer toolbar for navigation and printing, as well as access to a navigation pane. The Help Viewer navigation pane includes a table of contents for all JBuilder books and reference documentation, a comprehensive list of tasks, a thorough index, and a complex search page.

Embedded help

You can activate embedded help from the Preferences dialog box (Tools|Preferences|Help). If you choose to use embedded help, the documentation appears in JBuilder's content pane as a Help tab containing a Help Viewer toolbar. You can tear off the Help tab to undock and redock it elsewhere. You also can use the main Help menu, as well as the main Search menu to help you find the documentation you require.



When you are viewing embedded help, you can choose to display the Help Viewer navigation pane (Contents, Index, Tasks, and Search tabs), in the message pane area by clicking the Show Navigation Panel button. Embedded help also includes a helpful context menu. Some of the context menu options include: Show Navigation Panel, Zoom In, Zoom Out, Print, and navigation commands.

Standalone help

The standalone Help Viewer provides a main menu that isn't accessible from embedded help. Some of the helpful menu actions include: adding new tabs to the content pane, adding and editing bookmarks, changing fonts, and using a proxy server for internet access. For quicker viewing, choose to use embedded help, and for a broader range of Help Viewer capabilities, choose the standalone Help Viewer.

To activate embedded or standalone help,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose Help in the left tree to open the Help page.
- 3 Click the Embedded Help or the Standalone Help radio button.

The default setting is Standalone Help.

- 4 Click OK to close the dialog box and activate the type of help you chose.

Using the table of contents

To choose a topic from the table of contents,

- 1 Choose Help|Help Topics to open the Help Viewer.
- 2 Click the Contents tab in the top left of the Help Viewer.
- 3 Double-click a book icon to expand the topics in that book.
- 4 Click an item in the table of contents to view the topic in the content pane.

Synchronizing Help Viewer contents

When you use the Search or Index tabs of the Help Viewer, the table of contents isn't visible. However, once you've found a document, you can easily discover its exact location in a book by synchronizing the document contents with the table of contents.

To synchronize Help Viewer contents,

- 1 Choose HelpHelp Topics to open the Help Viewer.
- 2 Use the Search or Index tab to find your target document.
- 3 Click the Synchronize Table Of Contents button on the Help toolbar.



The Help Viewer expands the table of contents and highlights the topic heading.

Using the index

To look up a topic in the index,

- 1 Click the Index tab in the top left of the Help Viewer.
- 2 Type the topic in the text field.
As you start typing, the index is searched incrementally to find the closest match.
- 3 Double-click the highlighted index topic or press *Enter* to view the content.
If there is more than one topic for the selected index entry, the Index pane splits in two and displays a topic list in the lower portion of the pane. Click the topic you want to display.

Using the tasks list

The Help Viewer provides a comprehensive list of tasks to help you accomplish your project goals. The tasks are high-level, simple tasks that encompass all of the IDE's capabilities.

To find task instructions,

- 1 Choose HelpHelp Topics to open the Help Viewer.
- 2 Click the Tasks tab.
- 3 Expand one of the task nodes to view specific tasks.
- 4 Click a task in the Tasks page to open the instructions in the content pane.

Using search in the Help Viewer

The Search page in the Help Viewer provides complex searching, including using wildcards, regular expressions, and searching HTML as plain text. You also can customize the search order results.

To use the Search page for finding text in all books or a subset of books,

- 1 Choose HelpHelp Topics to open the Help Viewer.
- 2 Click the Search tab at the top left of the Help Viewer.
- 3 Enter the search string in one of the following search fields: All Of, Exact Match, At Least One Of, or Without Any Of.
- 4 Choose to sort or not sort the search results.
- 5 Choose to search all books or create a new subset of books.
- 6 Choose to use wildcards or regular expressions for your search.
- 7 Choose whether to match the case, whole word, or search HTML as plain text.
- 8 Click the Search button to run the search.

If more than one entry is found for a given search, a Topics Found pane displays in the lower portion of the Search page. The Topics Found pane lists all the topics that match the search criteria in the Search page. Click the topic you want to display.

Tip Full-text search returns results from the entire online documentation set. If you find that it returns too many results, use the Index tab or create a book subset for a narrower search.

Creating book subsets for searching in the Help Viewer

When you are searching for information in the Help Viewer, you can limit the extent and length of search time by creating smaller subsets of the documentation set.

To create book subsets for Help Viewer search,

- 1 Choose Help|Help Topics to open the Help Viewer.
- 2 Click the Search tab to open the Help Viewer Search page.
- 3 In the Search Options area, from the Search In Subset drop-down menu, choose the New Subset menu option.
- 4 Click the New button in the Defining Subsets dialog box.
- 5 Double-click the Unnamed Subset title in the Subset list and enter the name of the new subset.
- 6 Choose books the Available Books field and click the right arrow to add the books to the new subset.
- 7 Click OK and the Help Viewer adds the new book subset to the Search In Subset menu options.

Searching for text in the current topic

To find text in the current topic of the Help Viewer,



- 1 Click the Find button in the Help Viewer or press *Ctrl+F*.
- 2 Type the search words in the Find Text In Current Topic dialog box.
- 3 Click Find Next or press *Enter* to execute the search. Press *Enter* again to continue searching.
- 4 Click Close to close the dialog box.
- 5 Press *F3* to search again without opening the dialog box.

Copying text from the Help Viewer

To copy text from the Help Viewer,

- 1 Select the text in the content pane.
- 2 Choose Edit|Copy from the Help Viewer menu bar or press *Ctrl+C*.

Starting the Help Viewer in a separate VM

You can run the Help Viewer in a separate virtual machine (VM) to provide access to tutorials and other help topics while a modal dialog box is being displayed. This option is set in the Help Viewer.

To configure the Help Viewer to start in a separate VM,

- 1 Open the Help Viewer (Help|Help Topics).
- 2 Choose Tools|Preferences to open the Help System Preferences dialog box.
- 3 Check the Start Help System In Separate VM option.
- 4 Click OK.
- 5 Restart JBuilder for the change to take effect.

Hiding Help Viewer panes

The Help Viewer displays two main areas: the navigation pane on the left where the Contents, Index, Tasks, and Search pages display and the content pane on the right. You can hide the navigation pane to expand the content pane by choosing a command from the context menu of the content pane or from the View menu.

To hide the navigation pane and expand the content pane:

- 1 Right-click in the Help Viewer's content pane and choose Hide Navigation Panel to hide the left pane. Choose Show Navigation Panel to show both panes.
- 2 Choose View|Hide Navigation Panel to hide the left pane. Choose View|Show Navigation Panel to show both panes.

Adding tabs to the Help Viewer content pane

The Help Viewer content pane (the right pane of the Help Viewer) only displays one document by default. You can add multiple tabs to the content pane to quickly view and navigate to additional documents without leaving the content pane. Each new tab pane displays a separate document.

Create new tabs by following these instructions:

- 1 Choose File|New Tab.

After creating a new tab, you can also access the New Tab menu command by right-clicking a tab and choosing New Tab or by using the keyboard shortcut, *Ctrl+T*.

- 2 Choose the document title in the table of contents on the Contents page.

The new tab displays the document title and the content pane displays the selected document.

Close tabs using one of these methods:

- Choose File|Close Tab or File|Close Other Tabs.

The Close Other Tabs command closes all tabs in the content pane. The Close Tab command only closes the tab of the displayed document.

- Right-click a tab and choose Close Tab or Close Other Tabs.

Zooming in the Help Viewer

You can change the apparent size of the font and images in the Help Viewer by zooming in and out. This allows you to examine images more closely, fit more text in the view window, or otherwise adjust the viewing size for your comfort. The zoom increments are small, so you can control the apparent size of the view precisely.

There are two ways to perform each type of zoom using a menu command or the keyboard:

Action	Menu command	Keyboard shortcut
Zoom In	View Zoom In	Press <i>Ctrl+NumPad+</i> (number pad plus sign (+) on the right side of the keyboard); hold down <i>Ctrl</i> and press + on the number pad again until the view is the size you want.
Zoom Out	View Zoom Out	Press <i>Ctrl+NumPad-</i> (number pad minus sign (-) on the right side of the keyboard); hold down <i>Ctrl</i> and press - again until the view is the size you want.
Normal view	View Zoom Normal	Press <i>Ctrl+NumPad*</i> (number pad asterisk (*) on the right side of the keyboard).

Using bookmarks in the Help Viewer

To add a bookmark for the current page in the Help Viewer, choose Bookmarks|Add Bookmark from the menu bar.

To edit bookmarks in the Help Viewer,

- 1 Choose Bookmarks|Edit Bookmark from the menu bar.

The Bookmark Editor dialog box opens.

- 2 Select the bookmark and click the option to remove, rearrange, or go to bookmark.

To navigate to a bookmark, choose Bookmarks and choose a bookmark from the list. The first ten bookmarks have mnemonics in the Bookmarks menu, so you want the most commonly used bookmarks at the top of the list. You are limited to 25 bookmarks on the menu, although you can edit the `maxMenuBookmarks` value in the `HelpProperties.xml` file to increase this number. The `HelpProperties.xml` file is created the first time you exit the Help Viewer and is saved to the `<.jbuilder>` directory in your home directory.

See also

- [“Using bookmarks in the editor” on page 125](#)

Setting proxy configurations

To access Internet sites from the Help Viewer through a proxy server, you must specify the host ID and port number for your proxy server. These values are set in the Help Viewer.










To enable the Help Viewer to use a proxy server to connect to the Internet,

- 1 Open the Help Viewer (Help|Help Topics).
- 2 Choose Tools|Preferences from the Help Viewer menu bar to open the Help System Preferences dialog box.
- 3 Check the Connect To Internet Via Proxy Server option.
- 4 Enter the name or numeric IP address of the proxy server in the Host field.
- 5 Enter the proxy server's port number in the Host field.
- 6 Check or uncheck the Use Simple Dialog For Context Sensitive Help option to select the following viewer options.
 - Checked (default setting): opens the documentation in a simple dialog box when you press *F1* in JBuilder.
 - Unchecked: opens the documentation in the Help Viewer when you press *F1* in JBuilder.
- 7 Click OK.
- 8 Restart JBuilder for the changes to take effect.

Navigating in the Help Viewer

The Help Viewer provides numerous, quick, one-click methods and keyboard shortcuts for navigating the help documentation.

The following table describes how to move around in the Help Viewer:

To	Do this...	Button	Keyboard shortcut
Change books or go to a topic	Click the topic title in the table of contents.		Use the arrow keys.
Expand or collapse a book group in the table of contents	Double-click the book group icon beside a topic on the Contents page.		Use the right arrow and left arrow keys to open and close a selected book group.
Expand or collapse a section of the table of contents	Click the icon beside a topic on the Contents page or double-click the topic.		Select the book and press <i>Enter</i> to open. You can also select the book and use the right arrow and left arrow keys to open and close the book.
Return to the previous topic in the history list	Click the Back button.		<i>Ctrl+Left Arrow</i>
Go to the next topic in the history list	Click the Forward button.		<i>Ctrl+Right Arrow</i>
Go to the first topic in the history list	Click the Home button.		<i>Ctrl+Home</i>
Find (search for) text in the current topic	Click the Find in Page button.		<i>Ctrl+F</i>
Search again			<i>F3</i>
Find (search for) text in all books	Click the Search tab.		
Follow a hypertext link underlined in the text	Click the hyperlink in the text.		
Synchronize the table of contents in the Contents page with the topic in the content pane.	Click the Synchronize Table Of Contents button.		<i>Ctrl+S</i>
Open the topic preceding the highlighted topic in the table of contents.	Click the Previous Topic button.		<i>Ctrl+Down Arrow</i>
Open the topic following the highlighted topic in the table of contents.	Click the Next Topic button.		<i>Ctrl+Up Arrow</i>
Open the Navigation Pane of the embedded Help Viewer in the message pane.	Click the Show Navigation Panel button.		

Using the keyboard for scrolling

When you select an entry in the Contents, Index, Tasks, or Search pages of the Help Viewer, the keyboard focus stays there. Keyboard navigation keys, such as *PgUp* (page up), *PgDn* (page down), and the arrow keys continue to scroll the left side of the Help Viewer rather than the content pane on the right side. If you want to use these keys in the content pane, you must first move the focus to that pane.

To shift focus to another area, use the *Tab* key. The *Tab* key changes the focus as follows:

- Contents page — Contents page, hyperlinks in content pane, Contents page
- Index page — entry field of Index page, index entries list, content pane
- Tasks page — Tasks page, hyperlinks in content pane, Tasks page
- Search page — entry field of Index page, index entries list, content pane

Using Dynamic Help

Dynamic Help provides quick access to task-oriented help information, along with access to the Borland home page, and to simple instructions for getting started. You can turn on Dynamic Help from the Preferences dialog box (Tools|Preferences|Help) and invoke Dynamic Help from either the Help menu (Help|Dynamic Help), or by moving your cursor in the work environment. Dynamic Help is off by default.

You can customize how quickly and in what form Dynamic Help appears. Go to the Help page of the Preferences dialog box (Tools|Preferences|Help) to view customization options for Dynamic Help, some of which include:

- None, minimum, medium, or maximum help
- Delay time before help appears
- Dockable window in the message pane
- Pop-up window in the IDE
- Context-dependent dockable window

You can easily turn off Dynamic Help when it's no longer needed whether it's in the form of a dockable or pop-up window. Simply navigate to the bottom of the Dynamic Help window, and choose Turn Off Dynamic Help.

Viewing class reference documentation

You can view reference documentation for a class in JBuilder's IDE, the Help Viewer, or a web browser.

To display reference documentation in the IDE, do one of the following:

- Double-click a class name in the structure pane or project pane to display source code for that class. Choose the Doc tab in the content pane to view the documentation for that class, if available.
- Choose Search|Find Classes, choose a class, and click OK. Choose the Doc tab in the content pane to open the Javadoc viewer to see the documentation for that class.
- Position the cursor on a class name and press *F1*.

For the class documentation to be found, the `import` statements at the top of your source file must point to the desired class or package. If the message `Java Symbol Not Found` is displayed, navigate to a source file that imports the desired class.

To display reference guides in the Help Viewer, do one of the following:

- In the JBuilder main window, choose Help and the desired reference guide.
- In the Contents page of the Help Viewer, choose a reference book.

The layout of the reference documentation

The documentation in the *DataExpress Component Library Reference* (Help|DataExpress Reference) is delivered in standard Javadoc format. For information on using the Javadoc format, see "How This API Document Is Organized."

Printing tutorials and other documentation

To print tutorials and other JBuilder documentation, open the documentation in the Help Viewer and choose File|Print. Preview the printed version of the document in a separate window by choosing File|Print Preview. The Help Viewer also enables you to change the document's printed appearance with the Page Layout command.

Choose File|Page Layout to open the Page Layout dialog box where you can set the following layout elements:

- Margins
- Scale
- Headers/Footers
- Portrait orientation
- Landscape orientation

Note To change the font family and size, choose Tools|Preferences|Fonts and make your selections before printing.

Documentation is also available in PDF format on the JBuilder web site at

<http://info.borland.com/techpubs/jbuilder/>.

For a list of tutorials and the names of the PDF files in which they appear, see Help|JBuilder Tutorials.

Adaptive techniques supported in JBuilder

JBuilder supports alternative navigation techniques for several reasons: to allow users to maintain their most comfortable working style, whether this includes a mouse or not, and to support adaptive strategies such as voice recognition software, which depend on commands being accessible from the keyboard.

JBuilder keyboard navigation

JBuilder supports common conventions of keyboard navigation:

- Use the *Shift+F10* keys to bring up the context menu for the object that has focus (wherever you place your cursor in JBuilder's IDE).
- Use the *Tab* key or the *Ctrl+Tab* keys to move between sections of the UI. Normally, this shifts focus left to right or in a clockwise direction.
- Use the *Arrow* keys to navigate up, down, and sideways within a section of the UI.
- Use the *Spacebar* to select or deselect an option, such as a radio button or check box.
- Where a mouse user would click a button, put focus on that button and press the space bar to achieve the same result.
- Every command is available from a menu. Accelerator keys are available for every menu command. (Normally this means pressing *Alt* + the key corresponding to the underlined letter in the menu option.) Consult the documentation for your operating system to determine how to display accelerator keys.
- Use extensive keyboard shortcuts while working in the editor. See the Keymaps page of the Preferences dialog box (Tools|Preferences).

Specific strategies for using JBuilder features without a mouse are described in context throughout the documentation.

Section 508 compliance

Supporting these techniques for keyboard navigation makes JBuilder compliant with Section 508 of the Rehabilitation Act of 1973 (29 U.S.Code 794d).

Additional resources for learning about JBuilder

For more information about JBuilder, see these options:

- Visit the JBuilder web site at <http://www.borland.com/jbuilder/> and the Borland Community web site at <http://bdn.borland.com/>.
- Take an introductory tour of JBuilder by opening the Welcome Project. Choose Help|Learning About JBuilder|Welcome Project from the JBuilder main menu.
- Check the Release Notes (Help|Release Information) to find information about the latest JBuilder updates and known problems with suggested workarounds.

Developer support and resources

Borland provides a variety of support options and information resources to help developers get the most out of their Borland products. These options include a range of Borland Technical Support programs, as well as free services on the Internet, where you can search our extensive information base and connect with other users of Borland products.

Contacting Borland Developer Support

Borland offers several support programs for customers and prospective customers. You can choose from several categories of support, ranging from free support upon installation of the Borland product, to fee-based consultant-level support and extensive assistance.

For more information about Borland's developer support services, see our web site at <http://www.borland.com/devsupport/>, call Borland Assist at (800) 523-7070, or contact our Sales Department at (831) 431-1064.

When contacting support, be prepared to provide complete information about your environment, the version of the product you are using, and a detailed description of the problem.

For support on third-party tools or documentation, contact the vendor of the tool.

Online resources

You can get information from any of these online sources:

World Wide Web

<http://www.borland.com/>
<http://info.borland.com/techpubs/jbuilder/>

Electronic newsletters To subscribe to electronic newsletters, use the online form at:
<http://www.borland.com/products/newsletters/index.html>

World Wide Web

Check the JBuilder page of the Borland website, www.borland.com/jbuilder, regularly. This is where the Java Products Development Team posts white papers, competitive analyses, answers to frequently asked questions, sample applications, updated software, updated documentation, and information about new and existing products.

You may want to check these URLs in particular:

- <http://www.borland.com/jbuilder/> (updated software and other files)
- <http://info.borland.com/techpubs/jbuilder/> (updated documentation and other files)
- <http://bdn.borland.com/> (contains our web-based news magazine for developers)

Borland newsgroups

When you register JBuilder you can participate in many threaded discussion groups devoted to JBuilder. The Borland newsgroups provide a means for the global community of Borland customers to exchange tips and techniques about Borland products and related tools and technologies.

You can find user-supported newsgroups for JBuilder and other Borland products at <http://www.borland.com/newsgroups/>.

Usenet newsgroups

The following Usenet groups are devoted to Java and related programming issues:

- [news:comp.lang.java.advocacy](#)
- [news:comp.lang.java.announce](#)
- [news:comp.lang.java.beans](#)
- [news:comp.lang.java.databases](#)
- [news:comp.lang.java.gui](#)
- [news:comp.lang.java.help](#)
- [news:comp.lang.java.machine](#)
- [news:comp.lang.java.programmer](#)
- [news:comp.lang.java.security](#)
- [news:comp.lang.java.softwaretools](#)

Note These newsgroups are maintained by users and are not official Borland sites.

Reporting bugs

If you find what you think may be a bug in the software, please report it to Borland at one of the following sites:

- **Support Programs page** at <http://www.borland.com/devsupport/namerica/>. Click the Information link under “Reporting Defects” to open the Welcome page of Quality Central, Borland’s bug-tracking tool.
- **Quality Central** at <http://qc.borland.com>. Follow the instructions on the Quality Central page in the “Bugs Report” section.
- **Quality Central menu command** on the main Tools menu of JBuilder (Tools|Quality Central). Follow the instructions to create your QC user account and report the bug. See the Borland Quality Central documentation for more information.

When you report a bug, please include all the steps needed to reproduce the bug, including any special environmental settings you used and other programs you were using with JBuilder. Please be specific about the expected behavior versus what actually happened.

If you have comments (compliments, suggestions, or issues) for the JBuilder documentation team, you may email jppubs@borland.com. This is for documentation issues only. Please note that you must address support issues to developer support.

JBuilder is made by developers for developers. We really value your input.

P a r t I I

Working in JBuilder

Introduction

This section of *Getting Started with JBuilder* contains information to help you use the JBuilder work environment to perform common tasks, and directs you to documentation that describes how to perform more complex tasks.

This section contains the following chapters:

- [Chapter 6, “Working with projects”](#)
Explains how to create a JBuilder project, describes what a JBuilder project is, and explains how to perform basic project management tasks.
- [Chapter 7, “Developing code”](#)
Provides an overview of the methods for generating and editing code in JBuilder, including the use of visual design tools, JBuilder’s many wizards, and the editor.
- [Chapter 8, “Using the JBuilder workspace”](#)
Describes how to use the different elements of the JBuilder workspace to perform common tasks, such as using bookmarks, searching for classes or text, and viewing files.
- [Chapter 9, “Compiling and building Java programs”](#)
Gives an overview of how JBuilder is used to compile and build.
- [Chapter 10, “Running, testing, profiling, and debugging source code”](#)
Gives an overview of how you can run, test, profile, and debug your source code from within JBuilder.
- [Chapter 11, “Managing application development”](#)
Describes the many ways that JBuilder helps you manage your code, from local version control to integrated version control systems. Includes information about refactoring code, visualizing code with UML, and internationalizing your applets and applications.

- [Chapter 12, “Developing applications”](#)

Discusses some of the types of applications that you can develop with JBuilder, and how JBuilder speeds the development process with supplied tools and features.

- [Chapter 13, “Archiving and documenting applications”](#)

Describes how JBuilder can help you prepare your application for deployment with archiving and documentation features.

Working with projects

Every time you do any work within JBuilder's IDE, you do it within a project. A JBuilder project is an organizational structure that holds all the files you need to perform the unit of work you define, including the directories those files are in and all the paths, settings, and resources they need. You decide what goes into your project.

The files your project requires can be in any folder. If you reorganize the structure of your project, it won't have any effect on the directories that exist on your system. The project structure is independent from the file structure.

You can set your project's properties (including paths, filters, build options, and formatting preferences) by choosing **Project | Project Properties**. Set default properties for future projects by choosing **Project | Default Project Properties**. You can also configure which technologies are visible in JBuilder's IDE (referred to as a project personality), while you work on your project. For more information on personality configuration, choose **Project | Project Properties | Personality** and click the **Help** button.

While you are creating and working in a project, use code auditing to make your code more robust and code templates to help you make the process quicker and easier. Code auditing provides numerous types of audits for style, documentation, performance, design, and possible code errors. Code templates offer Java, HTML, JSF, and Common code templates to use to quickly add common code and markup language elements to your files.

See also

- [“Using code audits” on page 60](#)
- [“Code templates” on page 138](#)
- “Using audits” in *Building Applications with JBuilder*

Here are some common tasks needed for working with projects:

- Creating projects
- Adding files to a project
- Updating paths
- Adding libraries, projects, and archives
- Removing and deleting files and directories
- Renaming projects and files

- Opening projects
- Switching projects
- Closing projects

For more detailed information about projects, and a comprehensive set of procedures for working with projects, see “Creating and managing projects” in *Building Applications with JBuilder*.

Creating projects

All of the work you do in JBuilder must be done within the context of a project. A JBuilder project is an organizational tool that includes all of your files in your application, the directory structure those files reside in, as well as the paths, settings, and required resources. Each project is administered by a project file, a descriptive file with a `.jpx` file type, and also includes a `.jpx.local` project file that consists of a history list of local user settings and some local user actions.

Creating a new project

The Project wizard (File|New Project) lets you quickly generate a new project. When you create a new project, you specify the name for the project and the directory where the project will be saved. The Project wizard lets you use the default project properties, or you can use an existing project as a template. In either case, you can override project paths and general project settings in the Project wizard.

For more information about creating projects, refer to “Creating and managing projects” in *Building Applications with JBuilder*. For information on setting properties for the default project template, refer to [Chapter 21, “Specifying default project settings”](#)

Creating projects from existing code

If you have some existing code that you would like to incorporate into a new project, you can use the Project For Existing Code wizard (File|New|Project|Project For Existing Code). The Project For Existing Code wizard creates a new JBuilder project from an existing body of work, deriving needed paths from the directory tree. For more information, refer to “Creating a project from existing files” in *Building Applications with JBuilder*.

Tip If your source files are not in directories according to package, use the Import Source wizard (Project|Import Source), to create a project from existing code.

Adding files to a project

JBuilder uses the automatic source path discovery feature to automatically add package nodes that are in your project’s source directory, if any exist. (You can turn this feature off, if you wish. See “Automatic source packages” for more information.)

If you’re not using automatic source path discovery, files and packages must be explicitly added to a project for JBuilder to treat them as part of the project. You can also choose to add JAR, library, or project files to the classpath of a project. Add files and packages to the current project using the Add To dialog box, accessed from the project pane’s toolbar button or context menu, “Add Files/Packages/Classes”.

JBuilder gives you several ways to create new Java files. When you use JBuilder’s wizards, they often create and add one or more Java files to your project. For example, the Class wizard creates a Java class and the Application wizard creates multiple Java files that together make up a Java application.

Note When you create a new file using the Create New File dialog box (File|New File), you must check the Add Saved File To Project check box to add the file to the active project.

For more information, refer to “Adding to a project” in *Building Applications with JBuilder*.

To create a new file or package for a web application (within the web application root directory or subdirectory) from the context menu of the project pane, see “Module directory” in the *Developing Web Applications*. This is a feature of JBuilder Enterprise.

Updating paths

Each project has numerous paths associated with it. Among other things, these paths tell JBuilder where to locate source files, where to output generated files, where to place backup files, and where to find documentation files or libraries. These paths and related settings are specified on the Paths page of the Project Properties dialog box (Project|Project Properties). You can directly add JAR, library, or project files to the classpath of a project.

To add archive, library, or project files to the classpath,

- 1 Choose Project|Project Properties to open the Project Properties dialog box.
- 2 Choose Paths in the tree on the left.
- 3 Click the Required Libraries tab on the Paths page.
- 4 Choose the Add button to display the Add To Project Classpath dialog box.
- 5 Choose the archive, library, or project to add to the project classpath.

For more information about paths, how they are constructed in JBuilder, and how you can manipulate them, refer to “Managing paths” in *Building Applications with JBuilder*.

Adding libraries, projects, and archives

JBuilder uses libraries to help find everything it needs to run a project, to browse through source, view Javadoc, use the visual designer, apply CodeInsight, and compile code. Libraries are collections of paths that include classes, source files, and documentation files. Libraries are static, not dynamic. Individual library paths are often contained in JAR or ZIP files but can also be contained in directories. Library configurations are saved in `.library` files. Library paths are also contained in project files.

You can configure existing libraries and add new ones in the Configure Libraries dialog box (Tools|Configure|Libraries), as well as add required libraries to your project on the Paths page of the Project Properties dialog box (Project|Project Properties). Anything added as a library appears on the class path.

You also can add project and archived library paths from the Paths page of the Project Properties dialog box. You can choose to add libraries in zipped or archived-file format, and you can add project files that contain library paths and definitions. When you add a project file, you’re adding all the libraries defined in the project file to your project. This can be a very important option if you have one project dependent on another project’s libraries.

Database drivers are added as libraries, using the Enterprise Setup dialog box (Enterprise|Enterprise Setup). This is a feature of JBuilder Developer and Enterprise.

For more information, refer to “Working with libraries” in *Building Applications with JBuilder*.

Removing and deleting files and directories

You can both remove and delete files and directories from your project. Removing files and directories removes them from your project but not from your system's drive. Deleting files and folders deletes them from your project and also erases them permanently from your system's drive. To remove files from the project, use the Remove From Project button on the project pane toolbar. To delete files or directories, use the context menu commands (Delete Directory, Delete File), of the project or file page of the project pane.

For more information, refer to “Removing folders, files, classes, and packages from a project” in *Building Applications with JBuilder*.

Renaming projects and files

Files and projects can be renamed easily by selecting the corresponding node in the project pane, and choosing File|Rename “<file_name>” or Project|Rename “<project_name>”. Alternatively, you can right-click files and nodes in the project pane and choose Rename “<node_name>”. For more information, refer to “Renaming” in *Building Applications with JBuilder*.

Note Renaming a file does not change the file type. To change the file extension, use File|Save As.


Caution Renaming projects and files does not change the package and file names referenced inside the code. JBuilder Developer and Enterprise provide rename refactoring. This changes all of the uses of the old name to match the new name.

See also

- “Refactoring code” in *Building Applications with JBuilder*

Opening projects

Projects can be opened by choosing File|Open Project, navigating to and selecting the project file in the File Selection dialog box, and clicking OK or pressing *Enter*.

 **Tip** If you have not cleared the file history, the project may be on your reopen list. Choose File|Reopen on the main menu or click the Reopen toolbar button on the main toolbar to view the project list.

For more information about opening projects, refer to “Opening existing projects and files” in *Building Applications with JBuilder*.

Switching projects

When you are working on more than one project, you can quickly switch between multiple projects with the mouse or with a keyboard command. Use the Window|Switch Project command from the main menu to open the Switch Project dialog box. The Switch Project dialog box contains a list of open projects to access. Switch to another open project by choosing a project from the list.

You can also use the project pane toolbar button, Select Project, to see the list of open projects and change to a different project. The keyboard command, *Ctrl+F5*, opens the Switch Project dialog box and change to another project.

Closing projects

When you are done working on your project, you can close it by choosing File|Close Projects. When you close a project, JBuilder prompts you to save any modified work. When you close a project, any files you opened while working on the project will also be closed.



Tip JBuilder adds the closed files to the reopen list. You can access the list from the main menu (File|Reopen), or from the main toolbar (File Reopen button).

See also

- “Creating and managing projects” in *Building Applications with JBuilder*
- “Working with project groups” in *Building Applications with JBuilder*
- “Managing paths” in *Building Applications with JBuilder*

Developing code

JBuilder provides a variety of tools for fast application development: wizards for automatically generating code, visual design tools for quickly creating UIs, code audits for finding code problems early, and coding shortcuts and templates for creating and correcting code. The wizards are available from the object gallery (File|New), and from many other sources, including the main Edit menu, context menus, toolbar buttons, and keyboard shortcuts.

The visual design tools include a UI designer, a Menu designer, a Data Access designer, and a Default designer and are accessible on the Design tab of the content pane. The code audit tool is enabled in the Project Properties dialog box, and the audit results appear in the message pane. Coding shortcuts and code templates are accessed through the editor.

Using wizards to generate code

JBuilder wizards save you time by automatically generating code for you. You can access these wizards from various menus, keystrokes, and buttons:

- File|New menu opens the object gallery
- Edit|Wizards menu displays context-sensitive wizard options and an Add submenu containing more wizard selections
- Project, structure, and content pane context menus (Add|<wizard name>)
- New button on JBuilder's main toolbar
- Tools menu
- *Ctrl+Shift+W* opens a menu containing context-sensitive wizard options (only Java files open in the editor using CUA key binding)



Depending on the node you select, you can access XML and Web Services wizards from the project pane context menu. Wizards available in the object gallery or from the project pane context menu are for creating new objects, while the other wizards are for working with existing objects.

Wizards provide numerous options for quickly creating code elements or performing tasks. You can do anything from copying files with the Import Source wizard (Project|Import Source) and overriding methods with the Override Method wizard (Edit|Wizards|Override Methods), to creating a new interface with the Implement Interface wizard (Edit|Wizards|Implement Interface).

You also can use the Delegate To Member wizard (Edit|Wizards|Delegate To Member), to create delegates for methods and use the Archive Builder wizard, (File|New) to create JAR files for numerous file types, including a runnable JAR file for an application. Choose one of the many wizards to help you streamline your coding process.

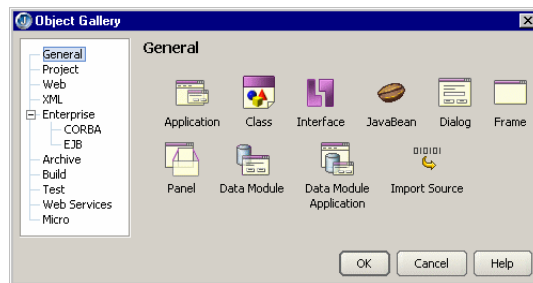
For more information about individual wizards, open the wizard, and click the Help button.

The Import Source wizard and creating a runnable JAR file for an application are features of JBuilder Developer and Enterprise.

Using the object gallery

The object gallery gives you access to wizards you can use to create basic implementations of many objects.

Figure 7.1 Object gallery



To use a wizard in the object gallery, choose File|New or click the New button on the main toolbar to open the object gallery. Double-click a wizard icon or select a wizard icon, then click OK. JBuilder opens the associated wizard and creates the skeletal code in an appropriate file which it adds to your project.

Wizards that are grayed out in the object gallery are disabled. A message explaining why the wizard is not available is displayed below the wizard name. Some wizards only become enabled after creating or opening a project or a specific type of file or by enabling a specific server. Also, wizards can be disabled if they are not available in a JBuilder edition. Wizards vary by JDK version and JBuilder edition.

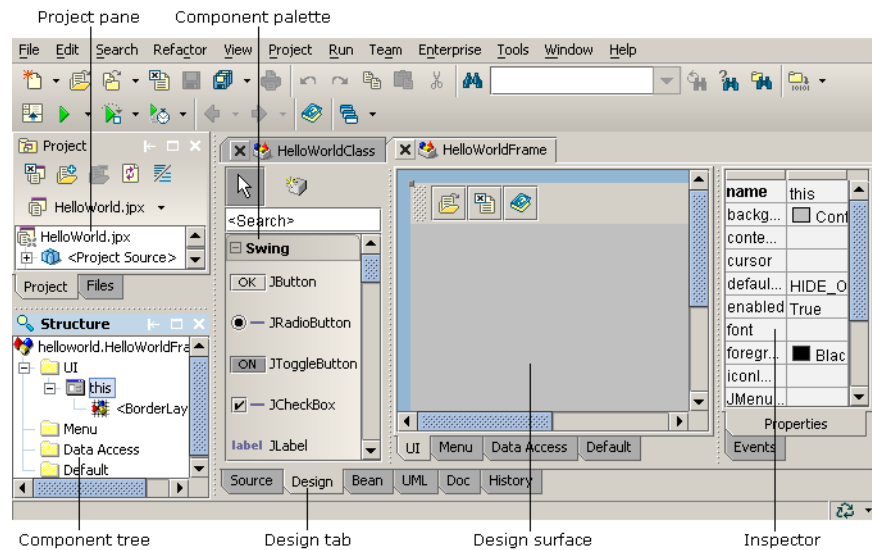
See also

- “Using the Archive Builder” in *Building Applications with JBuilder*

Working with the visual design tools

The JBuilder visual design tools consist of a component palette, an Inspector, several designers, and a component tree. To access the design tools, you must open a source file in the content pane, then click the Design tab.

Figure 7.2 JBuilder in design view



Visual designers share the design surface on the Design page of the content pane: the UI designer, the Menu designer, the Data Access designer, and finally, the Default designer for components that don't quite fit the other three categories.

You use the UI Designer to build user interfaces that contain visual elements, such as list boxes and buttons. When you first click the Design tab after opening a file in the content pane, JBuilder displays the UI designer by default. UI components appear in the **UI** folder of the component tree.

For information about using the UI designer, see "Introducing the designer" in *Designing Applications with JBuilder*.

The Menu designer is used to create menus. Menu components appear in the **Menu** folder of the component tree. To access the Menu designer, choose the Menu tab.

For information about using the menu designer, see "Designing menus" in *Designing Applications with JBuilder*.

The Data Access designer is used to create data access components for database applications. Data access components initially appear in the component tree. Click the Data Access folder in the component tree to access any available components. For more information about the Data Access designer, see "Data Access designer" in *Designing Applications with JBuilder*.

The Default designer provides tools for elements that aren't included with the other designers, such as dialog boxes. Click on the Default tab to view the available default components. See "Default designer" in *Designing Applications with JBuilder* for more information.

Editing files

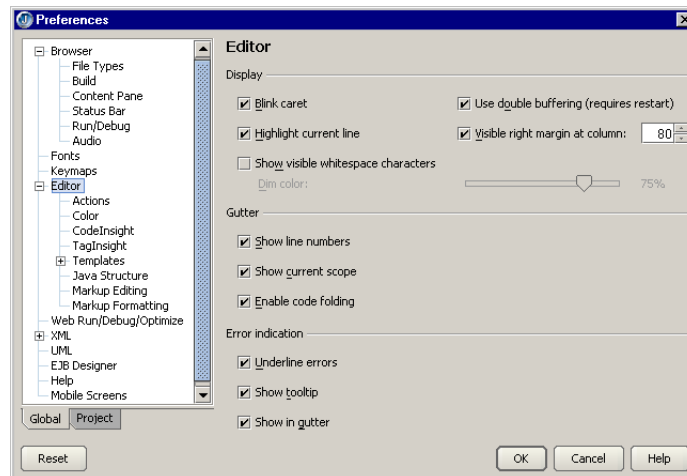
Editor features vary by JBuilder edition

The editor offers a variety of productivity features, such as keyboard shortcuts, customizable editor keymappings, and coding shortcut tools (CodeInsight, ErrorInsight, SyncEdit, JavadocInsight, code folding and scoping). The editor also offers customizable code templates, searching and identifying capabilities, a tagging tool for markup languages (TagInsight), and printing.

To edit your source code, double-click a file in the project pane to open it in the editor. A tab for the open file is displayed at the top of the content pane, making it easy to navigate between open files in a project. The editor provides unlimited line length editing for your files, adjusting the editor width and scrollbar action to the longest line in your file.

The editor is fully customizable. Use the Editor page of the Preferences dialog box (Tools|Preferences|Editor), to customize the editing environment. You can control many editor settings, including the display and actions for the editor, structure pane, CodeInsight and TagInsight tools, as well as settings for text, keyboard actions, color, and code templates.

Note To customize the editor, you can also access the Editor Preferences dialog box by right-clicking in the editor and choosing Editor Preferences.



Using code audits

Code auditing is a feature of JBuilder Developer and Enterprise

Using the code auditing tool helps you discover potential problems early in the development cycle, preventing more complex code problems later. You can turn on code audits and select specific problems to audit in the Project Properties dialog box (Project|Project Properties|Code Audits). The dialog box displays a description of the problem and an example of incorrect and correct code for each type of audit. Code auditing provides auditing for style, documentation, performance, design, and potential code errors.

The auditing results for each open file appears immediately after you enable code audits. Code auditing is turned off by default. Remember to turn on code audits for each open project. The audit process is ongoing while you create and edit your code. However, if your code contains errors, audits may be inaccurate.

Code audits appear in a Warnings folder in the structure pane. Expand the folder to see the list of code audit warnings and click on a warning to navigate to the associated problem in the editor. The warnings icon appears adjacent to the problem area in the editor.

See also

- “Using audits” in *Building Applications with JBuilder*
- “Audit definitions” in *Building Applications with JBuilder*

For more information about the features provided by JBuilder’s editor and how to use them, refer to [Chapter 15, “Working in the editor”](#), [Chapter 16, “Using code shortcuts”](#), and [Chapter 17, “Customizing the editor”](#).

Using the JBuilder workspace

JBuilder uses one window to perform most of the development functions: editing, visual designing, navigating, browsing, compiling, debugging, and other operations. This window is JBuilder's workspace, and it contains several panes (content, project, files, classes, structure, message), and status bars for performing these development functions. This section describes how to use the different elements of the JBuilder workspace to perform common tasks.

The most obvious way to run commands and perform tasks in JBuilder is using the menus on the menu bar, and the buttons on the main toolbar. The main toolbar provides a subset of the most frequently used commands. In addition to the menu bar and main toolbar, there are a variety of shortcuts available from different parts of the workspace.

The shortcuts include numerous context menus available when you right-click on various areas of the work environment, including panes, tabs, and tool, title, and status bars. The shortcuts also include colorful icons in the editor and workspace that provide a variety of actions. See the following topics in this section for descriptions about how to use the JBuilder workspace to perform your work more quickly and productively.

- [“Searching in the workspace” on page 66](#)
- [“Searching in the workspace” on page 66](#)
- [“Using the content pane” on page 67](#)
- [“Using the project pane” on page 69](#)
- [“Using the files pane” on page 73](#)
- [“Using the classes pane” on page 74](#)
- [“Using the structure pane” on page 75](#)
- [“Using the message pane” on page 79](#)
- [“Using the status bars” on page 81](#)
- [“Monitoring memory use” on page 82](#)

Navigating the workspace

JBuilder provides shortcuts and other time saving devices to help you navigate the workspace more quickly.

Keyboard navigation

Keyboard shortcuts help you perform certain tasks faster and more easily. JBuilder lets you map your keyboard to support the following editor emulations:

- CUA
- Emacs
- Brief
- Visual Studio
- Macintosh
- Macintosh CodeWarrior

The keymappings for each of these emulations includes predefined keyboard shortcuts for editor operations and JBuilder commands. Some of these operations include opening wizards, navigating to methods, collapsing and expanding code, showing white space characters, and formatting entire files.

You can select a keymap from the list on the Keymaps page of either the Preferences dialog box (Tools|Preferences|Keymaps) or the Editor Preferences dialog box, accessed by right-clicking in the content pane and choosing Editor Preferences.

Not only can you view and modify the keymappings associated with a specific keymap, you can also import, copy, find, save, and create new keymaps. To view and modify keymappings, click the Edit button on the Keymaps page of the Preferences dialog box or the Editor Preferences dialog box to open the Keymap Editor dialog box.

See also

- “Keymaps for editor emulations” in online help
- [“Adaptive techniques supported in JBuilder” on page 42](#)
- [“Setting keymapping preferences” on page 151](#)
- [Chapter 16, “Using code shortcuts”](#)

Using the history lists

JBuilder maintains a list of the projects and files you have opened. Files and projects are added to the list when they are closed. This history list lets you quickly return to previously viewed projects or files. The history list appears as a submenu when you choose File|Reopen or click the Reopen button on the main toolbar.

JBuilder also maintains a list of the files visited by JBuilder discovery features and of the files you visited them from. Items are added to this list only when you perform a JBuilder action that takes you away from the current file view, such as Find Definition or Find Classes.

You can navigate back and forth between visited files by choosing Window|Back and Window|Forward or the Back and Forward buttons on the main toolbar. These take you to the line and column of the visited code element.

See also

- “Reopening projects and files” in *Building Applications with JBuilder*

Using View navigation buttons

When you open HTML, XML, or JSP files in the content pane, the view pane provides navigation buttons for forward and backward navigation. (Click the View tab to open the view pane.) The Back and Forward view pane buttons also display a drop-down history list where you can choose to quickly navigate between recently visited file locations and sites.

Using favorite links



JBuilder lets you create and organize a list of favorite drives or directories to speed navigation. File and path selection dialog boxes, such as the Open File dialog box, and the Files tab of the project pane include a Favorites button for adding and organizing frequently visited drives or directories to your list of favorites.

In file and path selection dialog boxes, the Favorites links appear as a collection of buttons. The icon on these buttons is a folder with a heart. The location of the Favorites list in selection dialog boxes depends on your operating system. For instance, if you're using Windows, they're in a pane on the left of the dialog box. To put the dialog box selection in a favorite drive or directory, simply click the corresponding button in the selection dialog box.

Open the files pane by clicking the Files tab in the project pane. In the files pane, favorite folders are displayed with a heart icon beneath the Project folder. These nodes can be expanded to provide quick access to the contents of frequently visited drives and directories. To remove a favorites folder from the files pane, right-click a favorites folder and choose Remove From Favorites.

Adding favorite links

You can add a favorite link from an open file selection dialog box or from the files pane.

- 1 Navigate to the directory you want to add from the files pane or the dialog box.
- 2 Click the Favorites button in the upper right corner of the dialog box or click the Favorites toolbar button on the files pane toolbar.
- 3 Select Add To Favorites.

The Add To Favorites dialog box opens.

- 4 Type in a name for the link.
- 5 Click OK or press *Enter*.

The new favorite link is added to the list of favorites.

Organizing favorite links

For more convenient access, you can organize the order of the added favorite links in an open file selection dialog box or in the files pane.

- 1 Click the Favorites button in the upper right corner of the dialog box or click the Favorites button on the files pane toolbar.

The Organize Favorites dialog box appears.

- 2 Select a favorite from the list.
- 3 Rename, remove, or move the favorite up or down in the list.

The sort order in the dialog box determines the sort order in the favorites list.

- 4 Click OK or press *Enter* when done.

For more information, see "Using the File browser" in *Building Applications with JBuilder* or click Help in a file or path selection dialog box.







Using bookmarks

Use bookmarks to mark lines of code in the editor that you want to return to later. The bookmarks are persistent, project-wide, and for all file types. You can quickly move to your bookmarked code in the editor by choosing the Bookmarks command (Search | Bookmarks), or use the bookmark shortcuts you create in the Add Bookmark dialog box. The Bookmarks dialog box lists all the bookmarks you have set in your code.

See [“Using bookmarks in the editor” on page 125](#) for more information.

Using window action commands

When your workspace is configurable, choosing View | Window Actions or pressing *Alt+F10* opens a Window Actions context menu to control how to configure the active pane (the pane with focus) in the workspace. The following table describes the window action commands.

Icon	Command	Description
	Maximize	Sizes the pane to fill the workspace.
	Restore	Restores the maximized pane to its original size and location.
	Iconify	Reduces the pane to an icon on the left border of the workspace. Click the icon on the border to open or close the pane.
	Dock	Reattaches an iconified or undocked pane to the workspace.
	Undock	Detaches the pane from the workspace.
	Move	Lets you move the undocked pane with the arrow keys on the keyboard. Press <i>Enter</i> when you have positioned the pane where you want it.
	Size	Lets you use the arrow keys on the keyboard to resize the pane. Press <i>Enter</i> when you have sized the pane the way you want.
	Close	Closes the pane or the selected tab of the pane.

The project, structure, and message panes provide buttons on their title bars for some window action commands.

Note If your workspace is not configurable, choosing View | Window Actions opens the context menu for the active file in the content pane. The context menu contains some of the window action commands.

For more information about moving and sizing panes, see [Chapter 19, “Configuring your workspace.”](#)

Searching in the workspace

The Search menu on the menu bar provides access to numerous commands for locating and navigating to specified text, code elements, and other elements of your project files. Different parts of the workspace provide additional search capabilities.

Searching trees

You can search/navigate trees displayed in the project, structure, and message panes by moving focus to the pane and typing. The closest match is highlighted in the pane. The *Down* and *Up* arrow keys move the focus to the next and previous matches as you type, and the period key expands the tree.

Finding classes

The ClassInsight function (*Ctrl+Alt+Space* or *Ctrl+Alt+H*) simplifies searching for a class on the class path. ClassInsight invokes the ClassInsight dialog box, which can be used to find classes, and insert them into your code. ClassInsight lets you search for a class by name, without needing to know the package.

The ClassInsight functionality is used under different names in different places in JBuilder. For example, the Classes page of the Add To Project dialog box (Project Add Files/Packages/Classes) lets you search for source files to add to your project, and the Search page of Find Classes dialog box (Search/Find Classes) quickly locates a Java class to open in the editor.

For more information, see [“CodeInsight” on page 130](#).

Searching in the editor

The editor provides numerous ways to find and replace specific text in a file. Search commands for finding text are located on the Search menu, as buttons on the main toolbar, and on the editor’s context menu. For more information about finding text in the editor, refer to [“Finding text in the editor” on page 117](#).

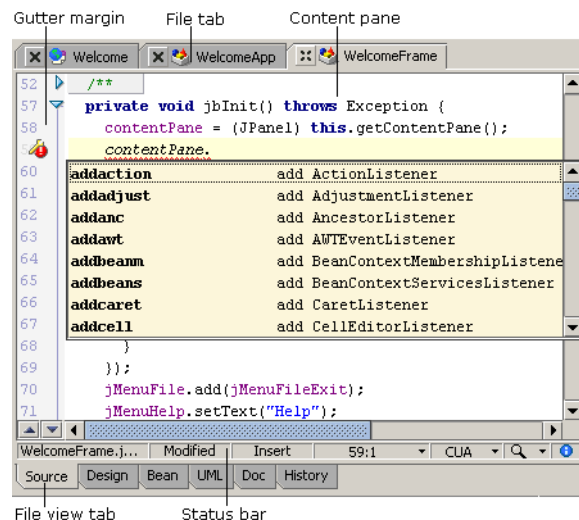
JBuilder also provides commands and tools for searching for code elements, such as variables, methods, classes, or overriding methods in the files you are editing. You can also find all source files that use a selected symbol. For more information about searching for code elements, refer to [“Finding code elements and definitions” on page 119](#).

Using the content pane

The content pane displays all opened files in a project. To open a file in the content pane, double-click it in the project pane, or select it and press *Enter*. The name of each opened file is displayed on a tab in the content pane. When you click a tab, that file becomes the current file.

The content pane provides access to various file views and operations by way of the file view tabs shown at the bottom of each file window. These tabs offer a variety of different views, including the source, design, bean, UML, doc, and history views. When you choose the Source tab of the content pane, the editable pane is then referred to as the editor. The available tabs vary by JBuilder edition.

Figure 8.1 Content pane showing the source view (provides editor)



Using the content pane file tabs

File tabs contain the names of the open files in a selected project in the content pane. Only the file tabs of open files in the active project are visible. One file at a time is active in each open editor.



Each file tab has a Close File button that serves two purposes:

- Allows you to close the file with one click on the button.
- Indicates whether the file has changed since it was last saved. If so, it looks like this, and you will be prompted to save the file before closing it. When you hover your cursor over the file tab, a tool tip appears with information about the file name, path, and modified status.

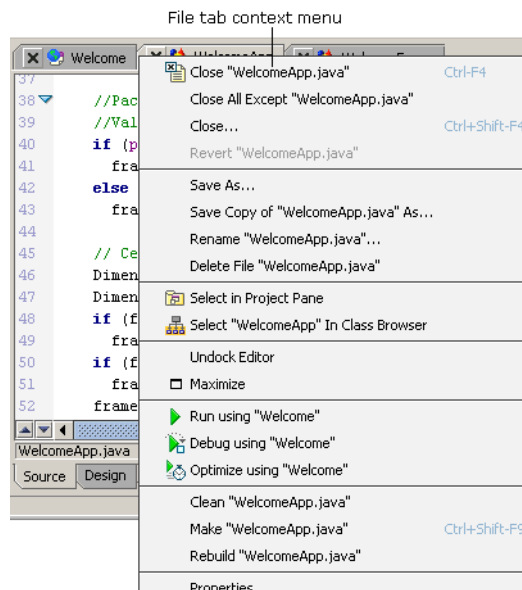


If the file is a read-only file, the content pane file tab displays an icon indicating read-only status if the file type icon is not displayed. The file tab displays the file name. JBuilder's title bar displays the full file path, name, and extension. Hold the cursor over the file tab and the file name and full file path displays.

You can customize the tab labels several ways: orientation, label type, and insertions. Open the Content Pane page of the Preferences dialog box (Tools|Preferences|Browser|Content Pane), to customize the file tabs to your preference. Click the dialog box's Help button for more information about these options.

Using the file tab context menu

You can access a comprehensive context menu from the content pane's file tabs. The context menu commands are dependent on the file type, and in certain cases, file content. For example, Run menu commands only appear for Java files that contain a `main` method. Right-click on a content pane's file tab to view all of the context menu options. The Select In Project Pane menu command provides rapid navigation to the project pane location of the open file.



Resizing the content pane

You can resize the content pane for easier viewing using any of the following methods:

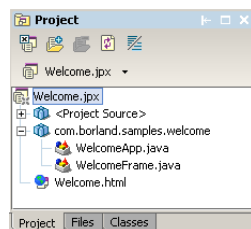
- Drag the splitter bar between the project pane and the content pane to the size you want.
- Choose View|Maximize Content Pane from the main menu, or press *Ctrl+Alt+Z* if using the CUA key bindings, to expand the content pane fully and to hide all the other panes.
- Double-click the file tab to expand the content pane to maximum size.

See also

- [“Content pane” on page 14](#) for more information about the content pane and what it can display
- [Chapter 15, “Working in the editor”](#) for information about working with source files in the content pane

Using the project pane

The project pane displays the structure of your project in the form of a tree. You can freely navigate the project tree, click and multi-select files, or drag and drop files between parent nodes in the project pane. You can drag files into a package from the project pane or from the desktop. You can also move the project pane and redock it to another location by dragging it by the Project tab.



The package nodes are sensitive to the files being dropped, respond appropriately, and sort the files alphabetically. You can also delete packages and directories from the project tree if they aren't included in open projects or project groups and if the tree's open node can't be saved or closed.

You can open as many projects in the project pane as you wish, but only one project is active at any given time. The active project appears as the selected item in the project pane drop-down list.

Project, package, and file nodes not only include informative tool tips, but also can include an icon and text to designate file states: modified, read-only, and VCS, among others. You can customize the corresponding decorations in the Project Properties dialog box (Project|Project Properties|Decorations). See [“Using file status decorations” on page 70](#) for more information.



Tip Automatic source packaging ensures that your added files' resources are included correctly. After adding new source files to the project, select the Refresh button on the project toolbar to update the automatic source packages list.

To learn more about working with projects, see “Creating and managing projects” in *Building Applications with JBuilder*.

Using the project pane toolbar

The project pane contains a useful toolbar that includes buttons for performing various project pane actions. Each button is identified by tool tip text when you hover the cursor over it. The following buttons help you quickly work with projects and files in the project pane.

- Close Project
- Add Files/Packages/Classes
- Remove From Project
- Refresh
- Show Decorated Only
- Select An Open Project To Work With

Using file tool tips

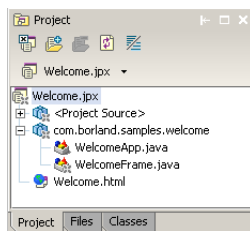
The project and files panes provide useful, multi-line tool tips. The tool tips include the full file name and path, the date the file was last modified, and the file size in bytes. (The content pane file tabs also display multi-line tool tips.)

You can quickly view the file's tool tip in the project or files pane.

- 1 Click the project or files tab to open the project or files pane.
- 2 Hold the cursor over the file name to display the full file path, name, date modified, and size in bytes.

Using file status decorations

When you are editing files, you can enable decorations (icons), for modified files, packages, and projects that appear in the project pane, VCS workspace, and VCS repository. You can also use modifier decorations for files not in an active project, not in a repository, and for read-only files. The project pane toolbar includes a Show Decorated Only icon for quickly turning the decorations on and off. You can also customize decoration options in the Project Properties dialog box.



To customize file, package, and project status decorations,

- 1 Choose Project|Project Properties to open the Project Properties dialog box.
- 2 Choose Decorations in the tree in the left pane to open the Decorations page.
- 3 Choose from the following decoration options for files, version control workspaces, or version control repositories:
 - Enable Icon Only
 - Enable Project Pane Tree Text Only
 - Enable All
 - Disable All

- 4 Enable and set the timing for the State Check Interval to refresh the decorations. The default is 15 minutes.

- 5 Click OK.

The decorations appear automatically in the project pane and on the content pane file tabs.

Navigating the project tree

To expand a package and view its descendants, click the expand icon next to it in the project pane. Click the expand icon again to collapse the package and hide its descendants.

You can select and expand an archive file, such as a JAR, WAR, or EAR, in the project pane and see the files it contains. This is a feature of JBuilder Developer and Enterprise.

Tip You can quickly search for a particular file or package in the project pane by moving focus to the project tree and beginning to type.

Viewing files

To view a file in a project, so you can examine it and edit it, double-click a file node in the project pane or drag the file from the project pane and drop it on the content pane. You can also select one or more files in the project pane, and press *Enter*. JBuilder uses the appropriate viewer for that file type to display the file. For example, JBuilder uses its image viewer for a *.gif* or *.png* file, and its browser viewer for an HTML file.

If a file is in a package, expand the package node by clicking the plus sign (+) adjacent to the package icon and name. Once you can see all the files in the package, double-click the one you want to view.

Each open file of a project appears in the content pane. A tab with the file name appears for each open file in your project. The full file path, name, and extension display in JBuilder's title bar.

- ✕ You can quickly close any files displayed in the content pane by clicking the Close icon on the left side of the file tab. You can also right-click the tab to discover other options for closing and managing the file in the project.

To learn about a way to switch quickly between a large number of open files in the content pane, refer to “Switching between files” in the “Creating and managing projects” chapter of *Building Applications with JBuilder*.

Tip To view or create the Javadoc documentation for a package, double-click the package node in the project pane. The Javadoc appears in the content pane.

Switching between open projects

If you open more than one project, only one active project is visible at a time. To activate another open project, do one of the following:

- Click the project selector drop-down menu



in the project toolbar above the project pane, and choose the project you want to make active.

- Choose Window from the menu bar and select the project from the Window menu.

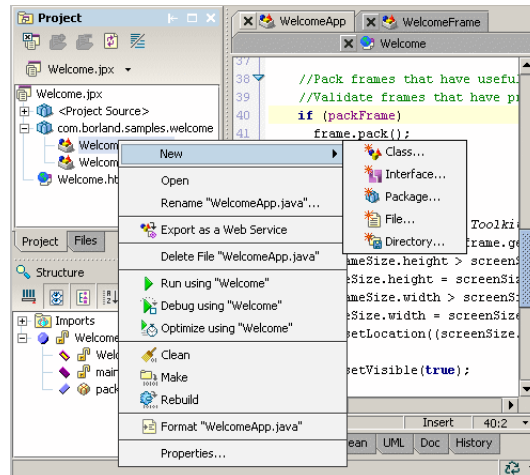
The project you selected appears and the previous project is no longer visible, but you can access it again from either menu.

Using the project pane context menu

Context menu commands vary by JBuilder edition

You can right-click and select items from the project pane's context menu, without ever opening a file. The context menu selections vary by the file type selected.

Figure 8.2 Project pane context menu



Adding new directories is a feature of JBuilder Developer and Enterprise

Note

Adding new files, folders, directories, or packages

To add a new file, folder, directory, or package from the project pane context menu,

- 1 Right-click the project, file, or package node in the project pane.

The source of the context menu is the package and any file on the source path of the project (which can be the child of the package, the project, or the directory view).

- 2 Choose New.

Depending on the file or package you choose, menu options include:

Menu options	Menu actions
New/Class	Opens the Class wizard.
New/Interface	Opens the Interface wizard.
New/Package	Opens the Create New Package dialog box.
New/File	Opens the Create New File dialog box.
New/Directory	Opens the Create New Directory dialog box.
New/Folder	Opens the Create New Folder dialog box. Be aware that project folders are for organizational purposes only and do not correspond to directories on disk.
New/Directory View	Opens the Select Directory dialog box.

- 3 Fill in the wizard or dialog box fields for the file or package.

For more information about creating and adding to JBuilder projects, see [Chapter 6, "Working with projects."](#)

Removing files and classes



To remove a file or class from a project, click the Remove From Project button on the project pane toolbar. You can also right-click on some of the file nodes and choose the Remove From Project context menu option.

Note Packages and files added dynamically by the automatic source package discovery feature can't be removed.

Creating new files or packages for a web application

To create a new file or package for a web application (within the web application root directory or subdirectory) from the context menu of the project pane, see "Root directory" in the *Developing Web Applications* for information. This is a feature of JBuilder Enterprise.

Running an application from the project pane

This is a feature of JBuilder Developer and Enterprise

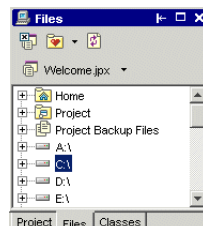
The Archive Builder wizard and the project pane context menu make it possible to run an application from the project pane. When you create a JAR file for an application using the Archive Builder wizard that includes the main class configuration, you can make and run the application using the project pane's context menu commands.

See also

- ["Using wizards to generate code" on page 57](#) to learn about JBuilder's wizards
- "Using the Archive Builder" in *Building Applications with JBuilder* to learn about creating JAR files
- ["Searching trees" on page 66](#) to learn more about searching in the project tree
- "Filtering packages" in *Building Applications with JBuilder* to learn about how you can filter packages to exclude them from the build system

Using the files pane

The files pane, also called the file browser, provides quick navigation to drives, directories, folders, and files on your computer and network so you can easily browse through all the files on your system. It also provides a toolbar for quickly closing, selecting, refreshing, and customizing. To display the files pane for file browsing, click the Files tab at the bottom of the project pane.



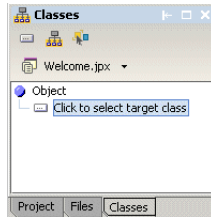
The file browser displays the directory for your current project, the JBuilder home directory, and all available drives. Double-click any file you want to open. To see the contents of a folder or drive, expand the corresponding node in the project pane. You can also delete package and directories from the File browser tree if they don't include open projects or project groups, and the tree's open node can't be saved or closed.

See also

- ["Using favorite links" on page 65](#) for more information about customizing the files pane

Using the classes pane

The classes pane, also called the class browser, provides class or interface browsing of inheritance structures. The class browser displays child-parent relationships for any class or interface within the specific inheritance hierarchy. When an interface is the target, the classes pane also displays *extends* and *implemented by* nodes in the tree that show interfaces that extend or classes that directly implement the interface.



You have to turn on the classes pane to display it in the project pane area. The classes pane does not appear by default. You can open the classes pane from the View menu, the structure pane context menu, and the editor's file tab context menu. The following instructions describe how to display the classes pane.

To display the classes pane,

- 1 Choose the View menu.
- 2 Choose Panes|Classes.

The classes pane with a Classes tab opens in the northwest corner of the IDE.

- 3 To close the classes pane, uncheck Classes on the View menu (View|Panels|Classes).

When you open the classes pane from the View menu, your initial view of the class pane shows only "Object" and a line of text, "Click to select target class". Click the text to open the Select Class dialog box so you can search or browse for the class hierarchy that you want to view in the classes pane.

The classes pane provides additional toolbar buttons: Find Class, Select in Class Browser, Show Interfaces, and Select An Open Project To Work With. The buttons help you to quickly access classes, display class hierarchy of the opened file, show interfaces, and select open projects.

Displaying class hierarchy

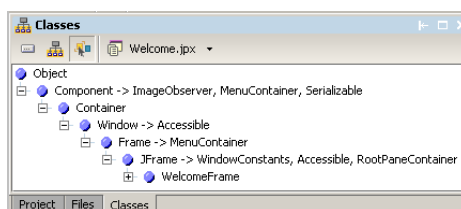
The classes pane displays either the inheritance structure for the top-level target class or for the target class where the cursor is located in the editor. Use the context menu or the classes pane toolbar button to choose a target class to display in the classes pane.

To display class hierarchy in the classes pane,

- 1 Right-click a Java file tab in the editor to display the top-level target class.
- 2 Choose Select In Class Browser.
- 3 Or, you can click the Select Class In Class Browser button on the Classes pane toolbar.



The classes pane displays the hierarchy of the selected class.



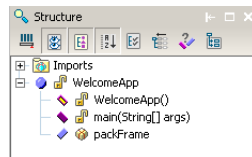
Navigating in the classes pane

You can easily change the class browser or editor view from the classes pane.

- 1 Single-click a class or interface in the class browser tree to open an associated file in the editor. (The class browser view and the class target don't change.
- 2 Double-click a class or interface to select the class in the editor and to make it the target in the class browser.

Using the structure pane

The structure pane displays the structure of the file currently selected in the content pane. This structure is displayed in the form of a tree showing all the members and elements in the file. Identifiable icons represent each member's type and visibility state.



When appropriate, the structure pane displays: an Errors folder containing any syntax errors, an Imports folder containing a list of imported packages, a To Do folder containing Javadoc `@todo` tag comments, a Markup Errors folder containing errors for markup languages, a Javadoc Conflicts folder containing conflicts in Javadoc text, and a Warnings folder containing code audit warnings. The Javadoc Conflicts folder is a feature of JBuilder Developer and Enterprise.

You can quickly search for an element in a file by moving the focus to the tree in the structure pane and starting to type the name of the element you want. Also, when you click on a structure pane element, the editor automatically moves to and highlights the corresponding element in the source code. For more information, see [“Searching trees” on page 66](#).

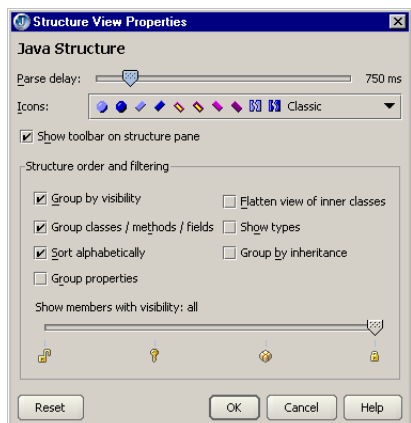
The structure pane offers sorting, filtering, viewing Javadoc, source code navigation, and viewing error messages. See the following topics for more information.

- [“Sorting in the structure pane” on page 76](#)
- [“Filtering in the structure pane” on page 76](#)
- [“Changing structure pane icons” on page 77](#)
- [“Viewing Javadoc information in the structure pane” on page 77](#)
- [“Using the structure pane to navigate in the source code” on page 78](#)
- [“Viewing structure pane error messages” on page 78](#)
- [“Navigating Java code hierarchy in the structure pane” on page 79](#)
- [“Getting Java help from the structure pane” on page 79](#)

Sorting in the structure pane

You can change the sorting order, filtering, and parse delay of the structure pane in the Structure View Properties dialog box. Right-click the structure pane and choose Properties to open this dialog box and modify the options. You can also expand or collapse the structure pane tree for easier viewing and navigation. Right-click on a structure pane element and choose Expand Children or Collapse Children.

Figure 8.3 Structure View Properties dialog box



The structure pane settings are also available on the Java Structure page of the Preferences dialog box (Tools|Preferences|Editor|Java Structure). You can also access this page from the editor. Right-click in the content pane and choose Editor Preferences. For a description of the available options, click the Help button on the Java Structure page of the Preferences dialog box or the Editor Preferences dialog box.

Filtering in the structure pane

The structure pane toolbar displays a variety of filtering icons to help you quickly find various class elements. The filters allow you to hide, show, and order structure pane data in a variety of defined ways. Multiple filters can be applied, and the resulting elements in the structure pane are the sum of the selected filters. The Group By Visibility, Group Classes/Methods/Fields, and Sort Alphabetically filters are applied by default.

Table 8.1 Structure pane filters

Icon	Description
	Adjust visibility filter: opens a slider to set structure pane visibility to display a combination of public, protected, package, or private class elements.
	Group by visibility: groups the class elements by visibility.
	Group Classes/Methods/Fields: groups the class elements by class, method, and field.
	Sort alphabetically: sorts elements alphabetically from a to z.
	Group properties: groups elements by individual properties.
	Flatten view of inner classes: displays all class elements, including inner class elements.
	Show types: toggles visibility of method return types and field types.
	Group by inheritance: groups elements by inherited classes.

Changing structure pane icons

Structure pane icons decorate code members and signify their type and visible state. You can choose to use different icons for code member types to make it quicker and easier to recognize them.

Figure 8.4 Structure pane icons



To change structure pane icons,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose Editor|Java Structure to open the Java Structure page.
- 3 Click the Icons drop-down menu and choose from the following icon types:
 - Classic
 - Classic Updated
 - Geometric
 - Letters
- 4 Click the OK button to update the structure pane icons and close the Preferences dialog box.

Viewing Javadoc information in the structure pane

The structure pane provides a variety of Javadoc information, including Javadoc “todos” and conflicts. Javadoc `@todo` tags in Javadoc comments display in the structure pane in a To Do folder. The `@todo` tags and the Todo folder in the structure pane make it easier to keep track of tasks you want to complete later.

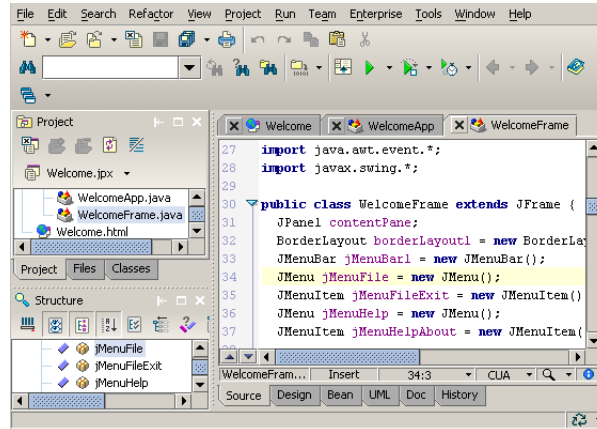
Javadoc conflicts are displayed in a Javadoc Conflicts folder in the structure pane. (This is a feature of JBuilder Developer and Enterprise.) To easily resolve Javadoc conflicts, right-click on the Javadoc Conflicts folder or the individual Javadoc file and choose Fix Javadoc Conflicts.

See also

- [“Adding todo tags in the editor” on page 124](#)
- [“Resolving Javadoc conflicts” on page 123](#). This is a feature of JBuilder Developer and Enterprise.
- “Conflicts in Javadoc comments” in *Building Applications with JBuilder*. This is a feature of JBuilder Developer and Enterprise.

Using the structure pane to navigate in the source code

In addition to viewing the structure of the class, the structure pane is a quick way to navigate to a class, method, or member in the source code. When you select an item in the structure pane, the content pane scrolls to the line that defines the item and highlights the line.

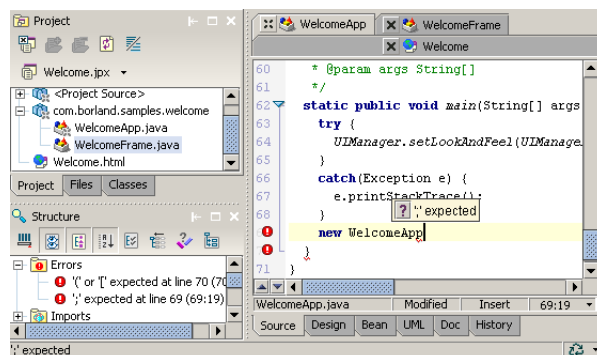


You can also use the structure pane for drilling down into ancestor classes and interfaces. To see the .java file for an ancestor class, an interface, or the type of a variable shown in the structure pane, double-click it (or select it and press *Enter*). JBuilder shows the file in the structure and content panes. To search a package, select SearchFind Classes or double-click the package in the structure pane and select a class in the dialog box to open it in the content pane. To return to the file you were viewing, click the arrow of the Back button on the main toolbar to choose from the file history list.

Viewing structure pane error messages

Messages about syntax errors are displayed in an Errors folder in the structure pane. Expand the folder and select an error message. The corresponding line of code is highlighted in the editor.

Syntax errors are underlined in the editor. Put your cursor on the error to see a tool tip that indicates the nature of the error. The tool tip contains a question mark button. Click the question mark button to open the compiler error message help file.



See also

- “Error messages” in *Building Applications with JBuilder*
- “ErrorInsight” on page 136

Navigating Java code hierarchy in the structure pane

The structure pane can help you navigate into ancestor classes and interfaces of .java files. To navigate into the class or interface hierarchy, just double-click the class or interface. JBuilder displays the ancestor file in the project pane, content pane, and structure pane.

See also

- [“Using the structure pane to navigate in the source code” on page 78](#)

Getting Java help from the structure pane

You can view Java documentation from the structure pane. If you’re in the source code or a UML diagram, select a class, interface, or field and press *F1*. If you’re in a designer (except the EJB designer), select a component in the component tree and press *F1*.

See also

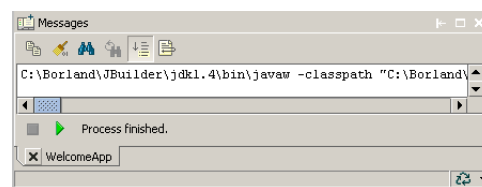
- [“Using Javadoc QuickHelp” on page 124](#)

Using the message pane

At times a tabbed message pane appears at the bottom of JBuilder’s IDE for displaying user messages from different operations, such as:

- Building
- Running
- Debugging
- Searching
- Refactoring code
- Unit testing
- Version control
- Managing project requirements

Figure 8.5 Message pane



JBuilder generates a new tab at the bottom of the message pane for each new process. These tabs allow you to interact with the running process. An icon on the tab indicates that a process is active and console output is possible through the message pane text area. A Build tab displays during compile if there are errors or warnings.

For some of the different operations, the message pane includes the following useful toolbar actions:

- Copy — Copies selected or all of the output content.
- Clear All — Clears all of the output content.
- Search — Searches in the output.
- Search Again — Searches again in the output.
- Auto Scroll — Automatically scrolls down when user operations generate output.
- Word Wrap — Wraps the output by the words and pane size.

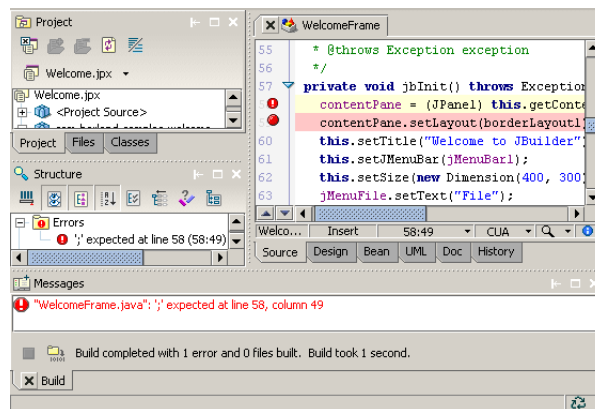
Some operations provide toolbar buttons on the tab that let you stop or restart the current process. The restart button associated with a process reuses the settings in effect when the tab was initially created, ignoring any subsequent changes you may have made. Using this feature, you can set up several different configurations to be repeatedly tested.

You can run two or more operations simultaneously on the same project or different projects. A separate message tab is created for each new operation started with the main toolbar Run and Debug buttons.

To start a new run or debug operation, make sure the project you want to run or debug is the active project in the project pane or the file you want to debug is the current file in the content pane. Then use the buttons on the main toolbar to start the operation.

During some operations, such as running, debugging, version control, and refactoring, the message pane contains a status bar to advise you of the operation or result of the current process.

Here's an example of JBuilder's IDE with a message pane at the bottom reporting a syntax error:



Copying message pane text

For some operations, the message pane contains a helpful toolbar button that you can use to copy text from the message pane. You can either click the toolbar Copy button to copy all of the text or any text you select, or you can use the context menu.

- 1 Select the text in the message pane.
- 2 Right-click the selected text.
- 3 Choose Copy Selected.

With no text selected, you can choose Copy All from the context menu to copy all of the text in the message pane.

Formatting message pane text

Use the Preferences dialog box to customize message pane text. The message pane appears during various operations. You can change the format type and size of the output text, as well as enable anti-aliasing.

- 1 Choose Tools|Preferences.
- 2 Choose Fonts in the tree on the left to open the Fonts page.
- 3 Click the Output tab of the Fonts page.
- 4 Choose an option from the Font Family and Font Size drop-down menus.
- 5 Check the Use Anti-Aliasing check box to enable anti-aliasing.

Wrapping message pane text

For certain operations, the message pane contains a toolbar button that you can use for wrapping text. If you have long lines of text output in the message pane, you may want to wrap the lines for easier viewing and less scrolling. Either click the Word Wrap button on the message pane toolbar or right-click the message pane and choose Word Wrap.

Hiding and showing the message pane



To show or hide the message pane, choose View|Panels|Messages or click the Messages button on the toolbar to toggle its state. You can also hide the message pane by right-clicking the message tabs and choosing Hide Message View, or pressing *Ctrl+Alt+M* if you are using the CUA keymappings.

When you want to close tabs in the message pane, click the Close Tab button. Close individual tabs or all tabs in the message pane by right-clicking the message tabs and selecting Remove <name> Tab or Remove All Tabs.

Undocking the message pane



If you prefer, you can undock the message pane so it becomes a free-floating window that you can position on your screen as you like. To undock the message pane, click on the message pane title bar and drag it away from the browser. To re-attach the message pane to the bottom of the browser, click the icon at the upper-right corner of the message pane title bar. For more information, refer to [Chapter 19, "Configuring your workspace."](#)

Using the status bars

There are three status bars in JBuilder's work environment.

- Main status bar
- Editor status bar
- Message status bar

Main status bar



The main status bar is displayed at the bottom of workspace and keeps you updated on any operations and their results.

Build succeeded with 0 files built. Build took 5 seconds.

Editor status bar

When you open the source view by choosing the Source tab, the editor status bar is displayed at the bottom of each opened file window. It displays information specific to the current file, such as the name of the file, the cursor location (line number and column), the insertion mode, and the key binding mode in a text file or the size of an image file.

When you have a text file opened in the editor, the status bar contains a magnifying glass tool you can use to change the font size used in the editor. It also contains the ScopeInsight icon that helps you identify to the associated class, method, or expression in your code.

WelcomeFrame.java | Insert | 30:35 | CUA | Q |  

Source | Design | Bean | UML | Doc | History

Using the editor status bar to display line numbers

The editor status bar provides an easy way to hide or display line numbers.

- 1 Click the line/column drop-down menu in the editor status bar.
- 2 Click the Show Line Numbers check box to hide line number display.

The default is checked (to display line numbers).

Using the editor status bar to navigate to lines

You can easily navigate to a specific line in your code using the editor status bar.

- 1 Click the line/column drop-down menu in the editor status bar.
- 2 Choose Go To Line to open the Go To Line Number dialog box.
- 3 Enter the line number.

The current line number displays in the text field. If you enter a value larger than the number of lines in your file, a beep sounds and the largest valid line number displays instead.

Changing keymaps from the editor status bar

The editor status bar displays the default keymapping type, CUA. JBuilder provides an easy way to change to a different keymapping selection, for example, to Brief, Visual Studio, Emacs, Macintosh, or Macintosh CodeWarrior.

- 1 Right-click the drop-down menu adjacent to CUA on the editor status bar.
- 2 Choose a keymapping type from the drop-down menu to change the current keymap selection.

Changing editor font size from the editor status bar

You can change the size of the font in the editor by zooming in and out. This allows you to fit more text in the editor or otherwise adjust the viewing size for your comfort. The zoom increments are small, so you can control the apparent size of the view precisely.

- 1 Click the drop-down menu adjacent to the magnifying glass in the editor status bar.
- 2 Choose Zoom In to enlarge the font size.
- 3 Choose Zoom Out to reduce the font size.
- 4 Choose Normal to restore the editor font size to the original setting.

Message status bar

The message status bar is displayed at the bottom of the message pane during such processes as running, debugging, and version control and keeps you updated on these operations and their results. The toolbar shown below displays during debugging operations. For more information about the debugger toolbar, see “Debugger toolbar” in *Building Applications with JBuilder*.



Monitoring memory use

If you want to check on memory usage, you can enable and customize heap reporting in the Preferences dialog box (Tools|Preferences|Browser|Status Bar). The heap report displays megabytes used and allocated in the lower right corner of the status bar and sounds an audible beep when heap memory is below 10 MB.

Compiling and building Java programs

Compiling and
building features vary
by JBuilder edition

Compiling and building Java programs includes the process of converting Java from text to bytecode, establishes project dependencies, and builds Java source files and other buildable files in projects, such as archive files, web modules, and other buildable nodes.

Compiling Java projects

Compiling your Java source files translates the code written in the Java programming language into bytecode that can be interpreted by a virtual machine for the Java platform. A Java compiler reads Java source files and produces the Java program in the form of `.class` files. Compiling produces a separate `.class` file for each class in a source file. When you run the resulting Java program on a particular platform, the Java interpreter for that platform runs the bytecode contained in the `.class` files.

JBuilder lets you choose the compiler you want to use and lets you set options for the compiler, such as the type of debugging information to include in compiled classes, target VM, and whether or not to obfuscate code when it is compiled. You can choose standard and extended compiler options from the Project Properties dialog box. See “Setting compiler options” in *Building Applications with JBuilder* for more information.

You also can use the command line and convenient command line tools and macros to compile and build your programs. See “Using command-line tools” in *Building Applications with JBuilder* for more information.

JBuilder provides incremental compiling for open, compilable files with resolvable dependencies. A file updates as you work on it, automatically compiled and stored in the cache. This means that you can use many of JBuilder’s tools, such as refactoring, CodeInsight, and ErrorInsight, without having to make your project first.

Building Java projects

Building your project is as simple as choosing Project|Make Project. The JBuilder build system, based on Ant, builds all buildable nodes. You can also right-click a buildable node or nodes in the project pane and choose Make to only build selected nodes. During the make phase, various processes may occur depending upon what node is selected: preparing non-Java files for compiling, compiling of Java source files, generation of archive files, and so on. Messages from the build system are displayed in the message pane.

JBuilder gives you control over the build process, letting you set global build preferences, specify build properties for a project, and create external build tasks. You can also choose the filters to exclude or include packages from the build process and control build output behavior in the message pane. You can set the Build tab behavior on the Build page of the Project Properties dialog box (Project|Project Properties|Build).

Refer to “Setting build preferences” and “JBuilder build menus” in *Building Applications with JBuilder* for more information about building projects.

JBuilder provides additional build features which vary by JBuilder edition and are described in the tables below.

Table 9.1 Build features available in all JBuilder editions

Feature
Building projects with Ant
Adding build files to your project with the Ant wizard (Wizards Ant)
Adding additional targets and tasks to the Project menu for building projects and project groups
Customizing and extending the build system with the OpenTool Builder class
Automatic discovery of source packages.

Table 9.2 Build features available in JBuilder Developer and Enterprise

Feature
Switching Java compilers
Filtering packages and excluding them from the build process
Exporting a JBuilder project to an Ant build file with the Export To Ant wizard (Wizards Export To Ant)
Building project groups
Creating external build tasks to execute during the build process
Selectively copying resources
Generate Java from SQLJ translators

See also

- “Compiling Java programs” in *Building Applications with JBuilder*
- “Building Java programs” in *Building Applications with JBuilder*

Chapter 10

Running, testing, profiling, and debugging source code

After you have developed and built your program or application, you will want to run it. You can run, test, profile, and debug your applications all from within JBuilder and with one click of a button. JBuilder also lets you set numerous options and create unique runtime configurations, so you have complete control over the process.

Running programs in JBuilder

When you're ready to run your program, you can simply run it, you can run it and debug it at the same time, or you can profile and optimize it. When you run your program, JBuilder uses the class path to locate all classes your program uses.

You can run runnable files individually or run whole projects, including OpenTools. Runnable files are files which contain a program entry point. For instance, an application file needs a class with a `main()` method to be runnable, and an applet file needs to have an `init()` method and a corresponding HTML file with an `<applet>` tag.

Tip To run your program more quickly and efficiently, click the garbage collector icon located in the right lower corner of the status bar to perform garbage collection for your code. The tool tip, "Force Garbage Collection", identifies the icon for you. And remember, you can perform garbage collection at any time, not just during a run.

See also

- "Running Java programs" in *Building Applications with JBuilder*

Unit testing

Unit testing is a feature of all editions of JBuilder

Cactus integration and Fixture wizards are features of JBuilder Enterprise

JBuilder integrates JUnit's unit testing framework into its environment. This means that you can create and run JUnit tests within the JBuilder IDE. JBuilder also supports Cactus, which provides unit testing of server-side Java code. In addition to the powerful unit testing features of JUnit and Cactus, JBuilder adds wizards for creating test cases, test suites, and test fixtures, and a test runner called JBuilderTestRunner which combines both text and GUI elements in its output and integrates seamlessly into the JBuilder IDE.

All editions of JBuilder include the following unit testing features:

- Test Case wizard
- Test Suite wizard
- Test running
- JBuilderTestRunner
- JUnit TextUI support
- JUnit SwingUI support
- Test debugging

JBuilder Developer adds these features:

- Micro Test Case wizard
- Micro Test Builder

JBuilder Enterprise adds these features:

- EJB Test Client wizard
- Cactus Setup wizard
- JDBC Fixture
- JNDI Fixture
- Comparison Fixture wizard
- Custom Fixture wizard

See also

- "Unit testing" in *Building Applications with JBuilder*
- "Micro Unit Testing" in *Developing Mobile Applications for MIDP*
- "Micro Unit Testing" in *Developing Mobile Applications for i-mode*

Optimizing and profiling source code

Optimizeit integration is a feature of JBuilder Enterprise

The integration of Optimizeit Suite in JBuilder makes it easy to optimize and profile both local and distributed applications, including servlets, JSPs, applets, EJBs, and unit tests. To test files in your project, choose RunOptimize Project. Your program may be compiled. When using Optimizeit, you can pinpoint memory problems, thread contentions, and dead code with the following processes: profiler, thread debugger, code coverage, and request analyzer.

You can customize Optimizeit processes in the Edit Runtime Configuration dialog box which opens when you choose RunOptimize Project. You also can access this dialog box when you click the Edit button on the Run page of the Project Properties dialog box (Project!Project Properties!Run). The Optimizeit user interface, consisting of the tool selection, toolbar, view tabs, and status bar, is displayed in the message pane.

For more information, refer to "Profiling your applications" in *Building Applications with JBuilder* in JBuilder's online help.

Debugging your program

**Debugging features
vary by JBuilder
edition**

You can debug both local and distributed applications, including servlets, JSPs, applets, EJBs, and unit tests. To debug files in your project, choose Run|Debug Project. Your program may be compiled. You can set runtime configuration options for the debugger. When debugging, you can view control your program's execution, examine data values and step through code. The debugger UI, consisting of the debugger views, the toolbar, the session tab and the status bar is displayed in the message pane.

See also

- “Debugging Java programs” in *Building Applications with JBuilder*

Managing application development

JBuilder provides a variety of tools and features to help you manage your application development. Whether you are tracking the changes you make locally, sharing work as part of a development team, managing team development requirements, or restructuring your code to improve design and maintainability, JBuilder offers tools and features to make it easy to manage your source files.

Local version control

JBuilder provides many ways to compare files, view differences between different files and between different versions of the same file, and to manage, merge, and revert file differences. The Compare Files dialog box (File|Compare Files), the Manage Local Labels dialog box (Project|Manage Local Labels), and the history view provide access to these features.

JBuilder also offers file decorations to help you monitor the files that you or others modify. The Decorations page of the Project Properties dialog box provides options for icons and text to signify files modified in the VCS workspace and repository.

See also

- “Comparing files and versions” in *Building Applications with JBuilder*
- [“Using file status decorations” on page 70](#) for more information about specifying the decoration options

Version control system (VCS) integration

**Team development
features vary by
JBuilder edition**

When you are developing applications as part of a team, you need a way to manage the revisions of the files you work on. Version control systems (VCSs) store a complete record of file revisions. Version control systems make it possible for a team of developers to work on the same set of files without overwriting each other’s changes, and provide logs and version tracking information so that anyone with suitable access can find when any change was made.

JBuilder integrates with several popular version control systems, including Borland StarTeam, Concurrent Versions System (CVS), Microsoft Visual SourceSafe, Subversion, and Rational ClearCase.

See also

- *Managing Application Development in JBuilder*

Refactoring code

Refactoring features vary by JBuilder edition

Refactoring is a method of restructuring your code without changing the way it behaves. You can use refactoring to update and improve your code base. Refactoring can make the design of your application clearer, and as a result, easier to understand and maintain. JBuilder supports many types of refactorings. Refactorings are available from the editor context menu, the Edit menu, the structure pane context menu, and a UML diagram context menu. (UML is a feature of JBuilder Enterprise.)

Some refactorings display a dialog box and a Refactoring tab in the message pane; other refactorings occur automatically. The Refactoring tab displays a preview of the refactoring. For a complete list of refactoring types and implementation instructions, see “Executing refactorings” in *Building Applications with JBuilder*.

See also

- “Refactoring code” in *Building Applications with JBuilder*
- “Distributed refactorings” in *Building Applications with JBuilder*

Visualizing code with UML

This is a feature of JBuilder Enterprise

JBuilder uses UML diagrams for visualizing code and browsing classes and packages. UML diagrams can help you quickly grasp the structure of unknown code, recognize areas of complexity, and increase your productivity by resolving problems more rapidly.

Two UML diagrams are available in JBuilder:

- Limited package dependency diagrams
- Combined class diagrams

To view a UML diagram, compile the project and double-click a source file or package in the project pane. Then choose the UML tab at the bottom of the browser.

The UML browser provides access to JBuilder refactoring features. When the UML browser is used in conjunction with refactoring, you can locate and address code structure or design problems quickly and effectively. You can perform the following types of refactorings from the UML browser:

- Rename
- Move
- Change Parameters
- Extract Interface
- Introduce Superclass

See also

- “Visualizing code with UML” in *Building Applications with JBuilder*
- “Tutorial: Visualizing code with the UML browser” in *Building Applications with JBuilder*

Managing project requirements

**This is a feature of
JBuilder Enterprise**

When you are working as a team to develop applications, you require efficient and effective project management. JBuilder offers CaliberRM, a shared, Web-based management system to streamline the process. CaliberRM provides the team with an efficient feature-tracking, requirement management, communication tool.

Managing project requirements as a cohesive unit makes developing superior e-business and enterprise applications easier. CaliberRM supports uncomplicated project requirement tracking and updating. When using CaliberRM, teams can understand and recognize the scope of project requirement changes and their effect on the outcome of the project.

See also

- “Managing requirements using CaliberRM” in *Managing Application Development in JBuilder*

Internationalizing programs

Special features in JBuilder make it easy to take advantage of Java's internationalization capabilities, allowing your applications to be customized for any number of countries or languages without requiring cumbersome changes to the code.

JBuilder includes the following features to help you easily internationalize your Java applets and applications:

- A multilingual sample application
- Resource Strings wizard, which is used to eliminate hard-coded strings
- dbSwing internationalization features
- Locale-sensitive components
- Components that display Unicode characters
- Internationalization features in the UI designer
- Unicode support in the debugger
- Support for all JDK native encodings

See also

- “Internationalizing programs with JBuilder” in *Building Applications with JBuilder*

Chapter 12

Developing applications

JBuilder provides all the tools and features you need to develop a variety of types of applications. These tools and features, and how to use them, are documented extensively. The following sections describe the types of applications you can develop, and refer you to documentation that will guide you in the development process.

Developing web applications

Web Development is a feature of JBuilder Developer and Enterprise

Applet development is a feature of all editions of JBuilder

JBuilder provides many tools that make web application development simple:

- Wizards make creating web modules, applets, servlets, JavaServer Pages (JSP), JSF-enabled web modules, Struts-enabled web modules, and JNLP files and Homepages for Java Web Start applications easy.
- Framework and JSP tag library support allows you to easily add support for various frameworks and tag libraries to your web module or JSP.
- Popular frameworks and tag libraries, such as JSF (JavaServer Faces), Struts, JSTL (JavaServer Pages Standard Tag Library), and InternetBeans Express are included with JBuilder.
- Automatic WAR file generation helps you create an archive file for your web module.
- The Archive Builder helps you create JAR files for an applet, a Java Web Start applet, or a Java Web Start application.
- The Web Module DD Editor provides a graphical user interface (GUI) for editing the `web.xml` deployment descriptor file.
- The JSF Flow designer provides a visual tool for designing the flow of a JSF application.
- The Faces Config Editor provides a visual designer for Navigation Rules and Navigation Cases, as well as a form-based editor for editing the elements in the `faces-config.xml` deployment descriptor file.
- The Struts Config Editor provides visual designers for Actions, Form Beans and Tiles elements, as well as a form-based editor for editing other elements in the `struts-config.xml` deployment descriptor file.
- JSP TagInsight provides lists of available tag library tags, elements, attributes, and entities when editing a JSP.

See also

- “Introduction” in *Developing Web Applications*
- “Overview of the web application development process” in *Developing Web Applications*
- “Package com.borland.internetbeans” in *DataExpress Component Library Reference*
- [“TagInsight” on page 145](#) for information about tag completion within HTML, JSP, and XML source files
- “Using the JSF framework in JBuilder” in *Developing Web Applications*

Developing enterprise applications

Support for developing enterprise applications varies by JBuilder edition

The Java 2 Platform, Enterprise Edition (J2EE) is an architecture for a Java development platform for distributed enterprise applications. It was developed by Sun Microsystems, with input from the development community, including Borland. J2EE platform products, such as the Borland Enterprise Server, offer the developer the capability of building applications with the following benefits:

- Reliability and scalability, so business transactions are processed quickly and accurately
- Excellent security to protect users’ privacy and the integrity of the enterprise’s data
- Ready availability, to meet the increasing demands of the global business environment

Combining J2EE technologies into an enterprise application

Today’s web-based enterprise applications often combine multiple facets of the J2EE architecture, such as Servlets, JavaServer Pages (JSP), the Java Message Service (JMS), Java Database Connectivity (JDBC), and Enterprise JavaBeans (EJB). JBuilder, when coupled with a supported application server from companies such as Borland, BEA, IBM, and JBoss, greatly simplifies and speeds the development of J2EE applications. JBuilder provides features and wizards to help you develop enterprise applications for specific application servers.

Developing Enterprise JavaBeans

This is a feature of JBuilder Enterprise

The EJB Designer is a true Two-Way Tool that allows you to design your EJB 2.0 and 1.1 components visually as JBuilder generates the code from your design. You can make changes to your design either through the EJB Designer, or by editing the generated source code directly. Your source code and your design remain synchronized. As you work with the EJB Designer, your deployment descriptors are being created for you, preparing your bean for deployment to your selected application server. You can edit those deployment descriptors using the EJB Module DD Editor. When you are done developing your enterprise beans, JBuilder has wizards and tools to help you deploy them to your target server.

See also

- *Developing Applications for J2EE Servers*
- *Developing Applications with Enterprise JavaBeans*

Working with XML

JBuilder provides several features and tools to support development of applications using the Extensible Markup Language (XML). XML, a platform-independent method of structuring information, separates the content of a document from its structure. XML can be used to exchange data between databases and Java programs. XML support features vary by JBuilder edition.

Some of the XML features include:

These are features of JBuilder Developer and Enterprise

- Creating XML-related documents
- Automatically completing XML code with TagInsight
- Validating XML documents
- Remapping of DTDs and schemas to local files and adding OASIS catalog files
- Transforming XML using XSLT stylesheets

These are features of JBuilder Enterprise

- Marshalling and unmarshalling data with BorlandXML and Castor
- Presenting XML in a Cocoon framework
- Creating a SAX handler to parse XML
- Using XML database components to transfer data between an XML structure and a database

You can view an XML document in JBuilder by opening the XML document and selecting the View tab in the content pane. If the View tab is not available, you need to enable it on the XML page of the Preferences dialog box (Tools|Preferences|XML).

JBuilder provides additional features to support the use of XML in Ant build files and deployment descriptors. These features are used extensively to develop web applications, enterprise applications, and web services.

See also

- *Working with XML*
- “Package com.borland.jbuilder.xml.database.template” in the *DataExpress Component Library Reference*
- “Package com.borland.jbuilder.xml.database.xmldbms” in the *DataExpress Component Library Reference*
- “Package com.borland.jbuilder.xml.database.common” in the *DataExpress Component Library Reference*

Developing web services

This is a feature of JBuilder Enterprise

Although it’s useful to learn about the technologies behind web services, JBuilder provides designers, wizards, and tools for quickly developing and consuming web services. JBuilder works with a variety of web services toolkits. Once a toolkit is selected, JBuilder uses the toolkit to enable projects for web services, import services, and export Java classes and Enterprise JavaBeans as web services.

Web services features in JBuilder, which vary according to the toolkit selected, include:

- Configuring your project for working with web services
- Importing and exporting services in the Web Services designer
- Modifying service message flows in the Flow designer

- Monitoring SOAP messages between the client and the service while invoking the service
- Searching for and publishing web services to a UDDI registry with the Web Services Explorer

The Web Services designer provides a design surface for working with web services. The Web Services designer includes these features, which vary according to the selected toolkit:

- Creating a WSDL document that describes the web service you've developed
- Creating client stubs to invoke a web service
- Importing an EAR or a WSDL describing a web service and creating classes to invoke the service
- Creating server-side code to host the service locally
- Automatic scanning and exporting of stateless session beans and message-driven beans
- Creating asynchronous web services
- Validating services and automatic error correction with ErrorInsight

See also

- *Developing Web Services*

Developing mobile applications

This is a feature of JBuilder Developer, JBuilder Enterprise, and JBuilder Mobile Edition

Mobile development is fully integrated into JBuilder for building and deploying J2ME applications using MIDP and CLDC. JBuilder includes visual design tools for creating mobile applications, device emulation and debugging, as well as incorporating all the standard features offered with JBuilder. Support for Over the Air (OTA) provisioning is available for JDKs that implement it.

JBuilder installs and configures the Java 2 Platform, Micro Edition (J2ME) Wireless Toolkit 2.1. It also provides support for the following:

- i-mode™ development with the DoJa™ 1.5oe, 2.0, 2.5oe, and 3.0 SDKs
- Nokia Developer's Suite
- Siemens Mobility Toolkit
- Sprint PCS Wireless Toolkit
- Sony Ericsson J2ME SDK
- Zentek Developer's Network (ZDN)

See also

- *Developing Mobile Applications for MIDP*
- *Developing Mobile Applications for i-mode*

Developing database applications

Database application
development features
vary by JBuilder
edition

JBuilder provides a number of features to easily support the creation and maintenance of database applications.

DataExpress and dbSwing components are Borland value-added components. They are located on the UI designer's component palette and can be dropped onto your design surface. These components make it easy to connect to a database, retrieve data from a data source, save changes back to the data source, and manipulate the data. These components also provide international support.

JDataStore provides embedded database functionality in your applications with a single JDataStore file and the JDataStore JDBC driver (and its supporting classes). No server process is needed for local connections. In addition to industry-standard JDBC support, you can take advantage of the added convenience and flexibility of accessing JDataStore directly through the DataExpress API. You can use both types of access in the same application.

The Database Pilot is an all-Java, hierarchical database browser that also allows you to view, edit, insert, and delete data in tables. It presents JDBC-based meta-database information in a two-paned window. To display the Database Pilot, select Tools| Database Pilot.

See also

- *Developing Database Applications*
- *JDataStore Developer's Guide*
- "Package com.borland.dbswing" in the *DataExpress Component Library Reference*

Chapter 13

Archiving and documenting applications

After you have developed and tested your application you are ready to prepare it for deployment. Deploying a Java program consists of bundling together the various Java class files, image files, and other files needed by your program, and copying them to a location on a server or client computer where they can be executed. You can deliver the files separately, or you can deliver them in compressed or uncompressed archive files.

Typically, generating documentation is done at the end of the development cycle. Generating documentation for your application consists of running the Javadoc tool to create API documentation from comments you have entered in your source code.

Archiving applications

Archiving your application files simplifies deployment, and JBuilder provides a variety of wizards to help you create an archive that is appropriate for your application and deployment environment. The wizards create an archive node in your project, allowing easy access to the archive and manifest files.

There are several ways to access archive wizards.

- Click the New button on the main toolbar to open the object gallery, then click Archive to view the archive wizards page.
- Click File|New to open the object gallery.
- Click the New button's drop-down menu, then choose the Archive submenu to see a list of archive wizards.

At any time during development, you can make the archive file, rebuild it, or modify its properties. You can also directly add JAR files to your project's classpath. See "Updating paths" in *Getting Started with JBuilder* for more information about adding archive files to your project classpath.

It is easy to view the contents of the archive, as well as the contents of the manifest file in the editor. When you hold the *Ctrl* key down and move the mouse over the text in the manifest file, you can identify reference links to files on the project source path.

See also

- “Deploying Java programs” in *Building Applications with JBuilder*
- “Understanding the manifest file” in *Building Applications with JBuilder*
- “Using the Archive Builder” in *Building Applications with JBuilder*
- “Adding unrecognized file types as generic resource files” in *Building Applications with JBuilder*
- “Creating executables with the Native Executable Builder” in *Building Applications with JBuilder*
- “Customizing executable configuration files” in *Building Applications with JBuilder*

Developing runnable application archives

**This is a feature of
JBuilder Developer
and Enterprise**

Though Java archive files are mainly used for deployment, you can also use them for quickly running applications. When you develop runnable Java archive files for applications, you can run those applications with one click from the project pane.

To create runnable application archive files,

- 1 Choose File|New to open the object gallery.
- 2 Choose Archive in the object gallery tree.
- 3 Choose the Application icon and click OK or double-click the Application icon.

The Archive Builder wizard opens.

- 4 Type a name into the Name field or accept the default application name.
- 5 Browse to the location for your JAR file, or accept the default JAR file in the File field.
- 6 Click Finish.

The other steps in the wizard are for more advanced settings. You can specify the archive contents, library dependencies, manifest options, obfuscator options, and the application’s main class.

- 7 Right-click the Application node in the project pane.
- 8 Choose Make.
- 9 Expand the Application node to display the JAR file.
- 10 Right-click the JAR file and choose Run Using Defaults.

The application runs.

Documenting source files

At the end of the development cycle for a project, you may want to generate the API documentation for your source code. The Javadoc tool created by Sun Microsystems generates documentation in HTML format, using comments you have entered in your source files. The comments must be formatted according to Javadoc standards.

JBuilder includes a number of features to support Javadoc documentation generation, including a wizard that creates a documentation node which holds properties for a Javadoc run. This node is displayed in the project pane. The Javadoc wizard includes settings for project and build properties, package and visibility level, and command line options. Javadoc can be generated each time you build your project, using the current properties. (These are features of JBuilder Developer and Enterprise.)

JBuilder also includes these other Javadoc-related features:

- Automatic generation of comment and parameters based on the class, interface, method, field, or constructor signature
- JavadocInsight for entering the Javadoc tag
- Ability to automatically add and view `@todo` tags
- Ability to report and fix Javadoc comment conflicts
- Documentation archiving with the Archive Builder
- “On-the-fly” Javadoc generation
- Ability to create custom tags (JBuilder Developer and Enterprise)
- A Doc viewer to view the generated Javadoc (JBuilder Developer and Enterprise)

See also

- [“Applying Javadoc shortcuts” on page 121](#)
- “Creating Javadoc from source files” in *Building Applications with JBuilder*
- The Javadoc Tool home page on Sun’s web site at <http://www.java.sun.com/j2se/javadoc/>

Part III

Using JBuilder's editor

Chapter 14

Introduction

JBuilder provides many powerful tools for creating and managing your source code. While JBuilder provides wizards and designers to help automate code generation, the editor is integral to the process of developing and maintaining your source code. In this section of *Getting Started with JBuilder* you'll learn how to use JBuilder's editor, which includes numerous coding tools and shortcuts.

This section contains the following chapters:

- [Chapter 15, “Working in the editor”](#)
Explains how to use JBuilder's code editor features to perform basic operations, such as selecting, formatting, and finding text, finding code elements and definitions, applying Javadoc shortcuts, using bookmarks, moving, opening, and adding files, and printing your source code.
- [Chapter 16, “Using code shortcuts”](#)
Describes the powerful tools and features available in the editor. Some of these include CodeInsight, ScopeInsight, Code folding, ErrorInsight, SyncEdit, code templates, and TagInsight for code and markup languages.
- [Chapter 17, “Customizing the editor”](#)
Provides information for modifying the editing environment in JBuilder. Includes instructions for setting keymapping preferences, and describes how to change the appearance and behavior of the editor.

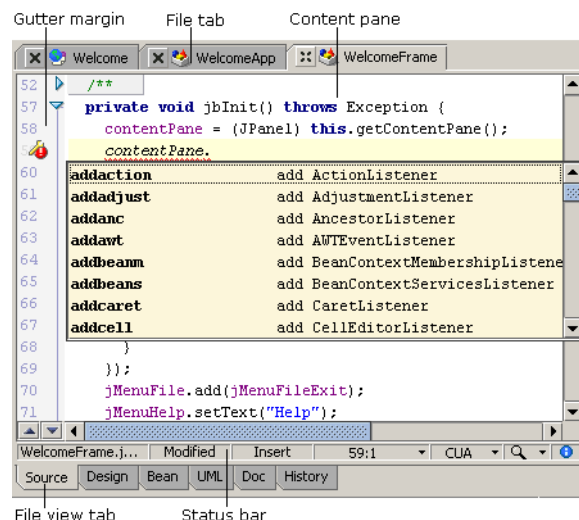
Chapter 15

Working in the editor

JBuilder has a superb code editor with many features for making the task of writing code quicker and easier.

To open a file in the editor, either double-click a text-based file in the project pane (upper left pane) or select the file and press *Enter*. To reopen a file in the editor, choose *File|Reopen|Select File*. The Select File dialog box provides a history list of files to reopen for that project. When you start editing your file, don't be concerned if it contains extremely long lines, the editor automatically adjusts its width and scrollbar to accommodate the longest line in your file.

Figure 15.1 JBuilder's editor



Note the editor status bar at the bottom of the editor, indicating the file name, the cursor location (line number and column), and the insertion mode of a text-based file.

The editor offers a variety of productivity features, such as keyboard shortcuts, syntax highlighting, customizable editor keymappings, and coding shortcuts. It also provides Javadoc coding shortcuts, searching, finding definitions for source code elements, and printing, as well as a fully customizable editor. Many of these features can be customized in the Preferences dialog box (*Tools|Preferences|Editor*).

Choose one of the following topics for detailed information about JBuilder's source code editor.

- ["Using the editor's context menus, file tabs, gutters, and actions" on page 108](#)
- ["Navigating in the editor" on page 109](#)
- ["Selecting, formatting, and finding text" on page 109](#)
- ["Finding code elements and definitions" on page 119](#)
- ["Applying Javadoc shortcuts" on page 121](#)
- ["Using bookmarks in the editor" on page 125](#)
- ["Moving, opening, and adding files" on page 127](#)
- ["Coding shortcuts" on page 127](#)
- ["Printing support in the editor" on page 128](#)

Using the editor's context menus, file tabs, gutters, and actions

The editor includes numerous handy tools and capabilities within its context menus, file tabs, and gutter. It also includes settings for various editor actions, including smart key, saving, and searching actions, among many others. Hover the cursor and right and left-click the mouse on the editor's pane, tabs, status bar, and gutter to display tool tips and menus to help you create your files.

Using editor context menus

Many features are available from the editor's context menu. These features vary based on which JBuilder edition you use, what your project settings are, and where the cursor is in the editor when you invoke the menu. Right-click with your cursor in the editor or click the editor and press *Shift+F10* to display the context menu. Some of the menu options include: searching, refactoring, customizing, optimizing imports, using wizards, adding bookmarks, setting breakpoints, and setting preferences.

Using file tabs

The file tabs offer context menus which vary depending on the JBuilder edition, the file type, and the project settings. The file tabs display the file name. The title bar displays the open file's full path and extension. When you hover the mouse over a file tab, a tool tip appears displaying the full file path, extension, and file modification information.

Right-click on the file tab to view the numerous context menu choices that range from closing, saving, and reverting files, to undocking, running, and debugging files.

Using the editor's gutter

The editor's gutter margin provides a context menu to help you navigate, customize, and debug. The gutter context menu options include displaying line numbers, showing the current scope, using code folding and bookmarks, enabling and toggling breakpoints, and viewing breakpoint properties.

The editor's gutter icons identify various editor tools and actions (such as breakpoints, SyncEdit, and code-folding), code problems, and bookmark locations. When there is more than one icon located on the same line, the icons are stacked with only one icon showing on top. A small plus sign (+) on the top icon signifies that there is more than one icon present.

When you left-click the top icon, a menu displays commands to remove the icons located in that position. When you right-click the top icon on the stack, the regular gutter context menu appears. An exception to these actions is the SyncEdit icon. When left-clicked, it automatically moves to a different gutter location adjacent to the selected text. Also, the SyncEdit and ErrorInsight icons only provide entry to their specific actions when left-clicked.

Using editor actions

Included among the editor's plentiful resources are settings to customize the editor's actions. The settings can make it easier for you to use the editor. Access the settings from the Preferences dialog box (Tools|Preferences|Editor|Actions). You can choose to activate text drag and drop, format code when pasting, generate closing curly braces, as well as specify the actions for saving, searching, and "Smart" key behavior (*Tab*, *Enter*, *Home*, *End*).

See ["Customizing editor actions" on page 156](#) for more information or click the Help button on the Actions page of the Preferences dialog box.

Navigating in the editor

You can navigate easily through the editor, finding specific code members by clicking on those members in the structure pane (the pane located at the left lower corner of the JBuilder workspace). The editor automatically positions the cursor on the code member in the source code of the editor. You can also use bookmarking, scoping, and code folding to make editing and navigating in the editor faster.

Selecting, formatting, and finding text

Use the editor's broad functionality to select, copy, resize, find text, navigate to lines, and format code and text elements while working in the editor. When you use any of these features, you will find several options for completing your task quickly and efficiently. The following topics describe these editor features and provide instructions for use.

- ["Selecting text" on page 110](#)
- ["Selecting text by line" on page 110](#)
- ["Selecting text blocks" on page 111](#)
- ["Adding characters to text blocks" on page 111](#)
- ["Deleting characters from text blocks" on page 111](#)
- ["Dragging and dropping text" on page 112](#)
- ["Copying text and adding imports" on page 112](#)
- ["Resizing the font" on page 112](#)
- ["Formatting code" on page 113](#)
- ["Finding text in the editor" on page 117](#)
- ["Navigating to line numbers" on page 119](#)

Selecting text



JBuilder's editor provides a variety of ways to select text by character, word, or line to delete, copy, or cut and paste. If the text you select includes duplicate code elements, the SyncEdit button appears in the gutter. SyncEdit is a tool that helps you quickly edit duplicate code identifiers. See [“SyncEdit” on page 143](#) for more information. The following steps explain one way to select and copy text.

To select text,

- 1 Choose File|Open File to select a file to open in the editor.
- 2 Hold the left mouse button down and drag the mouse over a section of text.
- 3 Hold down the left mouse button and drag and drop the text to cut and paste it.
- 4 Press the *Delete* key to delete the text.

Note Text drag and drop must first be enabled in the Preferences dialog box (Tools|Preferences|Editor|Actions).

There are many other ways to select text in the editor.

- Place the cursor at the starting location, hold down the *Shift* key and click the left mouse button at the end of the selection area.
- Hold down the *Shift* key and press the left or right arrow keys to select text by letter, word, or line.
- Hold down the *Shift* key and press the up or down arrow keys to select text by line.
- Press *Ctrl+W* to select a word. Press *Ctrl+W* again (only in Java files), to widen the selection to include the next larger context of code. Continue to press *Ctrl+W* to expand the selection to include multiple lines and finally the whole file.
- Double-click the left mouse button to select a single word. Then drag the mouse while holding the left button down to select additional words, one word at a time.
- Choose Edit|Select All from the main menu.

Selecting text by line

In addition to character and word selection shortcuts, the editor provides shortcuts for quickly selecting entire lines of text using keyboard shortcuts, the mouse click, or the line number display in the gutter margin. To use the line number shortcuts, make sure line numbers are displayed in the gutter margin (see [“Displaying line numbers” on page 158](#)). The following steps describe a quick way to select one line of text.

- 1 Choose File|Open File to select a file to open in the editor.
- 2 Triple-click a line of text with the left mouse button.
- 3 Press the *Delete* key to delete the selected text or drag and drop to cut and paste the text.

The editor provides many other options for selecting text by line in the editor.

- Hold down the left mouse button and drag the mouse over the line numbers in the gutter.
- Place the cursor in the gutter over any line number in the gutter margin, press *Ctrl*, and click on any line number with the mouse to select all the lines in the entire file.
- Place the cursor in the line and press *Ctrl+L* (CUA keymap) or *Alt+L* (Brief keymap).
- Click the left margin, hold the *Shift* key down, navigate to the end of the selection, and click the right margin to select a block of lines.

Note The editor always places an inserted line selection above the line of the caret to keep the insertion from splitting the current line.

Selecting text blocks

Selecting text blocks in the editor means selecting a rectangular or columnar area of text in any area of text, rather than selecting multiple, whole lines at a time to delete, copy, cut, or paste. You can use regular keyboard shortcut keys in the CUA keymap for copying (*Ctrl+C*), cutting (*Ctrl+X*), pasting (*Ctrl+V*), and deleting (*Delete* key), a block selection. You can also extend a block selection like other selection types, using *Shift* with the arrow keys.

The following steps describe one easy way to select a block of text.

- 1 Choose File|Open File to select a file to open in the editor.
- 2 Hold down the *Ctrl* key.
- 3 Drag the cursor over the rectangular text block.
- 4 Hold down the left mouse button and drag and drop to cut and paste or press the *Delete* key to delete the text block.

There are several other ways you can select text blocks.

- Place the cursor at one end of the text selection (beginning or end), navigate to the other end, and press *Shift+Ctrl+* left mouse-click.
- Hold down the *Ctrl* key after selecting text regularly to toggle to block selection mode.
- Use *Alt+Shift* with the arrow keys.

Adding characters to text blocks

After you have selected a block of text, you can add a character or a string of characters, including tabs or spaces, to the left margin of the selected text block with some simple key commands and a pop-up window.

- 1 Choose File|Open File to select a file to open in the editor.
- 2 Press the *Ctrl* key and drag the mouse to select a text block.
- 3 Press *Ctrl+Shift+'* (single quote), to open the Slide-In Text pop-up window.
- 4 Type the character or characters into the text field to be added to the block selection.
- 5 Click OK.

The editor adds the characters to the left side of each line of the block selection.

Deleting characters from text blocks

When you select a block of text, you can delete characters from the left margin of the block selection. The characters that can be deleted must be located on the left margin of the block selection and identically match the characters there.

- 1 Choose File|Open File to select a file to open in the editor.
- 2 Press the *Ctrl* key and drag the mouse to select a text block.
- 3 Press *Ctrl+Shift+;* to open the Slide-Out Text pop-up window.
- 4 Type the character or characters into the text field to be deleted from the left margin of the block selection.
- 5 Click OK.

The editor deletes only the identical characters present on the left margin of the text block from the left of each line of the block selection.

Dragging and dropping text

You can drag and drop selected text in the editor. Select the desired text or text block, then drag it with the mouse to the new location. Text drag and drop is turned on by default, however, you can turn it off by unchecking the Enable Text Drag And Drop option in the Preferences dialog box (Tools|Preferences|Editor|Actions).

When you move an entire line of text by dragging and dropping or copying and pasting, the line is automatically indented for you when you drop it on the new line. This automatic code formatting is turned on by default, however, you can turn it off by unchecking the Format Code When Pasting option in the Preferences dialog box (Tools|Preferences|Editor|Actions).

Copying text and adding imports

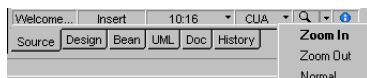
The editor automatically indents copied text in your source code and also allows you to automatically add the necessary import statements for the copied text. When you copy and paste a text selection, the editor analyzes the file that the text is pasted into and determines whether additional import statements are required. If at least one class needs to be imported, the Paste dialog box opens.

The Paste dialog box contains a list of all the required imports and asks if you want to include the import statements. When you click the Add Imports button, the editor adds the required import statements to the proper area in your source code.

Resizing the font

The editor provides a way to quickly resize the font to suit your needs. Use the editor status bar in the editor to increase or decrease font size.

- To increase the font size, click the magnifying glass icon at the right side of the status bar of the editor. Each time you click the icon, your text grows larger.
- To increase the font size, click the arrow next to the magnifying glass icon and choose Zoom In.
- To decrease the font size, click the arrow next to the magnifying glass icon and choose Zoom Out.
- To return the size of the text in the editor to its default size, click the arrow next to the magnifying glass icon and choose Normal.



Resize your code font, choose a different font family, or use look and feel settings (only for the browser), to make your text easier to see and work with in the editor, browser, and message pane. The Fonts page of the Preferences dialog box is the place to customize all of these settings.

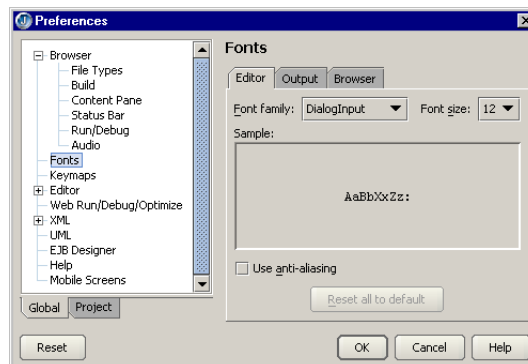
To change the font size and family used by the editor, message pane, and browser,

- 1 Open the Fonts page of the Preferences dialog box (Tools|Preferences|Fonts).
- 2 Click the Editor, Output, or Browser tab on the Fonts page.

3 Choose your changes for the font family and size.

A text sample of the font size appears in the Sample box.

Note You can choose to use default look & feel settings only on the Browser page.



Click the Help button on the Fonts page in the Preferences dialog box for more information.

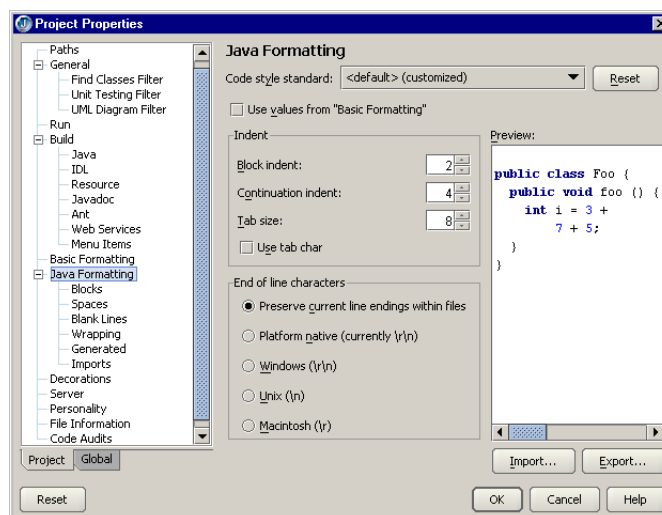
Formatting code

You can specify your formatting preferences and automatically format your source code and text in the editor. Choose one of the following topics for detailed information about code and text formatting:

- [“Accessing format settings” on page 113](#)
- [“Applying formatting to your code” on page 115](#)
- [“Customizing screen elements” on page 115](#)
- [“Wrapping curly braces around code blocks” on page 116](#)

Accessing format settings

The Project Properties (Project|Project Properties) and Default Project Properties (Project|Default Project Properties) dialog boxes offer a variety of formatting choices. These formatting options include basic formatting (indentation and end of line characters), and specific Java file formatting.



To access settings for formatting your code, choose the Basic Formatting page or the Java Formatting page of the Project Properties dialog box (Project|Project Properties). You also can access these pages in the Default Project Properties dialog box. The Basic and Java pages provide indentation and end of line customizations. The Java formatting page also contains many other Java formatting elements.

- Blocks
- Spaces
- Blank lines
- Wrapping
- Generated
- Imports

The Java and Basic formatting pages contain a Preview area that displays a sample of the formatting setting.

Note The *Tab* key on the keyboard is set to auto-format (to insert a tab character). If you want to change the *Tab* key's auto-format setting, go to the Keymaps page of the Preferences dialog box (Tools|Preferences|Keymaps) to change the key bindings for the *Tab* key. Alternatively, you can use the Actions page of the Preferences dialog box to customize the *Tab* key actions.

See also

- [“Editing keymaps” on page 152](#)
- [“Applying formatting to your code” on page 115](#)

For other formatting options for your source code in the editor and for content and message pane settings, access the Preferences dialog box (Tools|Preferences). You can choose to modify the following elements from the Font page of the Preferences dialog box (Tools|Preferences|Font).

- Editor font family
- Editor font size

You can modify the following elements from the Display area on the Editor page of the Preferences dialog box (Tools|Preferences|Editor).

- Blink caret
- Current line highlighting
- Visible white space characters
- Dimming white space colors
- Double buffering
- Visible editor margins for text

Note If you want to change the maximum width of the content or message pane tabs or customize other browser or content pane formatting elements, see the Content pane page or the Browser page of the Preferences dialog box.

The Preferences dialog box also provides formatting and editing options for markup languages. To access the settings, choose the Markup Formatting or Markup Editing page of the Preferences dialog box (Tools|Preferences|Editor).

For more information about the Editor, Browser, Fonts, Content pane, Markup Formatting, or Markup Editing pages of the Preferences dialog box, click the Help button on each page.

Applying formatting to your code

The editor provides several ways to apply formatting to your code. To apply formatting to all the code in a file, open a file in the editor and choose **Edit|Format All** or right-click in the editor and choose **Format All** from the context menu. To apply formatting to a single line or to a whole selected text block, press the **Tab** key. You can customize the **Tab** key behavior to match your formatting needs.

To customize the **Tab** key behavior,

- 1 Choose **Tools|Preferences** to open the Preferences dialog box.
- 2 Expand the **Editor** node and choose the **Actions** page.
- 3 In the **Smart Keys** section, click the **Tab** combo box arrow.
- 4 Choose one of the **Tab** actions from the drop-down menu.

Click the **Help** button on the **Actions** page of the Preferences dialog box for more information about the **Tab** key settings.

Important

Setting the format properties in the **Project Properties** dialog box (**Project|Project Properties**), applies to all files in a project. Setting the format properties in the **Default Project Properties** dialog box (**Project|Default Project Properties**), applies to all projects subsequently created. However, changing formatting properties does not auto-format the entire project.

You can export your code formatting preferences, or import previously saved preferences using the **Import** and **Export** buttons of the **Java Formatting** page in the **Project Properties** dialog box.

You can also easily change the default Java standard settings to other formatting options. New projects inherit the settings from the project template. There are several other code styles you can choose from on the **Java Formatting** page of the **Project Properties** and **Default Project Properties** dialog boxes.

- **Default (Customized)**
- **Borland**
- **Condensed**
- **Expanded**
- **Java Standard**

To change formatting from the Java standard settings,

- 1 Choose **Tools|Preferences** to open the Preferences dialog box.
- 2 Click the **Project** tab at the bottom of the left panel.
- 3 Choose the **Java Formatting** page.
- 4 Choose a code style standard from the drop-down menu on the **Java Formatting** page.

For details on setting code formatting options, press the **Help** button at the bottom of the **Formatting** page of the **Project Properties** dialog box.

Customizing screen elements

Use the **Color** page of the Preferences dialog box (**Tools|Preferences|Editor|Color**) to customize the screen elements of your code in the editor. You can specify syntax highlighting for any particular element in the **Screen Elements** sample box of the **Java**, **HTML/XML**, **JSP**, or **Other** pages.

The editor includes many screen elements to customize. For example, you can set the font or color for unused variables (including methods and imports), in the **Java** page. The default color is gray. Some of the many code elements you can customize include: unused overriding method, extra keyword, input method, field, superclass, and outer

class variable, and static and deprecated member. Some of the markup language elements you can customize include unknown attribute, unknown tag name, namespace declaration, attribute, and entity.

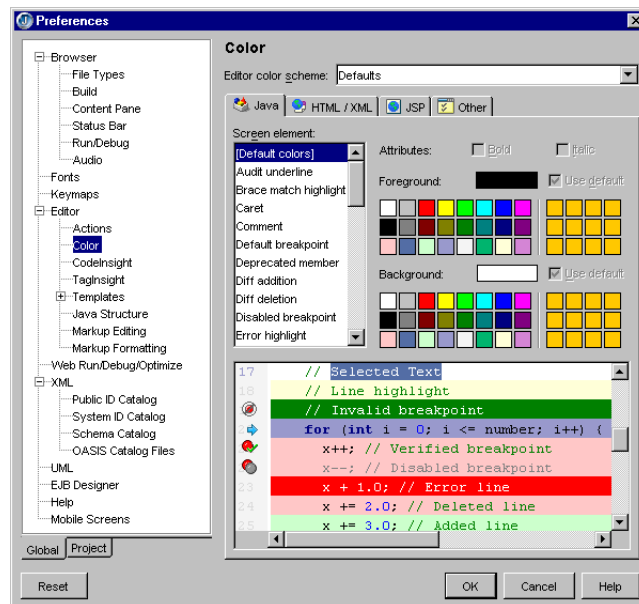
The screen elements that are common to all of the code and markup language types on the Color page of the Preferences dialog box retain the same color and font setting (for Java, HTML, XML, JSP, and Other) and are not individualized for each different screen element type.

The screen elements are grouped by file type: Java, HTML, JSP, and Other. Some screen elements are common to all file types. Changing the setting for a screen element changes it for all file types.

To customize screen elements,

- 1 Choose Tools|Preferences|Editor|Color.
- 2 Choose from one of the pages: Java, HTML/XML, JSP, or Other.
- 3 Choose the element from the Screen Element list or click the element in the sample box.
- 4 Choose from the Attributes, Foreground, and Background options.

For more information, click F1 or the Help button of the Color page.



Wrapping curly braces around code blocks

To quickly wrap a block of code in curly braces, place an open curly brace at the beginning of the block, move to the end of the block and press *Enter*.

Finding text in the editor

The editor provides numerous ways to find and replace specific text in a file. Search commands are located on the Search menu as well as from icons in the main toolbar. Modify search options on the Editor page of the Preferences dialog box (Tools|Preferences|Editor).

Table 15.1 Commands for finding text

Task	Command
Find text	Search Find
Search for text across all files in the selected path(s)	Search Find In Path
Find text and replace it with new string	Search Replace
Find text and replace it with a new string across all files in the selected path(s)	Search Replace In Path
Search for the same text again	Search Search Again
Search for text incrementally, as you type in the search string	Search Search Incremental
Go to a specific line number	Search Go To Line
Search for class member of open file.	Search Go To Class Member
Move to previous method of open file	Search Go To Previous Method
Move to next method of open file	Search Go To Next Method
Browse through a class, interface, or package	Search Find Classes
Find the declaration of a variable, method, class, or interface	Search Find Definition
Find uses or instances of a variable, method, class, or interface	Search Find References
Set bookmarks	Search Add Bookmark
Return to, edit, or remove bookmarks	Search Bookmarks
View list of Javadoc @todo comments, including file and directory location	Search View Todos
Switch to other open files from the file list	Window Switch
Switch to other open projects from the project list	Window Projects
Switch to other open projects from the Switch Project dialog box	Window Switch Project
Navigate to the previous item in the history list	Window Back
Navigate to the next item in the history list	Window Forward

Finding and replacing text

The Find dialog box and the Replace dialog box both provide the option to replace text. These dialog boxes include numerous search delimiters: case-sensitive, selected text, search direction, wildcards or regular expressions, and definition of the extent of the search/replace task. Click Help in the dialog box to learn about using these options.

Finding and replacing in the path

The Find In Path and Replace In Path dialog boxes provide searching and replacing in the directories specified in the search path using a search string. You can choose to use wildcards or regular expressions if you want to broaden your search beyond the exact words that you type.

You also can choose to include subdirectories, restrict searching to a specific location, and choose from the following search paths:

- Specified Directory
- Directory of Current File
- Directories for Open Files
- Project Source Path
- Full Source Path (Projects and Libraries)

The message pane displays the search results in table form. It includes the file name, number of matches, and directory path. When you click on the file name on the left side of the table, the search matches appear in the file text on the right. When you click on a line with a search match on the right of the table, the editor opens that file with the line highlighted. You also can double-click on an entry in the left side of the table to open the file.

You can use the message pane context menu or toolbar to cancel, suspend, resume, save, navigate to previous or next file or line, or to delete an entry. You also can perform a parallel search with each search controlled individually and begin another search before the prior search is complete.

Handy keyboard shortcuts make it easy to navigate in the table of message pane search results and to the other panes.

- Press *Ctrl+Shift+N* to move down the list of entries in the table.
- Press *Ctrl+Shift+P* to move up the list of entries in the table.
- Press *Ctrl+Tab* to move between the two panels in the table. If you continue to press *Ctrl+Tab* you will move through the other panes.
- Press *Ctrl+Shift+Tab* to move between the two panels in the table (in the other direction). If you continue to press *Ctrl+Shift+Tab* you will move through the other panes.

Finding text with the Find In Path command

The Find In Path and Replace In Path commands provide easy ways to find, navigate to, and replace strings in multiple files and directories in a defined path.

To find strings with the Find In Path command,

- 1 Choose Search|Find In Path.

The Find In Path dialog box opens. You can also choose Replace In Path to access the same dialog box options.

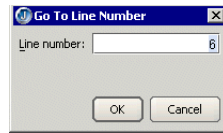
- 2 Enter the string you want to find or replace in the Text To Find.
- 3 Choose to match the case, whole word, or to use wildcards or regular expressions in your search.
- 4 Choose from the search path and subdirectory options and click OK.

The search results are displayed in table format in the message pane.

- 5 Click a file listed in the left of the table in the message pane to open up its details in the right.
- 6 Click an entry in the right side of the table to go to that line of the source file in the editor.

Navigating to line numbers

You can easily navigate to lines in your file by using the Go To Line Number dialog box.



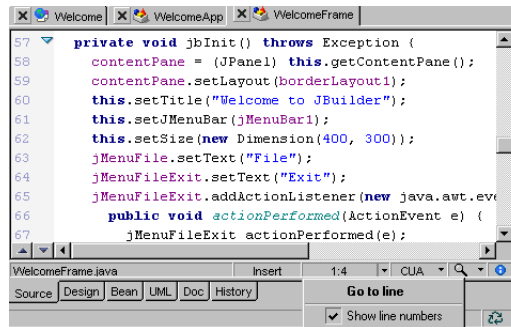
To navigate to lines in your file,

- 1 Choose Search|Go To Line or click the line and column number or its drop-down menu displayed at the bottom of the editor.

The Go To Line dialog box opens, displaying the value of your current line number location.

- 2 Enter the number of the line you want to navigate to in the Line Number field and click OK.

If you enter a value greater than the number of lines in your file, the maximum line number displays in the text field and a beep sounds.



Finding code elements and definitions

When working in the editor, you can quickly find code elements, such as a specific variable, method, class, overriding method, and you can also find references (local to JBuilder or outside of JBuilder's documentation) for specific code elements. The following subjects describe how to find code references and elements and how to easily identify unused code elements and overriding methods in the editor.

These are features of
JBuilder Developer
and Enterprise

- [“Finding a symbol's definition” on page 119](#)
- [“Finding references to a symbol” on page 120](#)
- [“Finding the overridden method” on page 120](#)
- [“Finding unused code elements and overriding methods” on page 121](#)

Finding a symbol's definition

Find Definition quickly takes you to the source code for the current variable, method, or class name in the editor if it exists. With this command, you can navigate from the usage of a symbol to its definition. If it's not available, it takes you to the stub source. Before you can find a symbol's definition, you must have compiled your project and the class that contains the definition must be on the `import` path. See [Help|Reference Documentation|Keymaps](#) for a complete list of keystrokes.

Note A code symbol is a fragment of code that represents something, such as a package, class, or method name.

To access the Find definition command position the cursor in the editor and choose one of the following actions:

- Right-click the symbol you want to see defined and choose Find Definition.
- Place your cursor on the symbol in your source code press *Ctrl+Enter*.
- Place your cursor on the symbol in your source code and press *Ctrl* and left-click on the symbol.

Note When you press *Ctrl* and left-click on an overriding method, the editor invokes the Find Super Definition action. Find Super definition will take you to the source code for the current overriding method.

The source file where the symbol is defined displays in the editor, with the cursor positioned on the symbol's definition.

Finding references to a symbol

The Find References command, available on the Search menu and on the editor and structure pane context menus, allows you to discover all source files that use a given symbol. The Find Local references command, available from the editor's context menu, displays only the local source files that use a given symbol (instances of a variable, method, class, or interface). Both commands display the reference information in the message pane located at the bottom of JBuilder's IDE.

To find all references to a symbol, right-click the symbol and choose Find References.

To discover the source file(s) where the reference exists, expand a category node and traverse the hierarchy tree in the message pane. Double-click a reference to open the source file and to position the cursor directly on the reference in the file.

Important To find references to your selected symbol, you must have already compiled your project. For more information, see "Discovering references before refactoring" in *Building Applications with JBuilder*.

Finding the overridden method

You can find the superclass of your method with the Find Overridden Method command available from the editor or structure pane context menu. The text of overridden methods displays with italic font in the editor and in the structure pane.

- 1 Compile your project.
- 2 Right click your method from the editor or structure pane.
- 3 Choose Find Overridden Method.

The Find Overridden Method command takes you to the correct superclass and highlights the method that you have overridden. You can navigate up the chain of superclasses to find all of the overridden superclass methods. The Find Overridden Method menu option is grayed out if the method does not override a super class method.

Finding unused code elements and overriding methods

**This is a feature of
JBuilder Developer
and Enterprise**

JBuilder provides an simple way to identify unused code elements and overriding methods in your code with a visual aid tool. You can easily locate unused variables, methods, imports, and discover overriding methods in your source code by the elements' customized text color or font attribute. The default text color is gray for unused variables, methods, and imports in the code editor. Overriding methods are italicized by default. You can customize these settings in the Preferences dialog box.

- 1 Choose the Color page of the Preferences dialog box (Tools|Editor|Color).
- 2 Click the Java tab.
- 3 Select Unused Variable or Overriding Method in the Screen Element field or click the sample element text in the sample box.
- 4 Choose bold or italic attributes or new foreground colors for the screen element.

Applying Javadoc shortcuts

Javadoc is a Sun Microsystems utility that generates HTML documentation files from comments you enter in API source files. The comments must be formatted according to Javadoc standards. Use JBuilder's Javadoc wizard (Edit|Wizards|Add|Javadoc) to set up properties for generating Javadoc (The Javadoc wizard is a feature of JBuilder Developer and Enterprise.). Then, when you build your project, Javadoc is automatically generated.

The following topics describe JBuilder's Javadoc tools and shortcuts that you can customize and use in the editor.

- ["Using Javadoc code templates" on page 121](#)
- ["Adding and editing Javadoc tags" on page 122](#)
- ["Using JavadocInsight" on page 122](#)
- ["Resolving Javadoc conflicts" on page 123](#) (This is a feature of JBuilder Developer and Enterprise)
- ["Using Javadoc QuickHelp" on page 124](#)
- ["Folding Javadoc comments" on page 124](#)
- ["Adding todo tags in the editor" on page 124](#)
- ["Viewing todo comments" on page 125](#)

Using Javadoc code templates

To make coding Javadoc comments easy, the JBuilder editor contains a Javadoc comment template that is activated when you type `/**` and press *Enter*. The template automatically adds the Javadoc end comment symbol, `*/`. If the cursor is positioned immediately before a class, interface, or method signature the template expands to include appropriate Javadoc tags.

To quickly add a Javadoc comment block,

- 1 Position the cursor at the desired indentation level before a class, interface, or method signature.
- 2 Type `/**`.
- 3 Press *Enter*.

The editor automatically adds the Javadoc end comment symbol and positions the cursor in the second line of the comment. If necessary, it adds appropriate tags.

For example, entering `/**` after the import statements in a class source file generates the following comment block:

```
/**
 *
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */
```

When you enter `/**` for the following method signature:

```
public void addValues(Double valueOneDouble, Double valueTwoDouble)
```

Javadoc creates the following Javadoc comment:

```
/**
 *
 * @param valueOneDouble
 * @param valueTwoDouble
 */
```

Adding and editing Javadoc tags

You can add or edit Javadoc tags and comments in your Java source code with the Javadoc dialog box. The dialog box contains Description and Tags fields where you can add or edit Javadoc text. You can also remove a tag or move the tags up or down in the tag list. Access the Javadoc dialog box from the editor or from JBuilder's main menu.

To add or edit Javadoc tags from the editor,

- 1 Right-click the class, field, method name, or existing Javadoc comment block in the editor.
- 2 Choose **AddJavadoc**.
- 3 Click the **Add** or **Edit** button to open up the corresponding dialog box and add or edit Javadoc tags to your specifications.

To add or edit Javadoc tags from the main menu,

- 1 Place the cursor on a class, field, or method name, or existing Javadoc comment block in the editor.
- 2 Choose **Edit|Wizards|AddJavadoc**.
- 3 Click the **Add** or **Edit** button to open up the corresponding dialog box and add or edit Javadoc tags.

For more information on JBuilder and Javadoc, see "Creating Javadoc from source files" in *Building Applications with JBuilder*.

Using JavadocInsight

JavadocInsight is a useful pop-up window containing a list of Javadoc tags. You can quickly enter Javadoc tags into a Javadoc comment block using JavadocInsight. You can also color-code Javadoc differently from normal line and block comments to make viewing Javadoc information easier.

Accessing JavadocInsight

To access the JavadocInsight window and see the list of Javadoc tag options,

- 1 Place the cursor in a Javadoc comment.
- 2 Enter @ in a Javadoc comment block or assign a specific keystroke in the current keymap.

You also can invoke MemberInsight in a Javadoc comment block by pressing *Ctrl+Space* or invoke ClassInsight by pressing *Ctrl+Alt+Space* and choosing the Insert Fully Qualified Class Name option.

Setting JavadocInsight pop-up timing

To set the JavadocInsight window pop-up timing,

- 1 Choose Tools|Preferences|Editor|CodeInsight.

The default Autopopup JavadocInsight option is on, and the delay is set for 250 ms.

- 2 Place your cursor on the Autopopup JavadocInsight slider and hold down the left mouse key while moving the slider. You can also use your right and left arrow keys to move the slider. The numbered display to the right of the slider reflects the changed time.

Setting JavadocInsight color options

To access the JavadocInsight page to set font and color options,

- 1 Choose Tools|Preferences|Editor|CodeInsight.
- 2 Click the Display Options button.

The CodeInsight Display Options dialog box opens.

- 3 Click the JavadocInsight tab.
- 4 Choose colors for the JavadocInsight foreground and background colors.

This is a feature of
JBuilder Developer
and Enterprise

Creating custom Javadoc tags

You can add custom Javadoc tags to your Javadoc. To display the Javadoc page of the Project Properties dialog box so you can create, remove, or edit custom Javadoc tags, choose Project|Project Properties|Build|Javadoc. Once the Javadoc page appears, use it to accomplish one of these tasks:

- To create a new Javadoc tag, click the New button to open the Create Custom Tag dialog box with tag name, heading text, and placement options. Fill in the appropriate fields.
- To edit a custom Javadoc tag, select the tag you want to modify in the list of custom tags and click Edit. Modify the fields you need to change.
- To remove a custom Javadoc tag, select the tag in the list of custom tags and click Remove.

You can also use options on the Javadoc page to add text for the custom tag that displays in Javadoc output as a header.

Resolving Javadoc conflicts

This is a feature of
JBuilder Developer
and Enterprise

You can view and resolve Javadoc conflicts from the structure pane. Javadoc conflicts occur when Javadoc tags do not match the method signatures or if no arguments are provided in the tags. When you build your project, JBuilder creates a Javadoc Conflicts folder in the structure pane. This folder contains a list of individual Javadoc conflicts including information about the line number and type of Javadoc error. Clicking on the individual item in the list highlights the Javadoc comment in the editor.

To resolve Javadoc conflicts,

- 1 Right-click the Javadoc Conflicts folder or the individual Javadoc error listed beneath the expanded folder in the structure pane.
- 2 Choose Fix Javadoc Conflicts.

JBuilder repairs the Javadoc error in the editor.

For more information about JavadocInsight, see “JavadocInsight” in *Building Applications with JBuilder*.

Using Javadoc QuickHelp

Javadoc QuickHelp is a context-sensitive pop-up window that displays Javadoc documentation without having to open and search a Javadoc file. Use Javadoc QuickHelp to quickly access Javadoc documentation from your Java source code in the editor.

To open Javadoc QuickHelp,

- 1 Place the cursor in your Java source code file in the editor.
- 2 Choose Edit|CodeInsight|Javadoc QuickHelp to open the Javadoc documentation window.
- 3 Or use the keyboard shortcut, *Ctrl+Q* (CUA) to open the lightweight Javadoc window.

Folding Javadoc comments

When you work on large Java source files with numerous Javadoc comment entries, not only can you use code folding to condense lengthy areas of source code, you can also fold Javadoc comment block areas.

- 1 Open your Java source file in the editor.
- 2 Click the down arrow in the left gutter of the editor adjacent to the Javadoc comment.

The Javadoc comment area folds to a single line.

Adding todo tags in the editor

Javadoc `@todo` tags are useful for adding reminders about what needs to be done to an area of code. Once you add Javadoc `@todo` tags to your code, you can find them easily by opening the `To Do` folder in the structure pane. To add a `@todo` tag, you can either type the opening Javadoc comment tag (`/**`) followed `@todo`, or use a quick template shortcut.

To add a Javadoc `@todo` tag using the code template,

- 1 Type `todo` at the appropriate indentation level in the editor.
- 2 Press *Ctrl+J* to expand the `@todo` tag template in your code:

```
/** @todo <cursor placed here> */
```

Customize the Javadoc `@todo` tags displayed in generated Javadoc on the Java page of the Preferences dialog box (Tools|Preferences|Editor|Templates|Java).

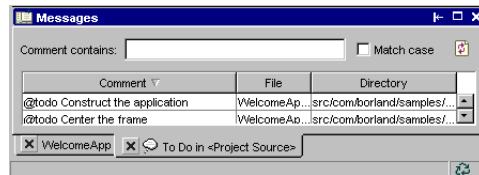
Viewing todo comments

View all of the `@todo` comments in the message pane for a project, project group, or package by accessing a command from the project pane's context menu. You also can access `todo` comments from the Search menu located on JBuilder's main menu. You can quickly open the message pane with the all of the `@todo` information displayed.

To view Javadoc `@todo` comments,

- 1 Right-click on the project or package node in the project pane and choose View Todos.
- 2 Or choose Search|View Todos from the main menu.

The message pane displays a list of todos at the base of JBuilder's IDE.



The message pane displays the `@todo` comment, file, directory, and a tab displaying the context of the `@todo` message. You can filter the message pane by `todo` comment by typing in the Comment Contains text field. Click the Match Case check box if the comment is case-sensitive. The Refresh button allows you to quickly update the todo list in the message pane when you add new todos. The structure pane auto-updates its todo list. You can also sort the pane by file name and directory.

Some of JBuilder's wizards optionally generate `@todo` tags as reminders to add code to the stub code that the wizard generates.

See "Viewing `@todo` tags" in *Building Applications with JBuilder* for more information.

Using bookmarks in the editor

Use bookmarks to help you mark elements in the editor that you want to view later. The bookmarks are persistent, project-wide, and for all file types. You can quickly move to your bookmarked code in the editor by choosing the Bookmarks command (Search|Bookmarks), or use the bookmark shortcuts you can set in the Add Bookmark dialog box (Search|Add Bookmark). The Bookmarks dialog box lists all the bookmarks you have set in your code. See the following topics for details about how to use bookmarks in the editor.

- ["Adding bookmarks" on page 125](#)
- ["Editing bookmarks" on page 126](#)
- ["Viewing bookmarks" on page 126](#)
- ["Navigating to bookmarks" on page 126](#)

Adding bookmarks

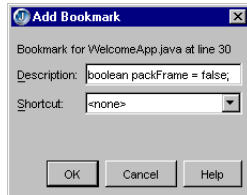
Add bookmarks to your file to make it easier to come back to a specific location in your code, text, or markup language file later.

To add bookmarks to files,

- 1 Choose Search|Add Bookmark or press *Ctrl+Shift+M*.
The Add Bookmark dialog box opens.
- 2 Enter a description for your bookmark.

- 3 Choose a shortcut from the drop-down menu or use the default shortcut setting, `<none>`.

The keyboard shortcut assignments range from *Ctrl+0* to *Ctrl+9*.



- B** When you complete adding a bookmark, the bookmark icon appears in the left gutter of the editor adjacent to your bookmark selection. However, when you assign a keyboard shortcut for the bookmark, the keyboard number of the shortcut appears on the bookmark icon.

1
Tip You also can use the context menu of the editor's gutter to add or remove bookmarks. Simply right-click in the gutter, choose Add Bookmark to open the Add Bookmark dialog box and add a bookmark. If you have already added a bookmark, right or left-click the bookmark icon and choose Remove Bookmark to remove the bookmark from the gutter and bookmark list.

Editing bookmarks

To edit bookmarks you created,

- 1 Choose Search|Bookmarks on the main menu or press *Ctrl+Shift+K*.
The Bookmarks dialog box opens.
- 2 Select the bookmark in the Description field.
- 3 Click the Edit button.
The Edit Bookmark dialog box opens.
- 4 Fill in the Description and Shortcut fields.

Viewing bookmarks

You can easily view the bookmark list and go to the bookmark in the editor:

- 1 Choose Search|Bookmarks on the main menu or press *Ctrl+Shift+K*.
The Bookmarks dialog box opens.
- 2 Select the bookmark in the Description field.
- 3 Click the Go button to view the text that you bookmarked in the editor.

Navigating to bookmarks

Use the following keyboard shortcuts to navigate to the previous or next bookmark in the editor (using default CUA keymap assignments).

- 1 Press *Ctrl+Shift+.* (period) to navigate to the next bookmark.
- 2 Press *Ctrl+Shift+,* (comma) to navigate to the previous bookmark.

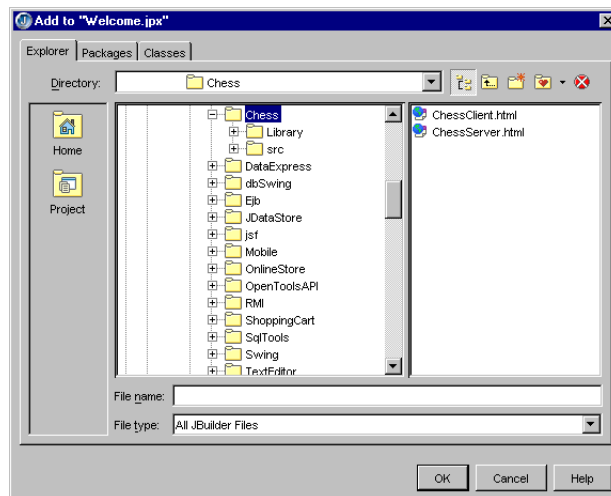
Moving, opening, and adding files

JBuilder makes it easy to move external files into the editor and project pane. You can use drag and drop to bring an external file from outside JBuilder to open the file in the editor. The file must be a type that JBuilder recognizes. To add the file to the project, drag and drop the file into the project pane. Double-click the file to open it in the editor.

Adding files to a project

You can add internal or external files to your project with the Add Files/Packages/Classes button at the top of the project pane or with the project pane context menu.

- 1 Right-click the project file in the project pane.
- 2 Choose Add Files/Packages/Classes.
- 3 Choose a file from the Explorer, Packages, or Classes page of the Add To dialog box.



Tip You can turn off the text drag and drop feature in the Actions page of the Preferences dialog box (Tools|Preferences|Editor|Actions).

Coding shortcuts

JBuilder features numerous coding shortcuts in the editor, such as CodeInsight, ErrorInsight, SyncEdit, TagInsight, ScopeInsight, code folding and code templates. CodeInsight assists with code completion, while ErrorInsight provides quick fixes for compile-time errors for code and markup languages. The SyncEdit tool offers simultaneous code editing.

The TagInsight tool works for HTML, JSP, and XML files, rather than for Java source files like the other coding tools. TagInsight provides tag completion within HTML, JSP, and XML source files, as well as a tag component palette and inspector for tag attributes.

ScopeInsight displays the current code scope for Java source files, in addition to code folding and code folding which provide rapid navigation through large code blocks. Code templates supply common code elements for many file types (default file types are Java, HTML, and JSF) for quick insertion into the editor. For more information about these features, see [Chapter 16, "Using code shortcuts."](#)

Printing support in the editor

You can use the File|Print command to print your source code directly from the editor.

The File|Page Layout command displays the Page Layout dialog box, where you can set layout options:

- **Page Layout** — The options on the Page Layout page let you choose the page orientation, layout, and font, and configure line numbering and wrapping.
- **Advanced** — The options on the Advanced page let you set margins and print page headers. Use the following variables to control what is printed in the page header.

Table 15.2 Page header variables

Variable	Description
%f	Filename and path
%g	Filename only
%p	Page number
%n	Total page count
%t	Time
%d	date (long version)
%s	date (short version)
%u	Username

Chapter 16

Using code shortcuts

JBuilder offers convenient coding tools that supply you with many coding shortcuts. You not only have numerous options for rapidly completing source code and markup languages, but also for finding code elements and correcting code and tag errors. The following tools are accessible from JBuilder's editor in the content pane.

- CodeInsight provides class lists, parameter lists, member lists, and expression evaluation.
- ScopeInsight offers a scoping tool to clearly identify the associated class, method, or expression in your source code.
- Code folding allows swift code navigation through large source files.
- Navigation tools provide quick access to class members in your source code.
- ErrorInsight provides rapid accessibility and immediate solutions for code errors.
- Code templates supply frequently used code and markup language elements for quick insertion into your files.
- SyncEdit offers rapid source and template code editing for duplicate identifiers.
- TagInsight presents a tag inspector and tag completion lists for HTML, JSP, and XML source files. It also includes a tag component palette for HTML and JSP files for tag editing.

Other coding shortcuts include finding the class, definition and references for a symbol, and finding the superclass for an overridden method. All of the “find” tools can be accessed from the context menu of the editor and from keyboard shortcuts (Tools|Preferences|Keymaps|Edit). Some of the Find menu commands are available from the Search menu and the structure pane context menu.

See also

- [“Finding a symbol's definition” on page 119](#)
- [“Finding references to a symbol” on page 120](#)
- [“Finding the overridden method” on page 120](#)

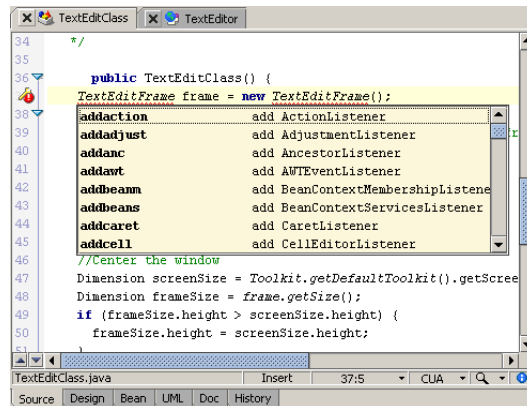
CodeInsight

CodeInsight supplies tools for code completion, such as parameter, class, and member lists, as well as tool tip expression evaluation inside Java files and inside the code segments of JSP files. JBuilder's CodeInsight displays context-sensitive pop-up windows within the editor that show the following:

- MemberInsight — data members and methods for the current context.
- Smart MemberInsight — filters accessible data members and methods for current context.
- ClassInsight — classes accessible through the current class path.
- ParameterInsight — parameters expected for the method being coded.
- Tool tip expression evaluation — values for variables display in the debugger. This is a feature of JBuilder Developer and Enterprise.
- ExpressionInsight — the contents of the expression selected in the debugger. This is a feature of JBuilder Developer and Enterprise.

Important The project must be compiled before imported classes are available to CodeInsight.

Figure 16.1 CodeInsight pop-up window



Invoke the CodeInsight tool with the corresponding shortcut keystroke, select the symbol you want to insert, and press *Enter*.

To access CodeInsight and other coding shortcuts, use these keyboard shortcuts:

Table 16.1 Shortcut keystrokes

Pop-up window	Keystrokes	Actions
MemberInsight	<i>Ctrl+H</i> or <i>Ctrl+Space</i>	Provides methods and members of active scope.
ParameterInsight	<i>Ctrl+Shift+H</i>	Provides method parameters.
ClassInsight	<i>Ctrl+Alt+Space</i> <i>Ctrl+Alt+H</i>	Inserts a class name into a writable file. JBuilder Enterprise provides insertion style options.
CodeInsight/code templates	<i>Ctrl+J</i>	Provides code templates.
Find Definition	<i>Ctrl+Enter</i>	Drills down to the definition of a symbol.
Find References	<i>Ctrl+Shift+Enter</i>	Displays references for a symbol.
SyncEdit	<i>Ctrl+Shift+J</i>	Invokes the SyncEdit tool.

These keystrokes are the JBuilder defaults for CUA keymapping. To view the current list of keystrokes for any keymap, see the Keymap Editor dialog box. To open the Keymap Editor, choose **Tools|Preferences|Keymaps**, choose the keymap you want to view, and click the **Edit** button. In JBuilder Developer and Enterprise, the keystrokes can be configured using this dialog box.

Tip If CodeInsight fails, look for such errors as missing braces or missing import statements.

Use arrow keys to move the selection up and down the list. To accept a CodeInsight selection, press *Enter*, any non-alphanumeric character ([, =, \, etc.), or the space bar.

See also

- [Help|Reference Documentation|Keymaps](#)
- [Chapter 17, “Customizing the editor”](#)
- “Finding a definition” in *Building Applications with JBuilder*
- “Finding local references” in *Building Applications with JBuilder*

MemberInsight

MemberInsight provides a pop-up list of all appropriate method calls for any given reference prefix depending on your CodeInsight settings. The pop-up list appears automatically when you type a dot character (.) in a valid context. You can also invoke the list manually by pressing *Ctrl+H* or by choosing **Edit|CodeInsight|MemberInsight** from the main menu.

The Smart MemberInsight pop-up list appears first if its content is applicable to the current code context, otherwise, the regular MemberInsight pop-up list appears. When you press *Ctrl+H* again, the editor toggles to Smart MemberInsight (or to MemberInsight if the editor was currently displaying Smart MemberInsight).

MemberInsight displays valid method, property, and event names when you type in a class name in the editor. The code completion feature can be used to complete assignment statements. CodeInsight provides automatic code completion, as well. If the remainder of the code statement is unique, CodeInsight automatically fills in the rest of the code and the member list does not appear.

The member list is based on the current project’s imports. Any deprecated methods are highlighted using strike-out. The list is filtered based on the accessibility rules of the Java language.

Smart MemberInsight

Smart MemberInsight filters the MemberInsight pop-up list of members and method calls for the current context. Smart MemberInsight determines the specific member expression type or typecast required for the cursor placement. You can toggle between MemberInsight and Smart MemberInsight by pressing *Ctrl+H* again after initially invoking MemberInsight or Smart MemberInsight.

ClassInsight

ClassInsight invokes the ClassInsight dialog box, which can be used to insert classes into your code. To display the dialog box, press *Ctrl+Alt+Space* or *Ctrl+Alt+H* in the editor. It has two tabs: **Search** and **Browse**. The **Search** page provides a search field and a list of matches for what you type. The **Browse** page provides a search field and the class browser.

Begin entering the name of the class you want to insert in your code in the Search For field. As you type, JBuilder dynamically searches any classes on the current class path and displays a list of possible matches, which changes as you type. Press *Enter* or use double-click to insert the selected class and *Ctrl+Enter* to insert the package. You can also use the arrow keys to navigate the list.

To find classes in any libraries, the libraries must be in your project. When you add a class using ClassInsight, the import statement is automatically added as the last import statement in your code. To reorder and optimize your import statements, use Optimize Imports in the editor. See “Optimizing imports” in *Building Applications with JBuilder*.

Important

If you’ve added a source file or library to your project, you must recompile the project and press the Refresh button in the project pane before ClassInsight can recognize the additions. Use Project|Make Project or Project|Rebuild Project to recompile.

See also

- “Optimizing imports” in *Building Applications with JBuilder*.
- “Adding and configuring libraries” in *Building Applications with JBuilder*.
- “Class path” in *Building Applications with JBuilder*.

ParameterInsight

When you are coding a method call, you can display a list of expected parameters for the method. To display the parameter list, type a method name and press *Ctrl+Shift+Space*, *Ctrl+Shift+H*, or the left parenthesis character (*)*. All possible parameters are displayed, including overloaded ones.

If the source code of the method is available, the parameter names are shown. As you fill in the parameters for the method call, the current parameter is highlighted in the parameter list.

Tool tip expression evaluation

When you are debugging a program, you can place the mouse cursor over any variable in the editor to display its value. The value is displayed in a small pop-up panel that looks like a tool tip.

ExpressionInsight

You can only invoke ExpressionInsight when you are debugging. Suspend the debugger and place the cursor inside an expression to access ExpressionInsight. ExpressionInsight is a small pop-up window that displays the contents of the selected expression in a tree structure.

To display the ExpressionInsight window,

- Hold down the *Ctrl* key and move the mouse over your code in the editor. The ExpressionInsight window displays when the mouse passes over a meaningful expression.
- Move your mouse to the expression you want to see in more detail and press *Ctrl* plus the right mouse button.

See also

- “Debugging Java programs” in *Building Applications with JBuilder*

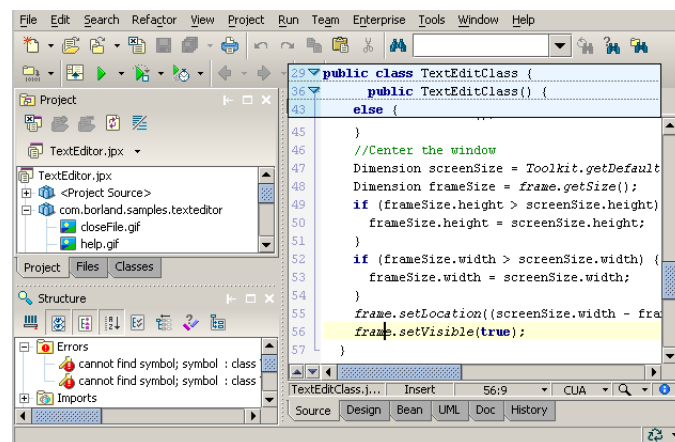
Configuring CodeInsight

To configure and choose display options for CodeInsight, right-click in the content pane, select Editor Preferences, and click the CodeInsight page or choose Tools|Preferences|Editor|CodeInsight. For more information, click the Help button on the CodeInsight page of the dialog box.

ScopeInsight

ScopeInsight is a scoping tool for source code that identifies code nesting levels above the current code element and shows the code context by displaying the associated class, method, or expression in the source code. ScopeInsight presents your location in the open file by displaying the class and method name of the current method within a shaded pop-up box at the top of the editor.

JBuilder offers additional scoping information in the left gutter of the editor. The current scope is displayed with a solid line that runs from the beginning to the end of the method, class, or code block.



Activate ScopeInsight to display the class and method name at the top of the editor by clicking the ScopeInsight icon located at the lower right corner of the editor. When you move your mouse over the ScopeInsight icon, the tool tip, ScopeInsight, appears. Showing the current scope is enabled by default, but can be disabled.

Enabling or disabling ScopeInsight

To enable or disable Show Current Scope,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose the Editor page.
- 3 Check the Show Current Scope check box in the Gutter section of the Editor page.
The current scope displays with a solid line in the left gutter.
- 4 Uncheck the Show Current Scope check box to disable.
The current scope does not display in the gutter.

Locating code members

JBuilder provides another visual aid tool to help you quickly locate code members. Identify unused variables, methods, imports, and find outer class variables and overriding methods in your source code by the members' customized text color or font attribute. Customize these elements on the Color page of the Preferences dialog box.

To locate unused members, outer class variables, and overriding methods,

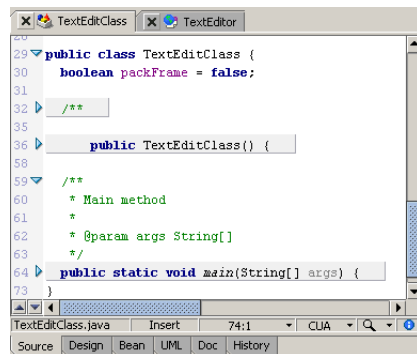
- 1 Open the Color page of the Preferences dialog box (Tools|Preferences|Editor|Color).
- 2 Click the Java tab.
- 3 Select Overriding Method or Unused Variable in the Screen Element list box.
- 4 Choose either bold or italic attributes for the text.
- 5 Choose foreground and background color selections to identify the code element.

The default color for unused variables (includes methods and imports) is gray, and the default attribute for overriding methods is italic.

Code folding

Code folding compresses and expands code, markup language, and comment blocks making it much easier to navigate through large files. The code folding tool also provides a quick-viewing tool tip that displays a full view of the folded source code, text, or comment when you move the mouse over the line of folded code.

JBuilder enables code folding by default, but you can disable the code-folding tool. You can also choose a keyboard shortcut for code folding actions from the Keymap Editor (Tools|Preferences|Keymaps|Edit).



Assigning code folding shortcuts

If you'd like to create quick keyboard access for code folding actions, you can assign a keyboard shortcut to fold code for easier navigation and code visualization.

To assign a code folding shortcut,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose the Keymaps page.
- 3 Choose the type of Keymap you want to use.

- 4 Click the Edit button on the Keymaps page.
The Keymap Editor dialog box opens.
- 5 In the Action column of the table, choose the Code Folding folder in the tree.
- 6 Click the plus (+) sign or double-click the Code Folding folder to expand the code folding keymap choices.
- 7 Choose one of the following actions for a new keystroke.
 - Fold-All
 - Toggle-Current-Fold
 - Unfold-All
 - Unfold-Nearest
- 8 Click the Add button.
- 9 Fill in the New keystroke dialog box with the event type, keystroke combination, properties, and key codes and click OK.

Accessing code folding options

You can choose Edit|Code Folding on JBuilder's main menu or right-click the Editor's gutter to access the following code folding options:

- Enable Code Folding
- Fold All
- Unfold All
- Unfold Nearest
- Toggle Current Fold

Folding code and comment blocks and displaying tool tips

- 1 Click the down arrow in the editor's left gutter, adjacent to the code or comment block.
The code or comment block folds and only the first line displays.
- 2 Hover the cursor over the line of folded code or comment.
The expanded code or comment tool tip displays in the editor.
- 3 Click anywhere outside of the tool tip to close the tool tip.
- 4 Click the right arrow adjacent to the folded block to expand the code or comment.

Disabling code folding

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose the Editor page.
- 3 Uncheck the Enable Code Folding check box under the Gutter section on the Editor page.

Navigating to class members

The editor and JBuilder's IDE provide helpful navigation tools to increase your coding speed. The up and down arrows located in the left lower gutter of the editor allow you to move up or down methods in the source code. Click the up arrow to move to the previous method, or the down arrow to move to the next method in the source code. You can also use the Go To Class Member dialog box (Search|Go To Class Member), to view a list of class members or search for class members and navigate to that member in the editor.

Navigating to class members using the main menu

To navigate to class members using the main menu,

- 1 Choose Search|Go To Class Member to open the Go To Class Member dialog box.
- 2 Type in the class member name in the Name field or select a class member from the Member List field.
- 3 Click OK.

The editor displays the class member location and highlights the class member line.

ErrorInsight

ErrorInsight helps you quickly access and correct many code errors. ErrorInsight is active for source code, HTML, JSP, and Struts. ErrorInsight provides wizards, pop-ups, refactoring, and TagInsight to help you correct code and tag errors. As you type your source code or text into the editor, errors appear dynamically in the structure pane and are also underlined with a red squiggly line in the editor. Code errors also display in the structure and message panes after compiling.

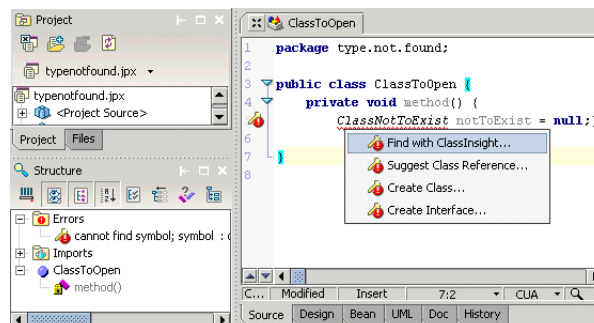


ErrorInsight icons signify the errors that you can correct with the ErrorInsight tool and appear adjacent to the error in the editor and structure pane. Error icons (errors that can't be corrected with ErrorInsight), appear adjacent to the error in the left gutter of the editor and in the structure pane. When you place your cursor on the ErrorInsight icon in the editor's gutter, the tool tip, "Click to invoke ErrorInsight (*Alt + Enter*)", appears.

To access ErrorInsight to correct an error you can,

- Right-click the error in the structure pane.
- Place your cursor in the source code text in the editor and press *Alt+Enter*.
- Click the ErrorInsight icon in the left gutter of the editor adjacent to the error.
- Choose Edit|CodeInsight|ErrorInsight from the main menu.

Shown here is an example of menu options available to fix the Type Not Found code error.



The menu commands are determined by the error you choose to correct. Choose whatever command best suits your error solution. The table below displays solutions for code errors; however, ErrorInsight also provides solutions for markup languages, including commands for suggesting or removing attributes.

Table 16.2 ErrorInsight menu options for source code

Code error	Menu options	Menu action
Class name or file name mismatch	Rename Class	Renames class to match file name.
	Rename File	Renames file to match class name.
Class not found in import	Remove Import	Removes import from source code.
	Create Class	Opens Class wizard.
	Create Interface	Opens Interface wizard.
Class should be declared abstract	Make Class Abstract	Declares the class abstract in the source code.
	Implement Methods	Adds required methods to the source code.
Constructor not found	Create Constructor	Creates constructor in source code.
	Override Constructor	Overrides constructor in source code.
Exception not caught	Add Throws Clause	Adds throws clause to source code.
	Add Catch Clause	Adds another catch clause to the existing try/catch block to handle the uncaught exception.
	Surround with Try/Catch	Surrounds exception with try/catch statement.
Method not found	Create Method	Opens Create Method dialog box.
	Suggest Method Reference	Opens Choose Method dialog box.
	Rename to Method	Uses refactoring to rename the method.
Package or directory name mismatch	Change Package Declaration	Changes package declaration to match source code.
	Move File	Moves file to correct folder.
Package not found in import	Remove Import	Removes import statement from source code.
	Create Package	Opens Create Package wizard.
Type mismatch	Add Type Cast	Inserts a cast for the correct type.
Type not found	Find with ClassInsight	Opens ClassInsight dialog box.
	Suggest Class Reference	Opens Suggest Correction dialog box with list of similar type names.
	Create Class	Opens Class wizard.
	Create Interface	Opens Interface wizard.
Variable not found	Create Field	Opens New Field dialog box.
	Create Variable	Opens Create New Variable dialog box.
	Suggest Variable Reference	Opens Suggest Variable Name dialog box with list of similar variable names.
Symbol not found	Find with ClassInsight	Opens ClassInsight dialog box.
	Suggest Class Reference	Opens Suggest Correction dialog box with list of similar type names.
	Create Class	Opens Class wizard.
	Create Interface	Opens Interface wizard.
	Refactor Class	Opens the Refactor Class dialog box with the old name and package and browsing options for the new name.

If you choose a menu command that opens a dialog box or wizard, fill in the appropriate fields and click OK. For more information, click the Help button in the wizard or dialog box.

Code templates

Code templates are snippets of frequently used code and markup language elements that you can insert into your files to save repetitive typing. You can use code templates in the editor to expedite the coding process. Java, HTML, JSF, and common code templates are available by default; Java templates also include import statements when they are needed.

You can add any type of code template on the Common page of the Preferences dialog box (Tools|Preferences|Editor|Templates|Common). The common template displays regardless of file type or location. The templates presented are determined by the file type under edit, except common templates.

You can also define templates for a specific file type to first create a template group by using the Templates page (Tools|Preferences|Editor|Templates) of the Preferences dialog box. When you insert a template into your code, the code template feature automatically formats the inserted template and your source code.

Preferences for code templates are also available from the Editor Preferences dialog box (right-click in the editor and choose Editor Preferences). The following topics contain instructions for using numerous code template features that can increase your productivity.

See the following sections for directions to add templates to your code, add file types to the code templates file type list, add import statements to your code using templates, or add template macros to code templates.

- [“Adding code templates to code by typing the code template name” on page 139](#)
- [“Adding code templates to code using the cursor” on page 139](#)
- [“Adding code template file types” on page 139](#)
- [“Adding import statements to your code using auto import” on page 140](#)
- [“Adding template macros to code templates” on page 140](#)

If you want to create your own code templates, surround your code with templates, or import or export code templates, see the following sections.

- [“Creating code templates” on page 140](#)
- [“Creating code templates from source code” on page 141](#)
- [“Surrounding source code with code templates” on page 141](#)
- [“Importing or exporting code templates” on page 142](#)

With JBuilder, you have the ability to edit JBuilder’s default code templates and your own templates or to create tabbed navigation within the template. The following sections tell you how to accomplish these tasks.

- [“Editing code templates” on page 142](#)
- [“Setting tabs within code templates” on page 143](#)

Adding code templates to code by typing the code template name

You can add a code template to your code by typing the code template name into your source code. Once you add a template to your code, you are in the SyncEdit mode. See [“Editing with SyncEdit” on page 143](#) for more information about using the SyncEdit mode.

- 1 Type the code template name, such as `classp`, into your code where you want the code to appear.
- 2 Press *Ctrl+J*.

If your typed entry fully matches or partially matches the template, JBuilder inserts the code template. Also, for Java files, the editor adds the template's import statement to the top of the source file. Otherwise, the code template list opens.

- 3 If the code template list opens, select the template and press *Enter*.

See Tools|Preferences|Editor|Templates for a complete list of template names.

Adding code templates to code using the cursor

You can add code templates to your code by positioning the cursor where you want the template to appear in your code and using keystrokes.

- 1 Position the cursor where you want the code to appear and press *Ctrl+J* to display a list of code templates.
- 2 Select from the list using the arrow keys and press *Enter*.

The editor automatically expands the template. Also, for Java files, the editor adds the template's import statement to the top of the source file.

Adding code template file types

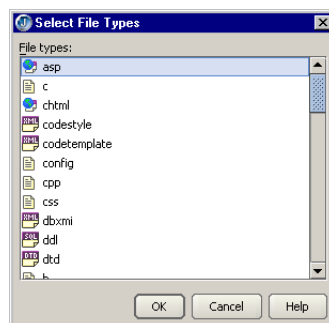
Expand your code template functionality by defining additional file types to associate with code templates. The default types are Java, HTML, and JSP. You don't need to write an additional OpenTool, simply start by following these instructions:

- 1 Choose Tools|Preferences|Editor|Templates.
- 2 Click the Add button.

The Select File Types dialog box opens.

- 3 Choose the file type from the File Type list to add to your code templates group.

Optional: Open the Templates page of the Preferences dialog box to see the added file type displayed in the File Types With Custom Templates list.



Adding import statements to your code using auto import

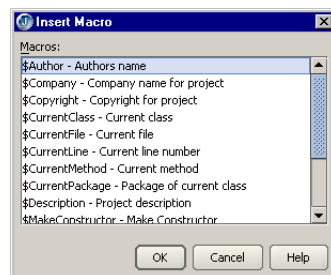
Code templates for Java include the appropriate import statement at the top of the code snippet (Tools|Preferences|Editor|Templates). When you insert the code template into your source code using the keyboard shortcut, *Ctrl+J*, the import statement is stripped from the template and correctly placed at the top of your source code.

Adding template macros to code templates

Template macros can automate the process of replacing identifier names in the source code when you insert a code template into your code. For example, if you select the `equals` template (includes the `$CurrentClass` macro) to insert into your code, the `$CurrentClass` template macro is replaced by the actual class name in your source code.

To add template macros to code templates,

- 1 Choose Tools|Preferences|Editor|Templates.
 - 2 Expand the Templates node.
 - 3 Choose one of the nodes within the Templates node; the default pages are Java, HTML, JSF, and Common.
- Note** The Macro button is grayed out on the Common page until you add a Common template.
- 4 Place the cursor in the Code text field where you want to insert the template macro. Click the Macro button on the bottom of the Templates page. The Insert Macro dialog box opens.
 - 5 In the Insert Macro dialog box, select the template macro from the list and click OK. The template macro is inserted into the code template.
 - 6 If you decide that you want to return to the original template, click the Restore Original button.



Creating code templates

Create your own code templates for Java, HTML, JSF, and Common code types to replace your frequently used pieces of code. By creating your own unique code templates you can streamline your coding, replacing lengthy and repetitive typing with a few, quick keystrokes. Follow these instructions to add your own customized template.

- 1 Choose Tools|Preferences|Editor|Templates.
- 2 Expand the Templates node.
Choose the Java, HTML, JSF, or Common page for the type of template you want to add.

The Common templates include all added template types and are always displayed regardless of file type or location within that file.

- 3 Click the Add button.
- 4 Enter the name, description, and code for the new code template, then click OK.
Remember to include the appropriate import statement.
- 5 Add or edit other templates, then click OK when you are finished to close the Preferences dialog box.

Creating code templates from source code

Use a quick and easy method to create new code templates directly from your source code.

- 1 Select the required text for the new template from your source code.
- 2 Right-click in the selected area. (The default color of the selected background is blue.)
- 3 Choose Add|Code Template.
The New Code Template dialog box opens with your selected code displayed in the Code text area.
- 4 In the New Code Template dialog box, enter a name and description for your new template.
- 5 Choose a file type for your template. The defaults are Java, HTML, JSF, and Common.
- 6 Click the Macro button if you want to add a macro to your new template.
- 7 Click OK and the editor adds your new template to your code template list.

Note To view your new code template, choose Tools|Preferences|Editor|Templates, expand the Templates node, and choose the page for the type of template you created (Java, HTML, or Common).

Surrounding source code with code templates

You can use code templates to surround your code, instead of inserting the templates within your code. For example, you can use the `ifb` (if statement) code template to surround a piece of appropriate code for that statement.

```
if (|) {
    |$Selection
}
```

To surround your code with code templates,

- 1 Select the code that you want to surround with a template.
You can begin your selection at the left margin and move the cursor down past the last line of code that you want to include.
- 2 Press `Ctrl+J`.
The code template list opens.
- 3 Double-click the code template from the code template list.
The code template surrounds your code without deleting it and correctly indents your code.

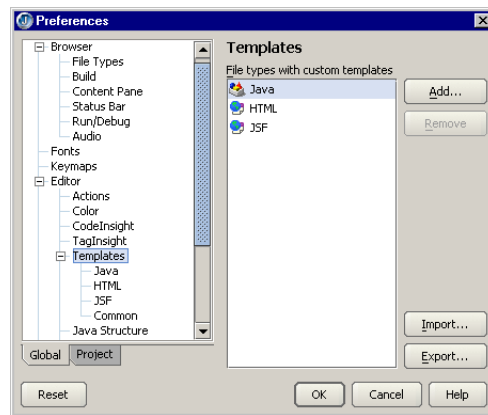
Tip Once you've expanded the template, use CodeInsight to assist you in writing your code. See [“CodeInsight” on page 130](#).

JBuilder's pre-defined code templates are formatted according to your project preferences. The Block Indent and Braces options are set in the Basic or Block pages of the Project Properties|Formatting dialog box.

Importing or exporting code templates

You can import or export code templates as XML files with a `.codetemplate` extension. Access the dialog box to import or export code templates by following these steps:

- 1 Choose Tools|Preferences|Editor|Templates.
- 2 Click the Import or Export button on the Templates page.
- 3 Browse to the necessary directory and choose the template file name.



Editing code templates

You can edit your own and JBuilder's default code templates or delete any code templates you don't need.

- 1 Choose the Templates page of the Preferences dialog box (Tools|Preferences|Editor|Templates).
- 2 Expand the Templates node.
- 3 Choose a node below the Templates node for the type of template you want to edit (Java, HTML, JSF, or Common).
- 4 Click the Edit button on the specific template page (Java, HTML, JSF, or Common) to change the name or description of the selected template.
- 5 Enter the name and description of the new code template, then press OK.
- 6 Type in the Code text editing area to modify the code sample.
Remember to include the appropriate import statement.
- 7 Edit other templates, then click OK when you are finished to close the Preferences dialog box.
- 8 If you decide that you want to return to the original template version, click the Restore Original button.

Important When you create or modify templates, JBuilder does not format them according to the preferences. You must set the braces and indents manually.

Setting tabs within code templates

Create easier keyboard navigation within template blocks by setting tabs within the template.

To set up tabbed navigation within the code template,

- 1 Choose Tools|Preferences|Editor|Templates.
- 2 Expand the Templates node.
- 3 Choose a node below the Templates node for the type of template you want to edit (Java, HTML, JSF, or Common).
- 4 Click the template name in the code template list to select the one you want to edit.
- 5 Type in one or more bars (|) in the Code text editing area where you want to set the tabs and click OK.

SyncEdit

SyncEdit is a convenient editing tool that allows you to simultaneously edit duplicate identifiers in source code, markup languages, and code templates (Tools|Preferences|Editor|Templates). You can also customize SyncEdit screen elements for easier viewing.

You can invoke SyncEdit with the following actions.

- Select code in the editor (if the code contains duplicate identifiers).
- Insert code templates into your source code with the *Ctrl+J* or *Ctrl+Shift+J* keystrokes.
- Choose Edit|SyncEdit from the main menu.

See the following instructions for editing and customizing using the SyncEdit tool.

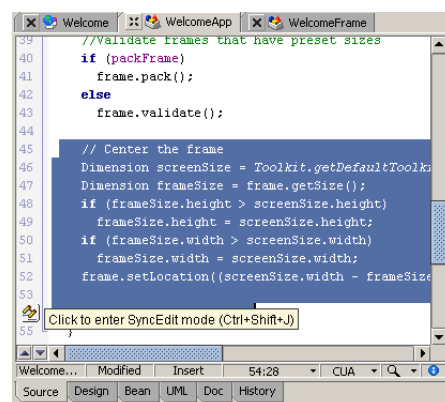
- [“Customizing SyncEdit screen elements” on page 144](#)
- [“Customizing SyncEdit screen elements” on page 144](#)

Editing with SyncEdit



You can only use SyncEdit for editing if the text contains duplicate identifiers. When SyncEdit is invoked, the SyncEdit icon appears in the editor's left gutter. However, if no identifiers are duplicated in the code, markup language, or code template selection, the SyncEdit mode is not invoked.

Figure 16.2 Example of SyncEdit text selection



To modify code, markup languages, and Java, HTML, JSF, or Common code templates,

- 1 Select the text block in the editor or in the code template.



The SyncEdit icon appears in the left gutter if duplicate identifiers are present.

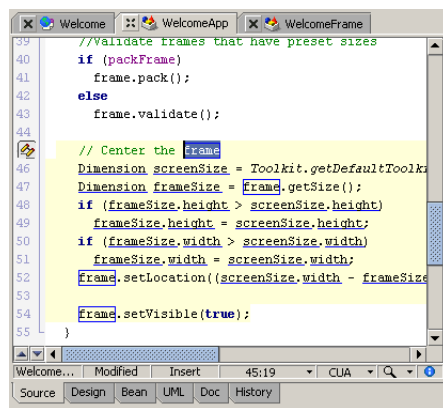
- 2 Click the SyncEdit icon in the left gutter of the editor or press *Ctrl+Shift+J* to enter the SyncEdit mode.

- 3 Edit the duplicate identifier you want to modify.

Notice that all of the duplicate identifiers within your selected region are edited simultaneously. You can also tab to the other underlined identifiers and edit the identifiers and their duplicates while in the SyncEdit mode.

- 4 Click the SyncEdit icon to exit the SyncEdit mode.

You can also exit the SyncEdit mode by moving the cursor outside of the selected region and clicking, pressing *Esc*, or pressing *Ctrl+Shift+J*.



For more information, click the Help button on the Templates page of the Preferences dialog box (Tools|Preferences|Editor|Templates).

Customizing SyncEdit screen elements

Customizing SyncEdit screen elements allows you to improve their identifiability. You can customize the following SyncEdit screen elements: code identifiers, SyncEdit background, and SyncEdit highlight. You can easily change the style or color of the SyncEdit screen element in the editor (Tools|Preferences|Editor|Color).

To customize SyncEdit code elements,

- 1 Choose Tools|Preferences|Editor|Color.
- 2 Select a SyncEdit element from the Screen element list.

The SyncEdit element options are Identifier, SyncEdit background, or SyncEdit highlight.

- 3 Choose from the style and color choices.

- Attributes: click the Bold or Italic check box.
- Foreground: click one of the colors for your foreground color or click the Use Default check box.
- Background: click one of the colors for your background color or click the Use Default check box.

Note See the result of your changes in the text field below the Screen element box.

- 4 Click OK to save the changes and close the Preferences dialog box.

TagInsight

This is a feature of
JBuilder Developer
and Enterprise

TagInsight provides tools for tag completion within HTML, JSP, and XML source files. JBuilder's TagInsight displays an automatic pop-up window sensitive to the file type opened in the editor and the current cursor location in the file. The window displays a list of tag options for that particular file type. The list displayed is determined by the current context at the cursor's location.

TagInsight displays only the valid tags, elements, or subelements in the list. If there is only one valid element at the current cursor position, no list appears, and TagInsight inserts the element into the text. You can turn this feature on or off on the TagInsight page of the Preferences dialog box (Tools|Preferences|Editor|TagInsight). When you choose a tag or element, TagInsight closes the tag, and when you choose an attribute, TagInsight adds an equal sign (=) followed by double quotes with the cursor positioned between the quotes.

TagInsight displays context-sensitive pop-up windows within the editor that show the following:

- **ElementInsight** — lists the available elements.
- **AttributeInsight** — lists the available attributes for the selected element.
- **ValueInsight** — lists the available enumerated values for the selected attribute.
- **EntityInsight** — lists general entities.

To invoke TagInsight, use the MemberInsight keystroke, which is *Ctrl+H* or *Ctrl+Space* in the CUA keymapping. Keystrokes for other editor keymappings are listed in the Keymap Editor (Tools|Preferences|Keymaps).

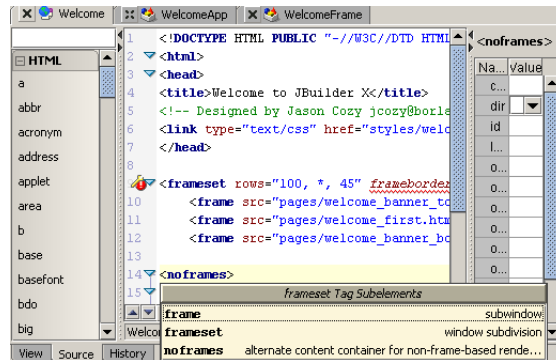
When you invoke TagInsight, it attempts to invoke the appropriate context-sensitive window, such as ElementInsight, at the cursor location. If the automatic pop-up features are enabled, you can type a left-angle bracket (<) or an ampersand (&) character in the editor to automatically display TagInsight. Entering a space or an = (equals) sign inside of tags also activates the timer. To commit a TagInsight selection, press *Enter*.

In addition, depending on the context, other keys commit the selection in different ways:

- **ElementInsight** — pressing the space bar will insert the selected element and immediately invoke AttributeInsight. If you are typing a new tag and know that you want to include an attribute, use the space bar. With *Enter*, the cursor is placed after the start tag. If you press the / (forward slash), then the element will be inserted as a single empty element, instead of a start/end tag pair.
- **AttributeInsight** — pressing = (equals) is the same as *Enter*. This simply mimics the natural typing of an attribute. Pressing the space bar also commits the selected attribute.
- **EntityInsight** — if a particular character or key sequence generates a character that has a corresponding entity, that entity will be inserted. In this case, the space bar is considered to be the non-breaking space (the Unicode character U+00A0). So in an HTML file, to insert (non-breaking space), you could type & and the space bar. To insert & (ampersand), type &&.

To cancel TagInsight, press the *Esc* key.

Figure 16.3 ElementInsight pop-up window



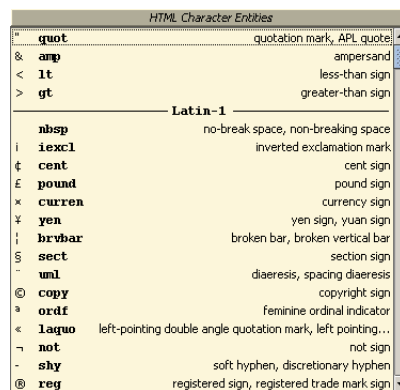
EntityInsight

TagInsight also provides EntityInsight, a list of entities for HTML, JSP, and XML files. To display a list of available entities, invoke TagInsight with the MemberInsight command or enter the *&* (ampersand) character in the editor if the auto pop-up feature is enabled.

Entities vary according to the file type open in the editor. For example, HTML files contain built-in, character, and general entities. All of these entities begin with an ampersand (&). Built-in entities include *&* (&), *<* (<), *>* (>), and *"* ("). Character entities replace characters that are difficult to type, such as the trademark character (TM), other punctuation marks, mathematical operators, and non-ASCII characters.

In addition to these entities, XML documents might also contain general and parameter entities. General entities must be defined in the DTD and can be used as reusable sections of replacement text to replace characters, text, or documents. General entities also begin with an ampersand (&) character. Parameter entities, which begin with a percent sign (%), also allow you to create reusable sections of replacement text but are only allowed in the DTD. TagInsight only supports general entities.

Figure 16.4 EntityInsight pop-up window



XML TagInsight

XML TagInsight is available in XML documents that are defined by a DTD or schema and in Ant build files. Although Ant build files aren't defined by a DTD or schema, TagInsight is able to build the completion list with Java reflection.

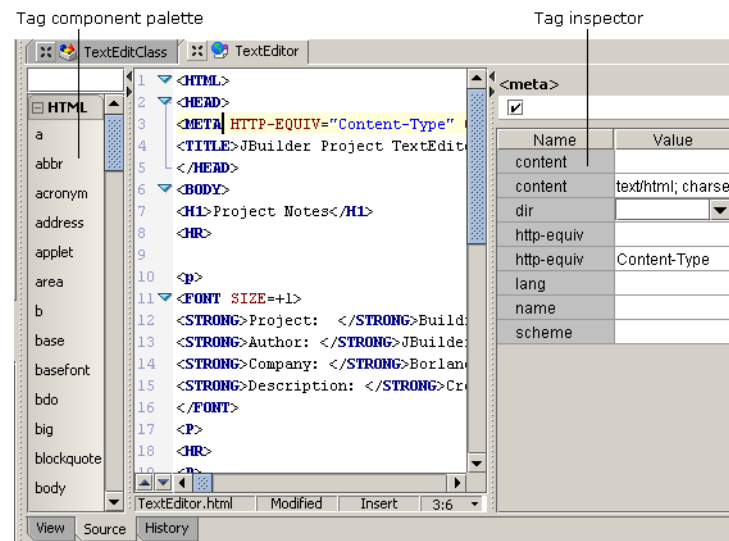
See also

- “Working with XML in the editor: XML TagInsight” in *Working with XML*

Tag inspector

The editor includes a convenient tag inspector for editing HTML, XML, and JSP files, located to the right of the source pane of the open file. The tag inspector is context-sensitive and displays only the appropriate tag properties for the present cursor location. The tag inspector provides an explanatory tool tip when you hover the cursor over a property name.

To add or edit a tag property, you can either enter a value or choose a value from the drop-down menus of attribute values. TagInsight automatically adds the tag attribute to your tag text. If your file includes tag errors, you can view a list of the errors in a Markup Errors folder in the structure pane. See [“Correcting tag errors” on page 148](#) for information about using ErrorInsight to correct markup language tag errors.



Tag component palette

The editor provides a component palette for HTML and JSP files. The component palette is located on the left side of the opened file in the Source pane and displays a list of the available tags and tag libraries to add to your file.

The component palette includes a Search field where you can enter text to find a particular tag and descriptive tool tips that appear when you hover the cursor over the tag. It also allows you to add tags to your file by clicking the tag name in the component palette, then clicking in the editor where you want to add the tag.

If the added tag requires an attribute, the ErrorInsight icon appears adjacent to the tag in the editor, and information about the error appears in the Markup Errors folder in the structure pane. For more information about correcting markup language tag errors, see [“Correcting tag errors” on page 148](#).

Use the context menu to expand and collapse tag groupings in the component palette. Right-click in the palette and choose from the following display options for easier navigation:

- Expand All Groups
- Collapse All Groups
- Show Icons Only

To drop a component into a HTML or JSP file,

- 1 Click the desired HTML, JSP, or JSF component in the tag component palette, located on the left side of the editor.
- 2 Click the desired location in the HTML or JSP source to drop the component.

See also

- Tag component palette in *Developing Web Applications*

Correcting tag errors

When your file contains HTML and JSP tag errors, you can correct them quickly with the context-sensitive ErrorInsight tool. The markup language errors appear in a Markup Errors folder in the structure pane.

To correct tag errors in HTML and JSP files,

- 1 Click the tag error listed in the structure pane in the Markup Errors folder.
- 2 Click the ErrorInsight icon in the editor’s gutter adjacent to the error.
- 3 Choose from the pop-up window options: Suggest tag, Suggest Attribute, Add Missing Attribute, or Remove Attribute to repair the tag error.

Customizing TagInsight

JBuilder offers quick access to TagInsight from the keyboard with the MemberInsight keystroke (*Ctrl+Space* for CUA key bindings). Use the Keymap Editor dialog box (Tools|Preferences|Keymaps|Edit) to modify TagInsight’s keyboard command. Find the MemberInsight keystroke listed under CodeInsight in the Keymap Editor dialog box.

TagInsight’s properties are customizable. You can tailor TagInsight’s many options on the TagInsight page of the Preferences dialog box (Tools|Preferences|Editor|TagInsight).

Set the following options on the TagInsight page of the Preferences dialog box:

- Enable the auto pop-up feature for ElementInsight, AttributeInsight, ValueInsight, and EntityInsight.
- Adjust the auto pop-up delay for TagInsight.
- Set automatic completion options.
- Hide out of place items.
- Generate required attributes.

To customize TagInsight font and color,

- 1 Open the TagInsight page of the Preferences dialog box (Tools|Preferences|Editor|TagInsight).
- 2 Click the Display Options button on the TagInsight page.
The TagInsight Display Options dialog box opens.
- 3 Click the Use Custom Font and Colors check box.
- 4 Choose fonts and colors for tags in the Sample box.

You can also choose display options for TagInsight font and colors by clicking the Display Options button on the TagInsight page of the Preferences dialog box and choosing custom font and colors.

Customizing markup language tags

You can customize tag elements for HTML, XML, and JSP markup languages on the Colors page of the Preferences dialog box (Tools|Preferences|Editor|Color).

Set the following options on the Color page of the Preferences dialog box:

- Modify the colors of screen elements.
- Modify the foreground and background colors.
- Change font attributes.

Customizing markup language editing

You can choose to include the ErrorInsight tool and non-standard HTML attributes while editing markup languages, as well as turn on or off the tag inspector and tag component palette.

To customize markup language editing,

- 1 Choose Tools|Preferences|Editor to open the Preferences dialog box.
- 2 Choose Markup Editing in the tree to open the Markup Editing page.
- 3 Choose from among the options in the page:
 - ErrorInsight for markup languages
 - Non-standard HTML attributes
 - Tag component palette for HTML and JSP
 - Tag inspector for HTML, XML, and JSP
 - JSF expression depth

Customizing markup language formatting

When you are working in markup language files, you can customize the indentation and attribute alignment for easier editing.

To customize markup language formatting,

- 1 Choose Tools|Preferences|Editor to open the Preferences dialog box.
- 2 Choose Markup Formatting in the tree to open the Markup Formatting page.
- 3 Choose from among the options in the page:
 - Indentation for tabs and spaces
 - Spaces between attributes
 - Attribute alignment on separate lines

Chapter 17

Customizing the editor

You can customize how the editor appears and works in many ways. For example, you can change the font and font size used in the editor, enable or disable line numbering, specify your favorite colors for editor elements (such as unused variables), determine how you want elements in the structure pane to appear, and customize how common keystrokes behave.

When you customize editor behavior you can choose how you want various CodeInsight features to work, where and how to position file tabs, how to display white space characters, and edit and create your own code templates, and much more.

Use the Preferences or the Editor Preferences dialog box to customize the editor. To access the Preferences dialog box, choose **Tools|Preferences**, then choose the **Editor** page. To access the Editor Preferences dialog box, right-click in the editor and choose **Editor Preferences**. For complete information about all the options available to you, click the **Help** button on the various pages in the **Editor** category of the Preferences dialog box.

See the following topics for information about keymaps and customizing the editor.

- [“Setting keymapping preferences” on page 151](#)
- [“Customizing the editor’s look and actions” on page 155](#)
- [Chapter 20, “Setting JBuilder preferences”](#)

Setting keymapping preferences

Choose one of the following topics for more details about configuring keymappings.

- [“Selecting a keymapping for the editor” on page 152](#)
- [“Editing keymaps” on page 152](#)
- [“Constructing new keymaps” on page 153](#)
- [“Creating keymap reference copies” on page 154](#)
- [“Searching for keymap actions” on page 155](#)

Selecting a keymapping for the editor

Keyboard shortcuts make certain tasks much faster and easier to do. Different editors map different keystrokes to the same action. This makes switching between editors inconvenient at best, risky at worst.

Because different programmers have different habits and preferences, JBuilder provides a number of editor emulations. Choose the one that you're most comfortable with:

- Brief
- CUA
- Emacs
- Macintosh
- Macintosh CodeWarrior
- Visual Studio

For more information about the keymaps associated with each editor emulation, refer to "Keymaps for editor emulations" in the online help or choose Help|Reference Documentation|Keymaps from JBuilder's main menu.

When you first install JBuilder for Windows, Linux, or Solaris, the CUA keymapping will be in effect.

To select a keymapping scheme for the editor,

- 1 Click the drop-down arrow just to the left of the magnifying glass at the right side of the status bar.
- 2 Choose the keymapping scheme of your choice.

You can also access the Keymaps page of the Preferences dialog box (Tools|Preferences|Keymaps) to change the keymapping scheme.

To select a keymapping scheme from the Preferences dialog box,

- 1 Choose the Keymaps page of the Preferences dialog box (Tools|Preferences|Keymaps).
- 2 Click the radio button adjacent to the editor type from the Keymapping list.
- 3 Click OK.

The new keymap emulation is active immediately.

Editing keymaps

Customize any keymap editor emulation or build new keymaps to your specifications to increase keyboard productivity. The Keymap Editor allows you to edit selected keymaps or build new keymaps.

To edit keymaps with the Keymap Editor,

- 1 Choose the Keymaps page of the Preferences dialog box (Tools|Preferences|Keymaps).
- 2 Select the editor emulation from the Keymaps list.
- 3 Click the Edit button adjacent to the Keymaps list.

The Keymap Editor dialog box opens.

- 4 Expand the folders in the Action column to view the associated keystrokes.
- 5 Select the key binding command you want to change, remove, or add.
- 6 Click the Change, Add, or Remove button.

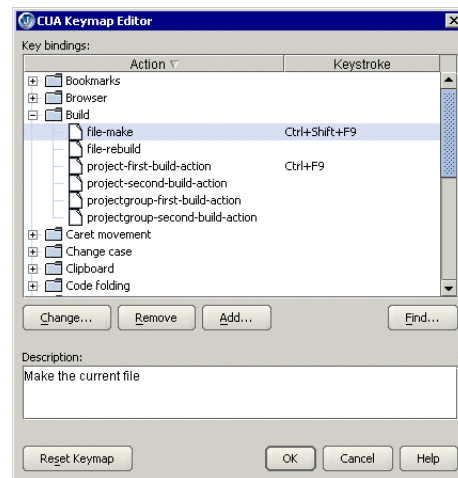
The editor removes the keystroke combination when you click the Remove button or opens the New Keystroke dialog box when you click the Change or Add button.

- 7 Enter the new keystroke combination in the New Keystroke dialog box and click OK.

The Keymap Editor adds the new keystroke to the table (keymap).

If you want to return to the original keymap settings, click the Reset Keymap button. You also can change the sort order of the columns in the Keymap Editor table by clicking the table headers.

Figure 17.1 CUA Keymap Editor



Constructing new keymaps

The Keymap Editor allows you to construct your own customized keymapping to help you increase your keyboard efficiency. You can create a new keymapping where you customize and add to the most basic, common key bindings from all of the keymap editor types.

To create a new keymap,

- 1 Choose Tools|Preferences|Keymaps to open the Keymaps page of the Preferences dialog box.
- 2 Click the New button.

The Keymap Editor opens.

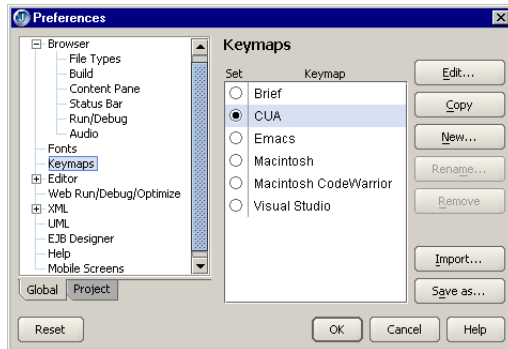
- 3 Expand the folders in the Action column.
- 4 Use the Change, Add, or Remove buttons to customize the basic key bindings.
- 5 Click OK to return to the Keymaps page.

The new keymap presents in the Keymaps list as “untitled”.

- 6 Click the Rename button to choose a name for your new keymap.

Creating keymap reference copies

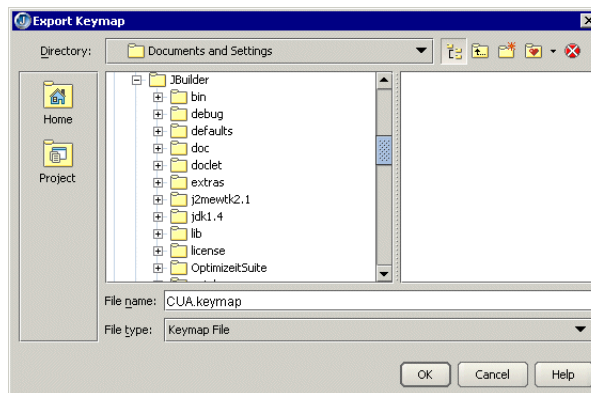
Copy individual keymap assignments easily and save them in text or keymap file format. You can access the keymap Save As button from the Keymaps page of the Preferences dialog box (Tools|Preferences|Keymaps).



To copy individual keymaps,

- 1 Choose Tools|Preferences|Keymaps.
- 2 Select the keymap editor emulation (that you want to copy), from the Keymaps list.
- 3 Click the Save As button.

The Export Keymap dialog box opens.



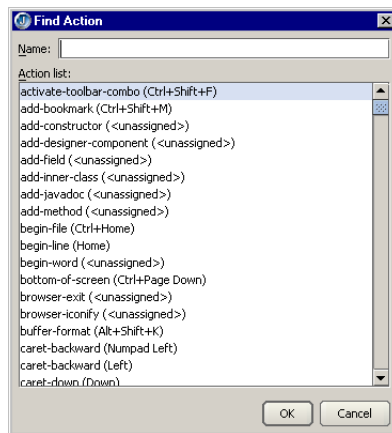
- 4 In the Export Keymap dialog box, choose the directory or folder where you want to place the reference copy.
- 5 Enter the name for your text file in the File Name field.
- 6 Choose the text or keymap file extension from the File Type drop-down menu.
Choose text if you want the file for reference and keymap if you plan to import the keymap from another location.
- 7 Click OK to copy the current contents of the keymap table.
- 8 Choose any text editor to open and view the keymap reference file.

Searching for keymap actions

The list of keymap categories and actions in the Keymap Editor dialog box is lengthy. So, when you want to quickly find a specific keymap action and associated keystroke, all you have to do is use the Find button.

To search for keymap actions,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose Keymaps to open the Keymaps page.
- 3 Click the Edit button to open the Keymap Editor dialog box.
- 4 Click the Find button to open the Find Action dialog box.
- 5 Enter either the action name, its description, or menu item name (if it has one) in the Name field or use the arrow keys to scroll up or down in the Action list to choose the name of the keystroke action.
- 6 Click OK and the focus moves to the keystroke action in the Keymap Editor dialog box with the action highlighted.



For more information about using the Keymap Editor dialog box, click the Help button on the dialog box.

Customizing the editor's look and actions

Choose from the following topics for details about making editing easier with an editor customized to your specifications.

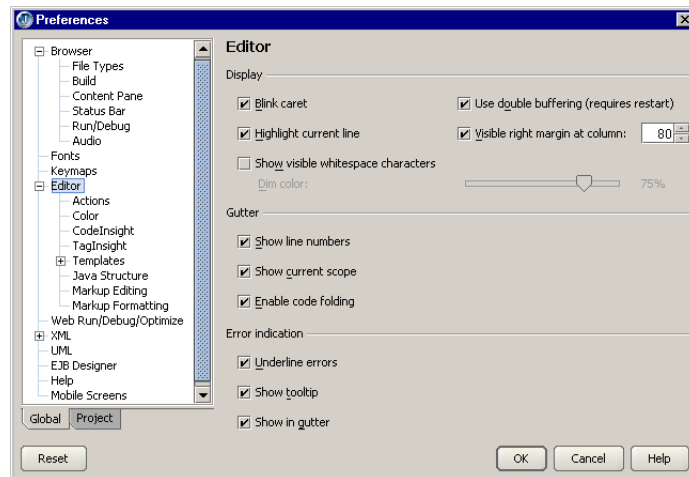
- [“Customizing editor display” on page 156](#)
- [“Customizing editor actions” on page 156](#)
- [“Splitting the source view” on page 157](#)
- [“Displaying line numbers” on page 158](#)
- [“Displaying white space characters” on page 158](#)
- [“Positioning file tabs” on page 159](#)
- [“Limiting open files” on page 159](#)
- [“Customizing unused variables” on page 159](#)

Customizing editor display

Choose from among the editor's display options, including the gutter and error indication display to customize your work environment to suit your working style.

To customize editor display,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Click the Editor node to open the Editor page.
- 3 Click the check boxes to choose options in the Display, Gutter, and Error Indication sections.
- 4 Click the OK button to close the dialog box and activate the settings.



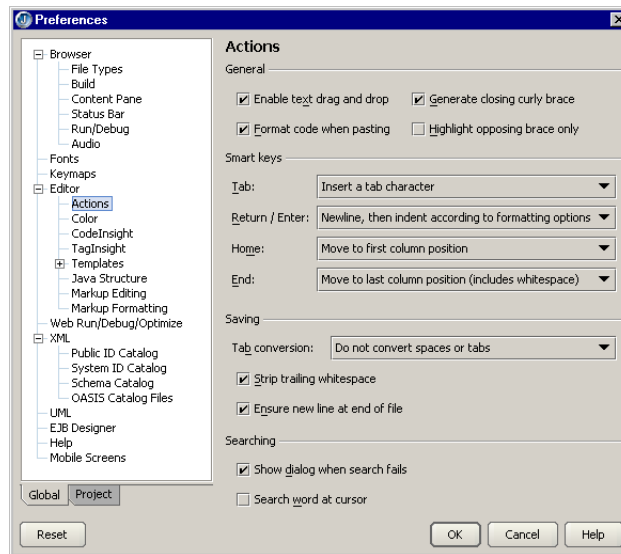
Customizing editor actions

The editor provides numerous settings to control its behavior. You can turn on or off actions and settings for “Smart” key behavior, searching, dragging and dropping, pasting, and more. Customize the editor's behavior to function efficiently with your work routine.

To customize editor actions,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Expand the Editor node and choose the Actions page.
- 3 Click the check boxes to choose actions from the General section.
- 4 In the Smart Keys section, choose from the Tab, Return/Enter, Home, and End key options from the drop-down menus.
- 5 Choose the type of tab conversion from the drop-down menu in the Savings section.
- 6 Click the check boxes to choose actions from the Saving and Searching sections.

7 Click the OK button to activate the settings and close the dialog box.



Splitting the source view

The editor lets you split the source view of a file into two or more vertical or horizontal panes. This setting applies to an individual file, not to all the opened files in the content pane. You can have a different configuration for every opened file.

To split the view,

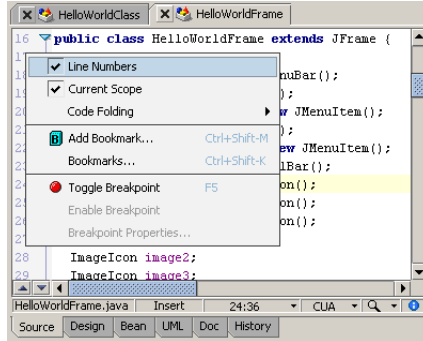
- 1 Click the Source tab with a file opened in the editor.
- 2 Right-click in the source pane.
- 3 Choose View.
- 4 Choose Split Vertically to split the editor into vertical panes.
- 5 Choose Split Horizontally to split the editor into horizontal panes.

To close the editor's multiple split pane views,

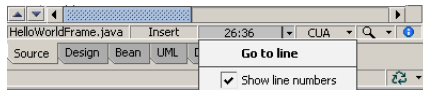
- 1 Right-click in a section of the split pane.
- 2 Choose View.
- 3 Choose Close View to close the section with the focus.
- 4 Choose Close Other Views to close the sections that don't have the focus.

Displaying line numbers

The editor includes an option to display line numbers in the left gutter margin. This feature can be turned on and off with the gutter setting, Line Numbers, on the Editor page of the Preferences dialog box (Tools|Preferences), and from a shortcut on the status bar. Line number display is turned on by default.



You can quickly change the line number display by accessing the editor status bar at the lower right corner of the editor. The line and column number for the location of the cursor are always displayed on the status bar. To the right of the line and column number is a drop-down menu for line commands.



To hide line numbers,

- 1 Click the arrow adjacent to the line and column number on the status bar. The drop-down menu opens.
- 2 Uncheck the Show Line Numbers check box.

To display line numbers,

- 1 Click the arrow adjacent to the line and column number on the status bar.
- 2 Check the Show Line Numbers check box.

Displaying white space characters

White space characters are invisible in the editor by default. However, if you want to display white space characters in the editor while working on your files, the editor provides a simple way to make the white space characters visible and to dim the color of the white space characters for easier viewing.

When you want to toggle the white space character display on and off in the editor, press *Ctrl+Shift+X* or right-click in the editor and choose the Toggle Visible White space context menu command.

To display white space characters,

- 1 Open a file in the editor.
- 2 Choose Tools|Preferences to open the Preferences dialog box.
- 3 Choose the Editor page.
- 4 Check Show Visible White space Characters in the Editor page.
- 5 Move the slider adjacent to the Dim White space Colors option if you want to dim the white space character color.
- 6 Click OK.

The white space characters are now visible in the editor.

Positioning file tabs

The file tabs of open files appear at the top of the editor by default. However, you can move the file tab positions to modify your editing space to fit your work style.

To reposition the content pane file tabs,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Expand the Browser node.
- 3 Choose Content Pane to open the Content Pane page.
- 4 Click the arrow adjacent to the Orientation menu under the Tabs section.
- 5 Choose from the Horizontal, Vertical on the Right, and Vertical on the Left menu options to reposition the file tabs.

Note You can also position the tabs in multiple rows or a single row with scroll control, as well as set the appearance, maximum width, inserting, and sorting order of the tabs.

Limiting open files

When working on large projects, you can choose to limit the number of unmodified open files to control your working environment. Once you set the maximum number, the editor automatically closes the oldest (open for the longest time), unmodified file. However, the editor does allow for an unlimited number of modified files to be open at the same time. And the editor won't close the open file currently in view.

To limit the number of open, unmodified files,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose Browser|Content Pane to open the Content Pane page.
- 3 Check the Automatic Close Of Least Recently Viewed check box.
- 4 Enter a number in the Maximum Open Per Project field to set the amount of unmodified files you want open at a time.
- 5 Click OK to close the dialog box and save the new setting.

Customizing unused variables

A simple way to customize and find unused Java code variables is to set a highlighting color in the Preferences dialog box. When the code variable changes from unused to used as you work in your source code, the highlighting automatically disappears.

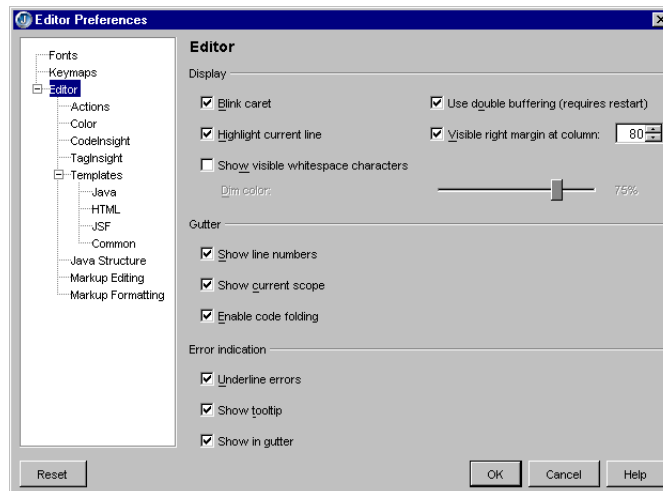
To set a color for unused code variables,

- 1 Choose Tools|Preferences to open the Preferences dialog box. (You can also right-click in an open Java file and choose Editor Preferences.)
- 2 Choose Editor|Color to open the Color page.
- 3 Click the Java tab.
- 4 Choose Unused Variables in the Screen Elements list.
- 5 Choose a color for the unused variable element.
- 6 Click OK.

Setting Editor preferences

Use the Editor preferences to modify the editing environment in JBuilder. The options on these pages specify font, keymapping, indent, search, error, display, color, CodeInsight, templates and Java structure settings.

To access the Editor preferences, choose Tools|Preferences|Editor from the main menu, or right-click in the content pane and choosing Editor Preferences.



The options on these pages vary by JBuilder edition and are described below. If an edition is not specified, the tip is available for all editions. The edition is specified by the following:

- F: JBuilder Foundation
- D: JBuilder Developer
- E: JBuilder Enterprise

The Editor Preferences include the following pages:

Table 17.1 Editor Preferences

Page	Description
Fonts	Select display and font options for the editor, browser, and output. (The Fonts page can also be accessed by right-clicking in the editor and choosing Editor Preferences Fonts.)
Keymaps	Choose a keymapping to use with JBuilder, or customize or copy a selected keymapping. (The Keymaps page can also be accessed by right-clicking in the editor and choosing Editor Preferences Keymaps.)
Editor	Customize the editor appearance. (The Editor pages can also be accessed by right-clicking in the editor and choosing Editor Preferences Editor.)
Actions	Customize editing behavior.
Color	Specify how the different elements of your code appear in the editor. You can set the color elements specifically for Java, HTML, XML, JSP and other types of screen elements.
CodeInsight	Configure CodeInsight for such things as enabling and setting the delay duration of automatic pop-ups, and setting MemberInsight and ParameterInsight options.
TagInsight	Configure TagInsight for such things as enabling and setting the delay duration of automatic pop-ups, and setting advanced autocomplete options. (D E)

Table 17.1 Editor Preferences (continued)

Page	Description
Templates	Add or remove template file types and import or export code templates. Expand the Templates icon to access the Java, HTML, and Common (includes any template you added), template pages.
Java	Create and edit templates for individual Java code elements you frequently use.
HTML	Create and edit templates for individual HTML tagging elements you frequently use.
JSF	Create and edit templates for JSF tagging elements you frequently use.
Common	Create and edit templates for common code elements you frequently use.
Java	Control the organization of your files in the Structure pane.
Markup Editing	Set editing options for markup language text files, for example XML or HTML.
Markup Formatting	Set formatting instructions for markup language text files, for example XML or HTML.

Part IV

Customizing JBuilder

Chapter 18

Introduction

This section of *Getting Started with JBuilder* describes how to setup JBuilder to best suit your preferences and work requirements. You'll learn how to configure your workspace, use the Preferences dialog box to modify JBuilder's appearance and behavior, use the Default Project Properties dialog box to specify the settings to apply when you create new projects, and use the OpenTools API to modify or extend features and functions.

This section contains the following chapters:

- [Chapter 19, “Configuring your workspace”](#)
Describes how you can rearrange the panes and tabbed pages of your workspace in a way that works best for you. You can save specific workspace configurations that you find useful for different kinds of work.
- [Chapter 20, “Setting JBuilder preferences”](#)
Introduces the Preferences dialog box (Tools|Preferences), which you can use to customize JBuilder's browser, key bindings, and editor, add file types for JBuilder to recognize, set options for running and debugging, configure audio options, and configure web, XML, UML, and EJB designer options.
- [Chapter 21, “Specifying default project settings”](#)
You can specify default project properties, such as source paths, JDK version to compile against, formatting, required libraries, and code auditing. The default project properties can be applied whenever you create a new project.
- [Chapter 22, “Extending JBuilder with OpenTools”](#)
JBuilder's OpenTools API provides a means for you to modify or extend the features and function of JBuilder to best suit your needs.

See also

- [Chapter 17, “Customizing the editor”](#) for information about modifying the editing environment in JBuilder

Configuring your workspace

Configuring your workspace means arranging the panes and tabs of the IDE in a way that works best for you. You can save specific workspace configurations which you find useful for different kinds of work.


JBuilder supports abundant workspace operations and configurations: maximizing and restoring panes, revealing and hiding panes, iconifying, docking, and undocking panes, positioning tabs, splitting the content pane between multiple files, and tearing file tabs off to view multiple files simultaneously. JBuilder also supports managing, editing, and using “classic” workspace configurations.

Move each pane using its individual title bar. The pane title bars support the following:

- Dragging and double-clicking.
- Host icons which provide specific move actions.
- Context (right-click) menu for move actions (also available from View/Window Actions).

Context menu commands are context-sensitive, and vary according to the pane’s current state and position.

Commands that control workspace configurations are available from one of two places:

- Choose View/Workspaces and choose a command from the submenu.
-  Click the down arrow next to the Workspaces icon in the toolbar and choose a command from the drop-down menu.

Both of these menus support the same commands. Because both menus let you do the exact same things, they are referred to, generically, as the *Workspaces menu*.

The “default” menu item in the Workspaces menu is the workspace configuration you see when you start JBuilder for the very first time. Choose Default from the Workspaces menu and then make a file active to return to JBuilder’s original layout at any time. Otherwise, name a useful configuration before you change it. Then it’s available from the Workspaces menu and you can switch to it at any time.

When you close JBuilder without saving the current workspace configuration, JBuilder automatically saves it as Workspace1. That new workspace is then active next time you start JBuilder. Next time you close JBuilder without saving a changed configuration, that new configuration becomes Workspace1.

Note Workspace configuration is different from personality configuration, a feature of JBuilder Enterprise. Personality configuration determines which technologies are visible in the IDE. For more information on personality configuration, choose Project | Project Properties | Personality and click the Help button.

Using JBuilder's "classic" configuration

If you don't want the panes to be moved at all and don't want to lose real estate to title bars in the panes, you can use the "classic" JBuilder configuration:

- 1 Choose Tools | Preferences to open the Preferences dialog box.
- 2 Choose the Browser page.
- 3 Uncheck Pane Locations Can Be Customized.
- 4 Click OK.

The panes return to their original positions and the title bars disappear from the panes. You can switch back to configurable workspaces by checking Pane Locations Can Be Customized.

Managing configurations

Workspaces can be renamed, removed, moved up and down in the Workspaces menu, and also reset to the original configuration. The Manage Workspace dialog box lets you rename or delete a configuration, and determines the order in which named configurations are listed in the Workspaces menu.

Once you've named your workspace configurations, pick from one of the named configurations either by choosing a configuration directly from the Workspaces menu on the main toolbar or by choosing Select Workspace from the Workspaces menu and selecting a named workspace configuration from the list.

To manage your workspace configuration,

- 1 Choose View | Workspaces | Manage Workspaces or choose Manage Workspaces directly from the Workspaces menu on the main toolbar.

The Manage Workspaces dialog box opens.

- 2 Select the workspace you want to configure.
- 3 Choose from the Rename, Remove, Move Up, or Move Down buttons to configure your workspace.

Choose Reset Workspace from the Workspaces menu (either from the toolbar or from the View main menu), to return to the previous workspace. Reset Workspace is only an option if you haven't saved the present workspace and also does not change the configuration of torn-off files.

Editing configurations

When you have a configuration you like, you can create a new workspace configuration for it.

- 1 Choose **Save Workspace As** from the **Workspaces** menu (or choose **View | Workspaces | Save Workspace As.**)

The **Save Workspace As** dialog box appears.

- 2 To create a new named configuration, type in a new name.
- 3 To overwrite an existing configuration, select the name of that configuration from the drop-down list.
- 4 Click **OK**.

The new configuration's name appears in the **Workspaces** menu.

Maximizing and restoring panes

Maximizing a pane makes the pane fill all the space of the IDE window. Restoring it reduces it to the size and position it had before being maximized. This is particularly useful, for instance, when checking message pane output, viewing a stack dump or other output, then returning quickly to the editor.

There are several ways to access the **Maximize** and **Restore** commands:

- Double-click the pane's title bar to maximize it. Double-click it again to restore it.
- Right-click on the pane's title bar and choose **Maximize** or **Restore** from the context menu.
- Left-click on the **Maximize** or **Restore** icon on the right side of the title bar.
- Choose **View | Window Actions | Maximize** or **Restore**.
- Press **Alt+F10** on the keyboard.

Revealing and hiding panes

The work environment provides multiple panes, menus, and toolbar buttons for viewing and working with your files and projects. The project, files, structure, and content panes, as well as the main menu and toolbar appear by default when you first open JBuilder. Use the **View** menu to display panes that aren't displayed by default (classes and message panes), or to hide those that are displayed. You also can choose to hide or reveal toolbar buttons.

To display or hide panes,

- 1 Choose the **View** menu.
- 2 Choose **Panes**, then choose from the list of pane choices to display or hide panes.

The **Panes** submenu commands provide keyboard shortcuts to reveal or hide some of the panes.

To display or hide toolbars,

- 1 Choose the **View** menu and
- 2 Choose **Toolbars**, then choose from the list of toolbar choices to display or hide toolbars.

Iconifying, docking, and undocking panes

These commands allow you to control the location and visibility of the project, structure, and message panes:

- When you iconify a pane, the pane hides but remains available from the pane's icon in the left border of the IDE window. Clicking the pane's icon opens the pane in an undocked position, floating on the main IDE window. The pane slides out automatically when you pause the mouse over the pane's icon. Iconify a pane by clicking the Iconify icon.
- When you open the pane in an undocked, iconified position, single-click its icon or right-click its icon and choose Dock All to return the pane to its original state and size.
- When you undock a pane, the pane separates from the main IDE window and can be moved across your display, independent of the rest of the IDE. Undock a pane by clicking and dragging the title bar or the tab (if the pane is in a tabbed view). Or, with the focus in the pane, undock a pane by choosing View|Window Actions, then choosing Undock in the pop-up menu.
- When you dock a pane, the pane reattaches to the main IDE window. Dock a pane either by clicking the Dock icon, dragging it to a docking location while holding down the *Ctrl* key and releasing it, or right-clicking the pane's title bar and choosing Dock.

Manipulation commands for the pane that has focus are also available by right-clicking its title bar or from View|Window Actions (*Alt+F10* on the keyboard).

Iconifying and undocking

When a pane is iconified, click the icon once to open and undock the pane. Click the icon again (without undocking the pane) to slide the pane back into the iconified position. Alternatively, press *Alt+1* to control the first pane iconified, *Alt+2* to control the second, and so forth. You also can resize an iconified pane when the pane is in its slide-out position.

You can position undocked workspace panes behind the work environment, rather than on top. When you choose this option, the panes are also displayed as separate windows on the desktop taskbar. The panes automatically maximize and minimize when you maximize and minimize JBuilder.

To position undocked workspace panes behind the work environment,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose the Browser page.
- 3 Uncheck the Undocked Workspace Panes Always On Top check box.

Docking

When you dock panes, think of the main IDE window as having four cardinal directions and two ordinal directions: North, South, East, and West; and at the left edge, Northwest and Southwest. The project pane defaults to Northwest, the structure pane defaults to Southwest, and the message pane defaults to South.

Panes can be stacked vertically or horizontally or both, depending on the border being docked on. To control docking a pane, press down the *Ctrl* key and drag the pane to where you want it to dock. Dark gray reference indicators appear, showing you where the pane will dock if you let go at a given moment. Release the mouse button when the reference indicators frame the desired location.

Positioning tabs

When you move the panes around in your work environment, you might also want to move the content pane file tab positions. If you have moved the project and structure panes to the right of the content pane (rather than leaving them at their default position on the left), moving the content pane tabs to the left of the content pane makes the work environment less crowded.

To reposition the content pane file tabs,

- 1 Choose Tools|Preferences to open the Preferences dialog box.
- 2 Choose Browser.
- 3 Choose Content Pane to open the content pane page.
- 4 Click the arrow adjacent to the Orientation menu under the Tabs section.
- 5 Choose from the Horizontal, Vertical on the Right, and Vertical on the Left menu options to reposition the file tabs.

You can position the tabs in multiple rows or a single row with scroll control, as well as set the appearance, maximum width (of the currently open file), insertion placement, and sorting order of the tabs. You also can reorder the open file tabs using drag and drop. However, you can only use drag and drop if you haven't chosen one of the Insertions sort settings on the Content pane page of the Preferences dialog box.

Viewing multiple files

You can view several files at once, using only one instance of JBuilder. There is only one content pane per project, but you can open several file viewers at once. You can either split the content pane between multiple files, or move a file into its own *viewing pane*. A viewing pane is a new window for displaying undocked files, with JBuilder's main menu and toolbar and all the normal file view tabs in it.

Click and drag a file's tab either to split the content pane between multiple files or to undock that file in its own viewing pane:

- The file tab's pane appears as a light gray rectangle when your pointer is in a location that splits the content pane. Release the mouse to see the file in its new split-pane view.
- The reference indicators disappear when the pointer is in a location that undocks the file.
- Alternatively, undock a file by right-clicking the file's tab and choosing Undock.

The viewing pane is a separate window, so it has the same controls in the title bar as any conventional window hosted on your operating system. Workspace-specific commands are available from the context (right-click) menu of the file's tab.

Once a file is undocked in a viewing pane, other files can share that viewing pane as tabs, but only one file at a time can be displayed inside it. Undocked viewing panes cannot display a split view of multiple files within the same undocked pane.

Redocking files

Redock a file in either of these ways:

- Grab the file tab inside the viewer and drag it back into the main content pane. Drop it when dark grey reference indicators appear around the content pane.
- Right-click the file tab and choose Dock.

Dragging the title bar of the viewer moves the whole viewer, but does not redock the file.

Clicking and dragging a file's tab to pull it away from the main window of the IDE is referred to as *tearing off* the tab.

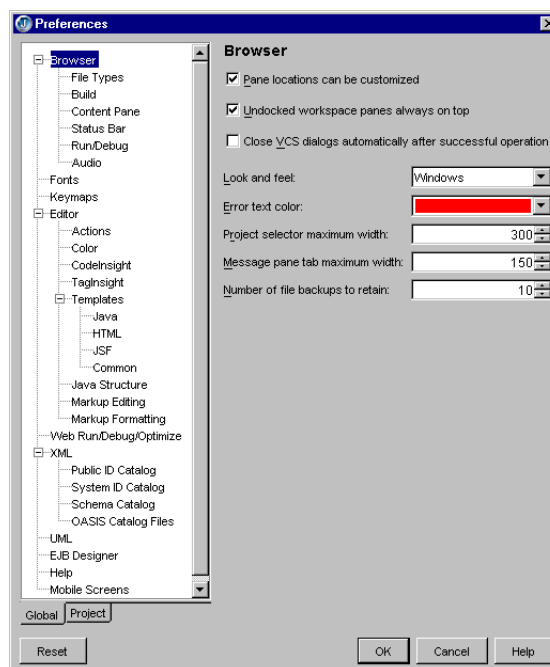
Note Viewing pane configurations (splitting the content pane or tearing off a file tab) are not saved as part of the workspace configuration, but as part of the project. The viewing pane configuration remains the same when you switch workspace configurations.

Chapter 20

Setting JBuilder preferences

Use the Preferences dialog box to customize the JBuilder IDE (integrated development environment). However, if you have previous JBuilder versions installed with prior customizations, you can retain those settings. See [“Importing settings” on page 176](#) for more information.

To open the Preferences dialog box, choose Tools|Preferences from the main menu.



The Preferences dialog box contains two tabbed pages for customizing JBuilder's IDE: Global and Project.

- The Global page has settings that persist in the IDE for all projects.
- The Project page has settings specific to the current project, and duplicates the Project Properties dialog box (Project|Project Properties.)

The options on these pages vary by JBuilder edition and are described below. If an edition is not specified, the tip is available for all editions. The edition is specified by the following:

F: JBuilder Foundation

D: JBuilder Developer

E: JBuilder Enterprise

Table 20.1 Global

Page	Description
Browser	Configure and customize JBuilder's IDE.
File	Customize the types of files the JBuilder editor recognizes.
Build	Customize the behavior of the build system.
Content Pane	Specify how the file tabs are organized and displayed.
Status Bar	Set the duration of the status bar message and customize heap reporting.
Run/Debug	Customize the runtime and debug update intervals.
Audio	Controls JBuilder's audio feedback for various events and is enabled by default. JBuilder currently provides an audio theme stored in the JBuilder <code>lib/audio/</code> directory. You can also add custom audio files and attach them to specified events.
Fonts	Select display and font options for the editor, browser, and output. (The Fonts page can also be accessed by right-clicking in the editor and choosing Editor Preferences Fonts.)
Keymaps	Choose a keymapping to use with JBuilder, or customize or copy a selected keymapping. (The Keymaps page can also be accessed by right-clicking in the editor and choosing Editor Preferences Keymaps.)
Editor	Customize the editor appearance. (The Editor pages can also be accessed by right-clicking in the editor and choosing Editor Preferences Editor.)
Actions	Customize editing behavior.
Color	Specify how the different elements of your code appear in the editor. You can set the color elements specifically for Java, HTML, XML, JSP and other types of screen elements.
CodeInsight	Configure CodeInsight for such things as enabling and setting the delay duration of automatic pop-ups, and setting MemberInsight and ParameterInsight options.
TagInsight	Configure TagInsight for such things as enabling and setting the delay duration of automatic pop-ups, and setting advanced autocomplete options. (D E)
Templates	Add or remove template file types and to import or export code templates. Expand the Templates icon to access the Java, HTML, JSF, and Common (includes any template you added), template pages.
Java	Create and edit templates for individual Java code elements you frequently use.
HTML	Create and edit templates for individual HTML tagging elements you frequently use.

Table 20.1 Global (continued)

Page	Description
JSF	Create and edit templates for JSF tagging elements you frequently use. (D E)
Common	Create and edit templates for common code elements you frequently use.
Java	Control the organization of your Java files in the structure pane.
Markup Editing	Set editing options for markup language text files, for example XML or HTML. (D E)
Markup Formatting	Set formatting instructions for markup language text files, for example XML or HTML. (D E)
Web Run/Debug/Optimize	Configure how JBuilder runs, debugs, and optimizes web applications. (D E)
XML	Set general and trace XML options globally for the JBuilder IDE. JBuilder also supplies the means to add XML catalogs. The XML catalogs provide remapping information for online schema location and online DTD location to downloaded local files. The catalog mappings allow you to edit and validate XML documents offline. (D E)
Public	Remap listed public IDs to local files. (D E)
System	Remap system IDs and schema locations to local files. (D E)
Schema	Remap schema namespace URIs (Uniform Resource Identifiers) to local files. (D E)
OASIS	Add XML catalog files that conform to the OASIS XML Catalogs Specification and/or text files that contain a sequence of entries. (D E)
UML	Globally configure the display of UML diagrams in JBuilder's UML browser. You can control such options as visibility icons, grouping of elements, display of properties, font, font size, and color of text, arrows, and backgrounds. (E)
EJB	Globally configure the display of the EJB Designer. You can control such options as font, font size, and color of text, borders, and backgrounds. (E)
Help	Specify Help Viewer and Dynamic Help options.
Mobile	Control the appearance of MIDP Displayable screens. (D E)

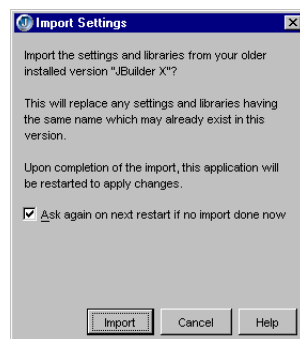
Note All the options on the Project page are duplicated in the Project Properties dialog box which can be accessed by choosing Project|Project Properties.

Page	Description
Paths	Choose the JDK version to compile against, the output path, the backup paths, the working directory, the source path, the documentation path, and the libraries to use when compiling.
General	Specify some basic project properties, such as the encoding that controls how JBuilder interprets text files containing characters beyond the ASCII character set, enabling source Package discovery and compilation, specifying the Javadoc tags and text to use in generated class files, including references from all of the project libraries (D E), and generating source references, such as EJB stubs, in the UML diagram (E).
Find	Filter class lists so that the classes you select don't appear when the class lists are invoked. This means the classes displayed are more likely to be pertinent and the display less cluttered.
Unit	Specify packages and classes to exclude from stack traces when running unit tests using <code>JBTestRunner</code> .
UML	Specify any packages or classes to excluded from the UML diagram. However, they still appear in the structure pane. (E)
Run	Create and modify runtime configurations. Runtime configurations are preset runtime parameters, including application and VM parameters, server settings, build targets, and other settings based on what is being run. (D E)

Page	Description
Build	Set build options for the project.
Java	Choose the Java compiler, debug options, language features, target VM, and other compiler options for Java compilation.
IDL	Choose the IDL compiler, generated code options, and conditional symbols for IDL compilation. (E)
Resource	Set project-wide resources by file extension. (D E)
Javadoc	Create, edit and remove custom Javadoc tags. (D E)
Ant	Add Ant libraries that are required by custom Ant targets in your project.
Web	Enable or disable automatic generation of a web services deployment file when you build the project. (E)
Menu	Specify which build targets to display on the Project menu and on the toolbar.
Basic	Specify general indentation, tab size, and end of line characters.
Java	Choose a specific Java formatting style, and specify whether to use the basic formatting settings or specify different ones for Java files.
Blocks	Control different aspects of formatting for various types of code blocks.
Spaces	Specify where to insert space in your code to make the elements easier to see.
Blank	Control where blank lines are inserted in your code.
Wrapping	Choose how code wrapping is implemented.
Generated	Specify how code is generated for event handlers, variable visibility, and JavaBeans instantiation.
Imports	Control how packages and files are imported into and displayed in the import statements section. (D E)
Decorations	Control what and where to display visual indicators for file and VCS states.
Server	Select one or more servers to provide the services you need for your current project. (D E)
Personality	Specify which groups of JBuilder features are visible in the active project. Features are grouped by technology. These technology groups are referred to as personalities.
File	Displays information on the current project, such as file name, type, location, size, and date last modified.
Code	Lets you enable or disable code auditing. When code auditing is turned on, JBuilder checks for errors, inconsistencies, performance inefficiencies for the active file, and design flaws in your code and documentation. (D E)

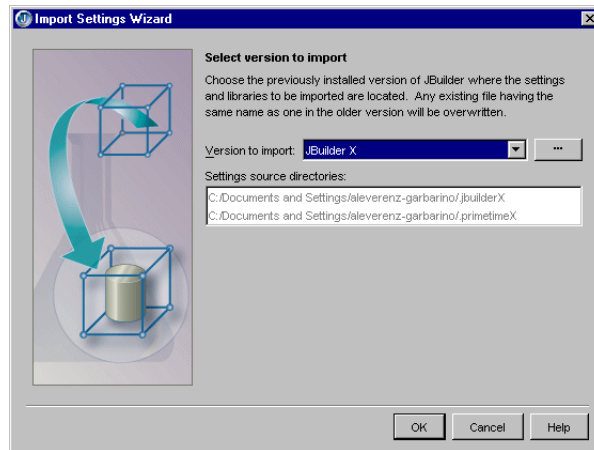
Importing settings

When you install JBuilder, you have the option to import user-specific settings and libraries from older installed versions of JBuilder. Some of the settings you can import include keymaps, color schemes, file types, workspaces, fonts, histories, and code templates. The Import Settings dialog box only appears during start-up when you have had previous versions of JBuilder installed.



If you decide not to import the older settings during start-up, you can choose to invoke the Import Settings dialog box the next time you open JBuilder. When you do choose to import previous settings, the import settings action replaces the newer settings and libraries of the same name with the older settings.

If you didn't import the settings during start-up, you also can use the Import Settings Wizard while you're working in JBuilder to retain previous settings and libraries. Choose Tools|Import Settings from the main menu to invoke the Import Settings wizard.



For more information, click the Help button in either the Import Settings dialog box or the Import Settings wizard.

Specifying default project settings

JBuilder's project settings determine, among other things, how a project is compiled and run. You can specify project path settings, set up a run configuration for your project, specify how a project is built, define filters, specify server options, and much more. If you have standard settings for paths, libraries, database drivers, runtime configurations, build properties, or code formatting, you can modify the default project properties (Project!Default Project Properties) to incorporate these required settings in any project you create.

Default project settings are stored in an actual project called `Default.jpr` found in the `.jbuilder` subdirectory of your home directory. The `Default.jpr` file can be used as a template whenever a new project is created by selecting "(Default project)" from the Template drop-down list box in the first step of the Project wizard.

All of the project properties specified in the Default Project Properties dialog box are applied to the new project. Project paths and general project properties can be changed in step 2 of the Project wizard, and all project settings can be changed at any time using the Project Properties dialog box (Project!Project Properties).

Setting paths

The Paths page of the Default Project Properties dialog box lets you set project path settings for the JDK version, output path, backup path, working directory, source paths, test path, documentation paths, and required libraries paths.

You may want to modify the default settings on the Paths page for the following reasons:

- You use a specific version of JDK, different from the default JDK. For example, if you are developing mobile applications, you will want to change the JDK to one that supports the mobile profile for which you are developing, such as a J2ME MIDP/CLDC JDK for the MIDP profile.
- You prefer to store your work on a different drive or directory instead of your home directory.
- You have standard items, such as libraries or database drivers, that you normally want to add to the class path.

Click the Help button in the Default Project Properties dialog box for information on the individual items on the Paths page.

See also

- “Creating and managing projects” in *Building Applications with JBuilder*
- “Managing paths” in *Building Applications with JBuilder*

Setting general properties

The General page of the Default Project Properties dialog box lets you set options for encoding, enabling automatic source packages, modifying class Javadoc fields that wizards can generate, including references from project libraries, and diagramming references from generated source.

You may want to modify the default settings on the General page for the following reasons:

- You want to specify a specific character encoding for your projects. You can specify an encoding to control how the compiler interprets characters beyond the English (ASCII) character set. By default, JBuilder automatically sets character encoding to the native encoding for your operating system’s locale. If you are developing for specific locales, you may want to set the encoding to a specific value.

Note

Encoding can also be set at the command line when you use Borland Compiler for Java (**bcj**) and Borland Make for Java (**bmj**). Borland Compiler for Java and Borland Make for Java are features of JBuilder Developer and Enterprise.

- You have standard Javadoc tags and text that you want to appear in your class files. Specify the labels and text in the supplied fields, by selecting a field and typing. When you check the Generate Header Comments check box in wizards that generate Java classes, such as the Application wizard and Class wizard, the tags and text are included at the top of the generated class files.

Click the Help button in the Default Project Properties dialog box for information on the individual items on the General page.

See also

- “Internationalizing programs with JBuilder” in *Building Applications with JBuilder*
- “Automatic source packages” in *Building Applications with JBuilder*
- “Filtering packages” in *Building Applications with JBuilder*

Specifying runtime configurations

Support for multiple runtime configurations is a feature of JBuilder Developer and Enterprise

The Run page of the Default Project Properties dialog box lets you select or create a configuration to use for running or debugging.

You may want to modify the default settings on the Run page for the following reasons:

- You have standard runtime configurations for testing and debugging.
- You routinely use individual runtime configurations for client and server applications in your projects.
- You have standard runtime configurations based on the type of emulator you are using. If you are developing mobile applications, you may want to define a runtime configuration that specifies the default device to use by the emulator, or you may want to define a runtime configuration for each device supported by a given emulator. Mobile Development is a feature of JBuilder Developer, Enterprise, and Mobile Edition.

Click the Help button on the Run page in the Default Project Properties dialog box for information on the individual items.

See also

- “Running programs in JBuilder” in *Building Applications with JBuilder*
- “Setting runtime configurations” in *Building Applications with JBuilder*
- “Debugging Java programs” in *Building Applications with JBuilder*
- “Unit testing” in *Building Applications with JBuilder*

Specifying build properties

Build features vary by JBuilder edition

The Build pages of the Default Project Properties dialog box let you set options for building a project.

You may want to modify the default settings on the Build page for the following reasons:

- To change the compiler, set additional compiler options, or customize debugging.
- To have custom Javadoc tags that you implement in all your projects. This is a feature of JBuilder Developer and Enterprise.
- To specify resource types to copy to the output path when you build your project.
- To customize how your project uses Ant.
- To change deployment regeneration for Web Services.
- To add build targets to the Project menu or customize build behavior.
- To specify which version of the Java JDK to use when building.

Click the Help button on the General page in the Default Project Properties dialog box for information on the individual items.

See also

- “Building Java programs” in *Building Applications with JBuilder*
- “Setting build preferences” in *Building Applications with JBuilder*

Setting code format properties

Formatting options vary by JBuilder edition

The Basic and Java Formatting pages of the Default Project Properties dialog box let you set code formatting options. JBuilder can speed your coding by formatting it automatically to your specifications.

You may want to modify the default settings on these pages if you use a standard format for your code and want to apply it all new projects you create.

Click the Help button on the Basic Formatting page or the Java Formatting page in the Default Project Properties dialog box for information on the individual items.

See also

- [“Formatting code” on page 113](#) for descriptions of code formatting options

Specifying file status decorations

The Decorations page of the Default Project Properties dialog box provides icon and text decorations for files, packages, and projects. The icons and text decorate: modified files, read-only files, files not in the active project, VCS workspace and repository files, and files not in the repository. Enable the decorator icons so you can quickly and visually identify the various file states.

Click the Help button on the Decorations page in the Default Project Properties dialog box for details about decoration options.

See also

- [“Using file status decorations” on page 70](#) for information about how to set file decoration options.

Setting server and service properties

**This is a feature of
JBuilder Developer
and Enterprise**

The Server page of the Default Project Properties dialog box lets you select one or more servers to provide the services you need for your projects.

You may want to modify the default settings on the Server page if you develop applications for a specific server or servers.

Click the Help button on the Server page in the Default Project Properties dialog box for information on the individual items.

See also

- “Configuring the target server settings” in *Developing Applications for J2EE Servers*

Configuring personality properties

The Personality page of the Default Project Properties dialog box determines which groups of features (personalities) appear in JBuilder. Configuring JBuilder’s personalities allows you to customize JBuilder so you see only the features you require.

You may want to modify the settings on the Personality page to hide groups of JBuilder features that do not apply to the development work you normally perform.

Click the Help button on the Personality page in the Default Project Properties dialog box for information on the individual items.

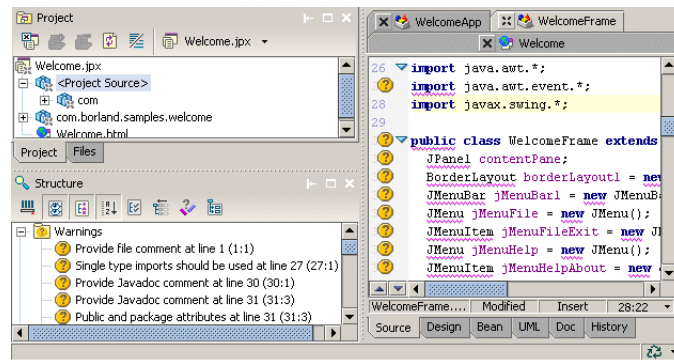
For more information about setting project properties, refer to “Creating and managing projects” and “Managing paths” in *Building Applications with JBuilder*.

Viewing file information

The File Information page displays basic information about the current project file, such as name, size, location, type, and date last modified. This is an informational page.

Configuring code audits

The Code Audits page lets you enable or disable code auditing. When code auditing is turned on, JBuilder checks the active, open file for errors, inconsistencies, efficient performance, and design flaws in your code and documentation. Check the box beside an item to audit it in the Code Audits page. As you write your code, JBuilder dynamically reports the auditing results as warnings in the structure pane.



See also

- [“Using code audits” on page 60](#)
- “Using audits” in *Building Applications with JBuilder*
- “Audit definitions” in *Building Applications with JBuilder*

Chapter 22

Extending JBuilder with OpenTools

**Access to OpenTools
features varies by
JBuilder edition**

JBuilder is open and extensible, and JBuilder's OpenTools API provides a means for you to modify or extend the features and function of JBuilder to best suit your needs. For example, using the OpenTools API you can write your own custom menu items, key mappings, and code generators, and integrate JBuilder with external programs such as version control systems.

Note When you register an OpenTool with a Wizards menu item, the registered OpenTool appears on the Wizards menu (Edit|Wizards).

See also

- *Developing OpenTools*
- "OpenTools Basics" in *Developing OpenTools*
- "Introduction to the OpenTools API" in the OpenTools Documentation in online help

Part V

Tutorials: Basic

Chapter 23

Introduction

The following basic tutorials help you learn how to use JBuilder as you build your own application and applet. The tutorials include information about how JBuilder works.

- [Chapter 24, “Tutorial: Building an application”](#) — Create a “Hello World” application.
- [Chapter 25, “Tutorial: Building an applet”](#) — Create an AWT applet.

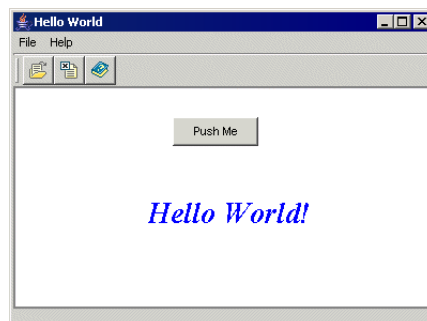
Chapter 24

Tutorial: Building an application

Many developers prefer to learn by doing. If this is you, follow the steps outlined in this tutorial to learn about using JBuilder's integrated development environment (IDE) to create applications. The tutorial shows how to

- Create a project for the application.
- Create a simple "Hello World" application.
- Build and run the application.
- Modify the user interface of an application.
- Add Swing components, such as JPanel, JLabel, and JButton.
- Edit the source code.
- Compile, build, and run the application.
- Run the application from the command line.
- Add an event method to a button.
- Complete the UI.
- Bundle the files for deployment.
- Run the deployed application from the JAR file.
- Run the deployed application from the command line.

When you have completed the tutorial, you will have a Java application that looks like this:



If you need specific information about the IDE or editor as you work through the tutorial, see the following chapters, [Chapter 8, “Using the JBuilder workspace”](#) and [Chapter 15, “Working in the editor.”](#)

For the complete source code, see [“HelloWorld source code” on page 211](#).

The Accessibility options section in the JBuilder Quick Tips contains tips on using JBuilder features to improve JBuilder’s ease of use for people with disabilities.

For information on documentation conventions used in this tutorial and other JBuilder documentation, see [“Documentation conventions” on page 31](#).

To suggest ways to improve this tutorial, send e-mail to jgpubs@borland.com.

Step 1: Creating the project

Before creating an application in JBuilder, you must first create a project to work in. JBuilder uses a project file with a `.jpx` extension to organize the application files and maintain the settings and project properties. The Project wizard can create a project for you.

- 1 Choose File|New Project to open the Project wizard.
- 2 Make the following changes to the appropriate fields in Step 1 of the Project wizard:
 - a Type `HelloWorld` in the Name field.

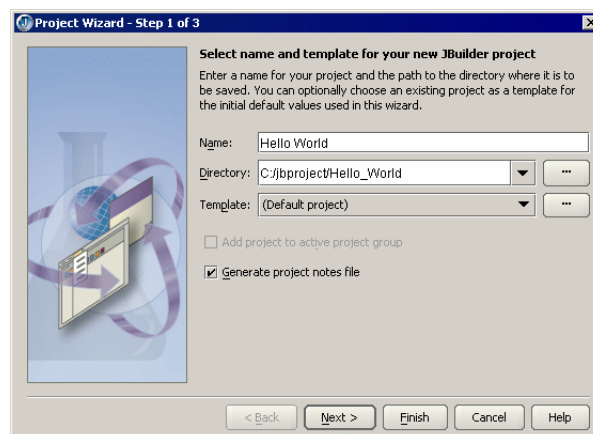
Note

As you type the project name in the Name field, the same name is entered in the Directory field. By default, JBuilder uses this project name to create the project’s directory name and the package name for the classes. The package name is forced to lowercase according to the Java convention for naming packages. Projects in JBuilder are saved by default in the `<home>/jbproject/` directory. The home directory varies by platform. See [Chapter 4, “Learning more about JBuilder.”](#) For more information on projects, see [Chapter 6, “Working with projects.”](#)

- b Accept `jpx` as the project file type.
- c Check the Generate Project Notes File option.

When you check this option, the Project wizard creates an HTML file for project notes and adds it to the project.

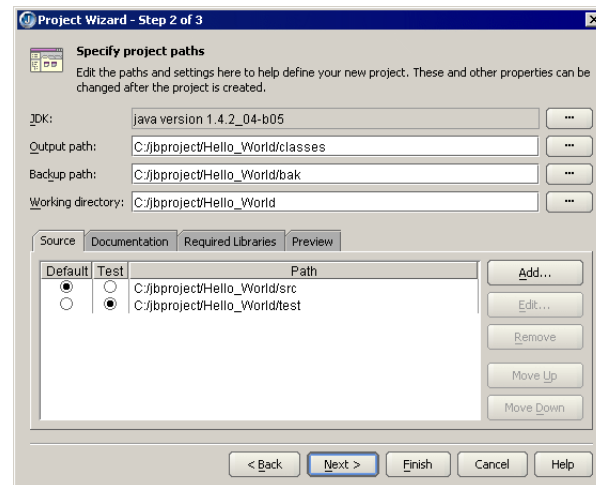
Step 1 of the Project wizard should look this, with the exception of the Directory path:



- 3 Accept all other defaults in Step 1.
- 4 Click Next to go to Step 2 of the Project wizard.
- 5 Accept the default paths in Step 2. Note where the compiled class files, project files, and source files will be saved.

For more information on paths and how JBuilder saves files, see “Managing paths” in *Building Applications with JBuilder*.

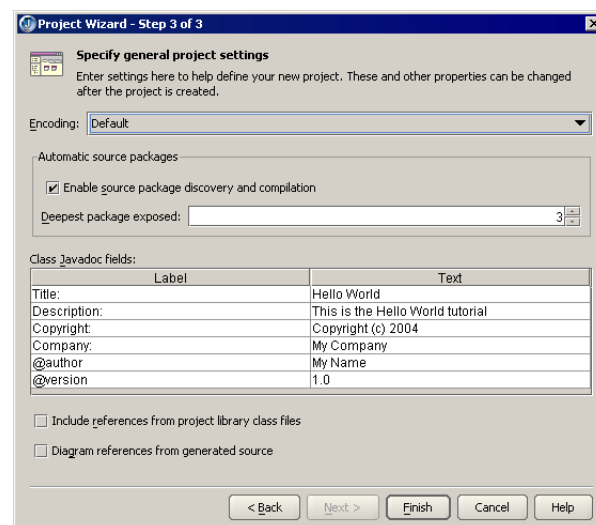
Step 2 of the Project wizard should look similar to this:



- 6 Click Next to go to Step 3 of the wizard.
- 7 Make the following changes in the appropriate fields of Step 3:
 - a Accept the Encoding and Automatic Source Packages defaults.
 - b Type `Hello World` in the Title field of the class Javadoc fields.
 - c Enter your name, company name, and a description of your application in the appropriate optional fields.

Note The information in the class Javadoc fields appears in the project HTML file and as optional header comments in the source code.

Step 3 of the Project wizard looks similar to this:



- 8 Click the Finish button.

Two files, `HelloWorld.jpx` and `HelloWorld.html`, are generated by the wizard and appear in the project pane located in the upper left of JBuilder's IDE.

- 9 Double-click `HelloWorld.html`, the project notes file, to view it in the content pane located in the center of the IDE. Note that it contains the project name, author, company, and description information you just entered in Step 3 of the Project wizard.

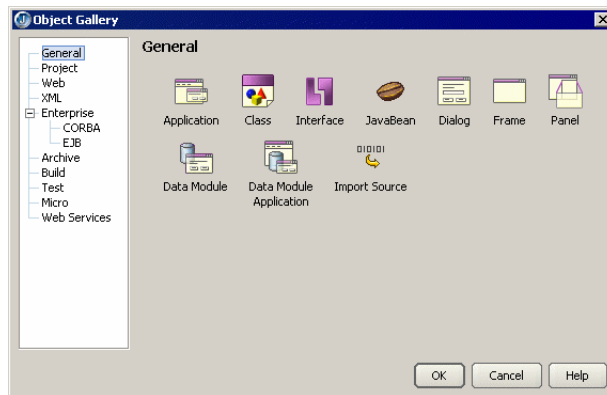
Step 2: Generating your source files

The Application wizard creates `.java` source files that are added to the project you just created.

To generate source files for your application using the Application wizard, follow these steps:



- 1 Choose File|New or click the New button on the main toolbar to open the object gallery.



- 2 Select General and double-click the Application icon to open the Application wizard.

- 3 Type `HelloWorldClass` in the Class Name field, in Step 1 of the Application wizard.

This is a case-sensitive Java class name.

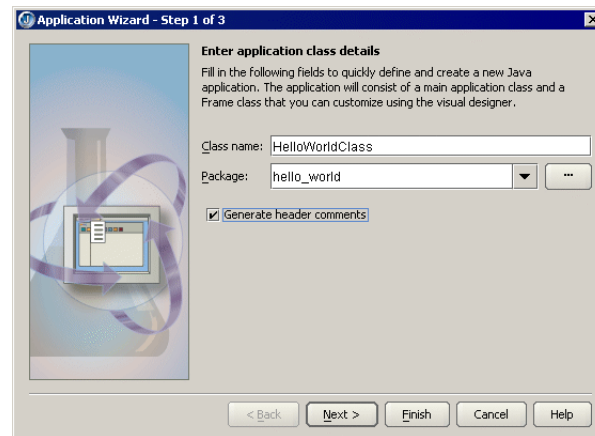
- 4 Accept the default package name, `helloworld`.

By default, the wizard takes the package name from the project file name, `HelloWorld.jpx`.

- 5 Check Generate Header Comments.

When you select this option, the project notes information you entered in Step 3 of the Project wizard appears at the beginning of each source file that the Application wizard generates.

Step 1 of the Application wizard should look like this:



6 Click the Next button to go to Step 2 of the Application wizard.

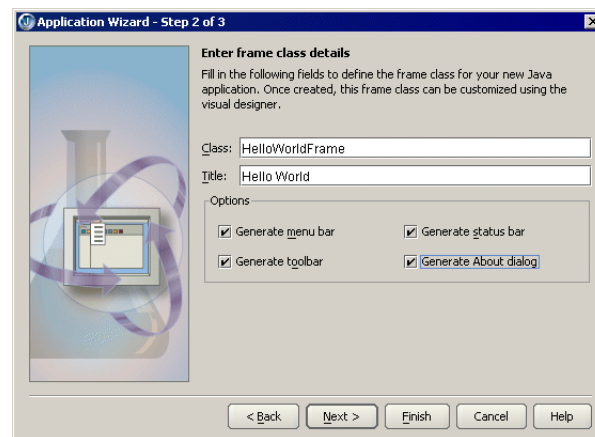
7 Type HelloWorldFrame in the Class field to name the Frame class.

8 Type Hello World in the Title field.

This text appears in the title bar of the frame in your application.

9 Check all of the options for additional application features: Generate Menu Bar, Generate Toolbar, Generate Status Bar, Generate About Dialog, and Center Frame On Screen. The wizard generates the basic code to support these features.

Step 2 of the Application wizard should look like this:



10 Click the Finish button.

The new .java source files and toolbar images are added to your project and displayed as nodes in the project pane. The source code for HelloWorldFrame.java is open in the content pane as shown in the following image.

The message pane appears only when a message is displayed. Therefore, you won't see the message pane yet if you are following these steps.

Note

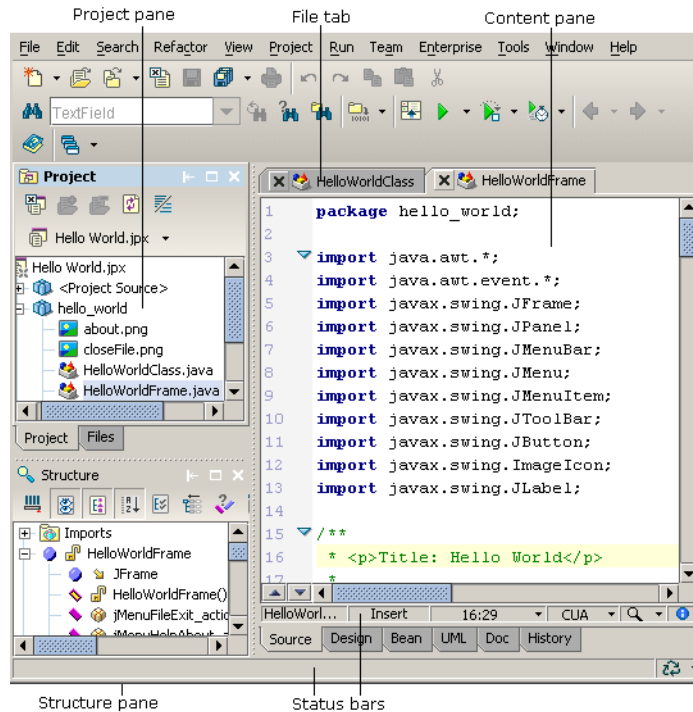
Step 3 (defining and creating a new runtime application configuration) is skipped because a new runtime configuration for your HelloWorld application was automatically created in that step of the wizard. If you want more information about runtime configurations, see "Setting runtime configurations" in *Building Applications with JBuilder*.

Note An automatic source package node named `helloworld` also appears in the project pane if the Enable Source Package Discovery And Compilation option is enabled on the General page of the Project Properties dialog box (Project!Project Properties).

JBuilder's IDE should appear similarly to the following image displaying the Hello World project, files, structures, and source code.

The following image demonstrates how JBuilder's IDE should appear with the Hello World project, files, structures, and source code displayed in the project, structure, and content panes. (You may see a slightly different view depending on your operating system and workspace settings.)

Figure 24.1 JBuilder's IDE elements



11 Choose File!Save All to save the source files and the project file.

Note The source files are saved to: `/<home>/jbproject/HelloWorld/src/helloworld`.

The class files generated from the source files by the Java compiler are saved to:

`/<home>/jbproject/HelloWorld/classes/helloworld`.

Step 3: Building and running your application

Now, build and run the application. Building includes build tasks such as preparing non-Java files for compiling, compiling Java source files, copying resources, archiving, deploying, and so on. Compiling is the process of running the Java compiler. The compiler, which translates source code into Java bytecode, generates `.class` files.

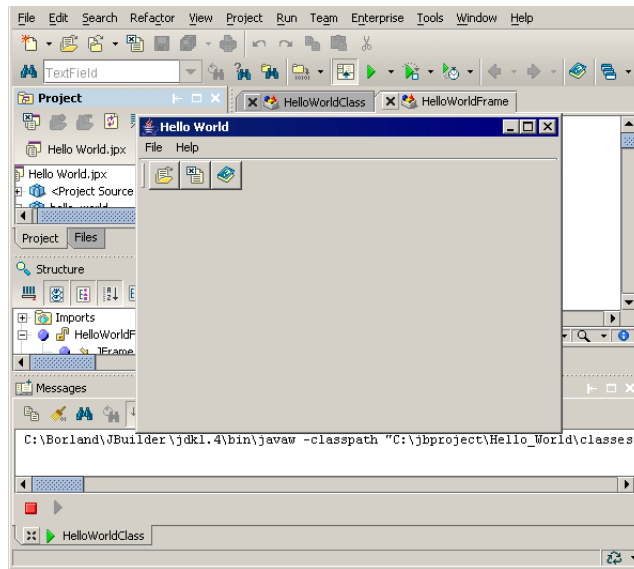


Tip

- 1 Choose Run|Run Project or click the Run Project button on the JBuilder toolbar to compile and run your application.

You can also select `HelloWorldClass.java` in the project pane, right-click, and choose Run using “HelloWorldClass”.

First, the message pane opens displaying the run process. Then, your application is displayed and should look like this:



- 2 Choose File|Exit in the “Hello World” application to close it.
- 3 Click the Close button on the HelloWorldClass tab in the message pane to close any messages.

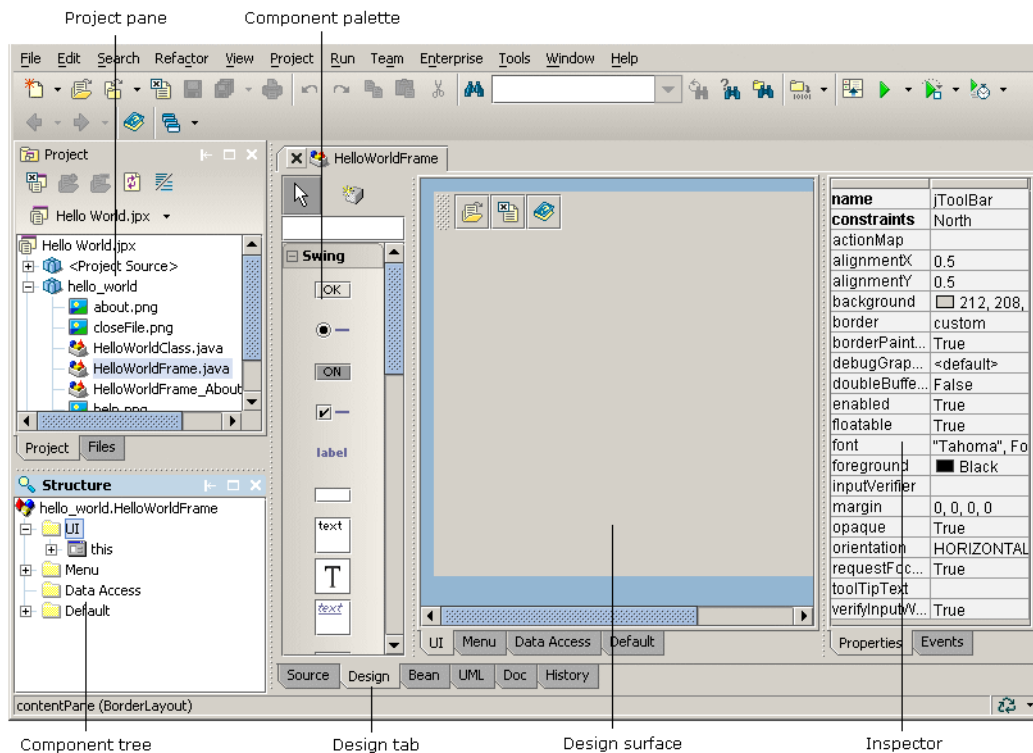
Step 4: Customizing your application's user interface

Follow these steps to customize your application's user interface:

- 1 Double-click `HelloWorldFrame.java` in the project pane if it's not already open.
- 2 Click the Design tab at the bottom of the content pane to change to the design view.

The UI designer appears in the content pane with the component palette on the left and the Inspector on the right. The structure pane, located to the left of the content pane, now contains a component tree containing components and such folders as UI, Menu, and Default.

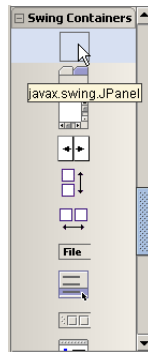
Figure 24.2 UI designer elements



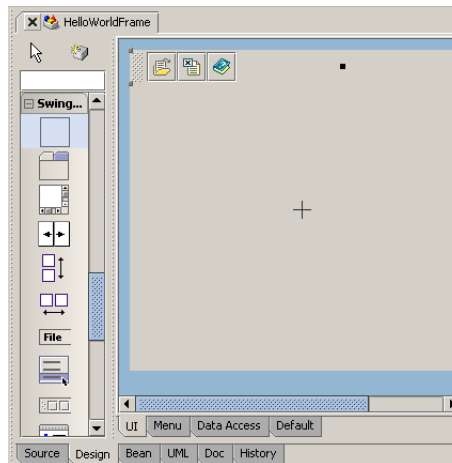
You use the component palette to add components to your UI and the Inspector to modify properties and add events to your code. The components are grouped into categories on expanding/collapsing pages. The designer opens with the Swing components expanded and everything else collapsed. To see the other pages, use the scroll bar at the right of the palette.

- 3 Scroll down the palette and click the Swing Containers page to expand it. Click the `JPanel` component to add a panel to your design.

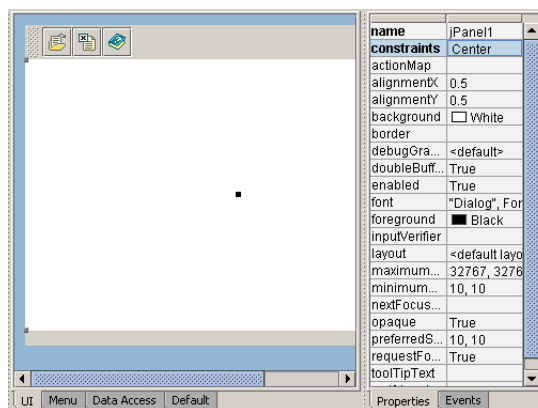
Tip Place the cursor over a component on the palette to see its name in a tool tip.



- 4 Click the center of the frame in the UI designer to drop the component into the center of your design's frame.

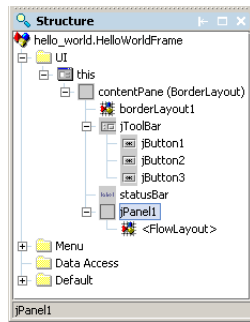


Note The `constraints` property in the Inspector should be `Center`. If not, click the column to the right of the `constraints` property, and choose `Center` from the drop-down list.



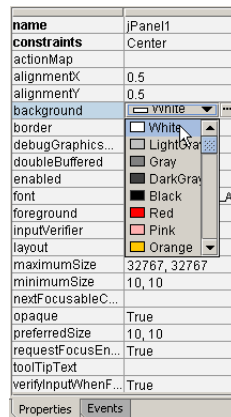
Step 4: Customizing your application's user interface

`jPanel1` is now selected in your application design and in the component tree.

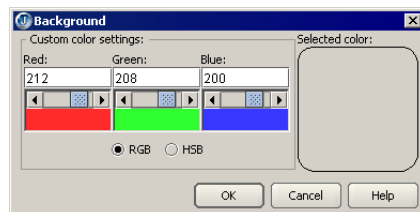


Note The component selected in the component tree or the UI designer displays in the status bar below the structure pane.

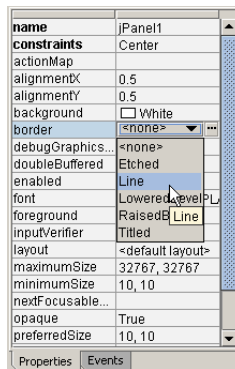
- 5 Set the background color of `jPanel1` to White in the Inspector.
 - a Click the column to the right of the `background` property in the Inspector.
 - b Click the *Down arrow* to open the color drop-down list and select White at the top of the list.



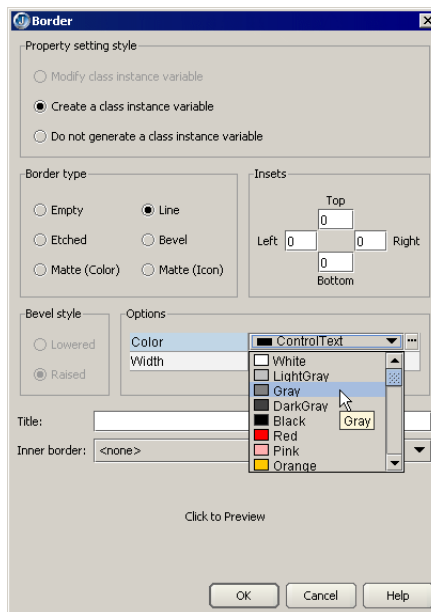
Note If you click too far right in the column, you'll click the ellipsis button which opens the Background dialog box. You could also use this to choose white by entering 255 for each RGB color value (red, green, and blue.)



- 6 Add a line border to `jPanel1` and change the border color to Gray.
 - a Click the column to the right of the `border` property in the Inspector.
 - b Click the *Down arrow* to open the border drop-down list and select Line.

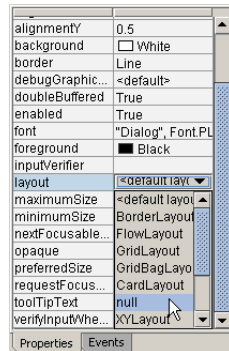


- c Click the ellipsis button (...) to access the Border dialog box.
 - d Notice the Options group that contains Color and Width options. Click down arrow to the right of the Color option and select Gray.



- e Click OK to close the Border dialog box.

- 7 Change the layout manager for `JPanel1` to `null`.
 - a Click the column to the right of the `layout` property in the Inspector.
 - b Choose `null` from the drop-down list.



No layout, or `null`, is a good choice when prototyping your design. Because `null` does not use a layout manager, you can place components exactly where you want them. However, because `null` uses absolute positioning of components instead of relative positioning, the UI will not resize properly when the user resizes the application window. Therefore, it's recommended that you never leave a container in `null` for deployment. Later in the tutorial, you'll switch to an appropriate portable layout for your design before deploying it.

Note The message pane appears only when a message is displayed. Therefore, you won't see the message pane yet if you are following these steps.

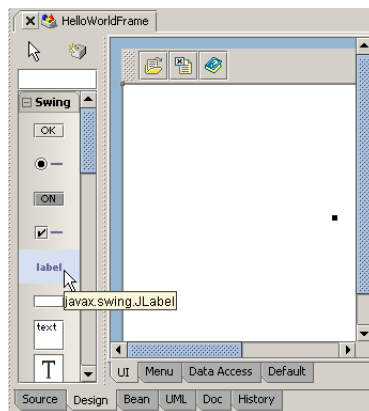


- 8 Choose `File|Save All` or click the `Save All` button on the toolbar to save the project.

Step 5: Adding a component to your application

Now, you'll use the component palette to add a label to the panel.

- 1 Select the `Swing` page on the palette and click the `JLabel` component.

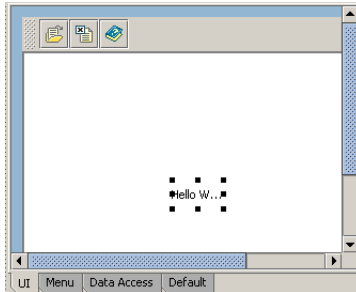


- 2 Drop the component into your design's `JPanel` using one of the following two methods:
 - Click `JPanel1` in the component tree. This places it in the upper-left corner of the panel.
 - Click the UI designer, which is filled with the `JPanel1` component. The upper-left corner of the component is placed where you click.

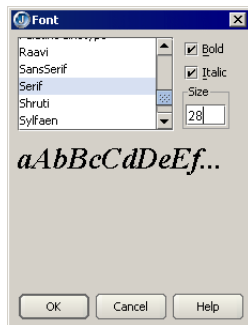
Note that `JLabel1` is added below `JPanel1` in the component tree. If you drop the component in the wrong place, you can select it in the component tree or in the designer and press the `Delete` key. Then try again.

- 3 Click the middle of the label component in the designer, and drag it to the center of the panel.
- 4 Select `jLabel1` in the component tree and complete the following steps:
 - a Double-click the column to the right of the `text` property in the Inspector and type `Hello World!` and press `Enter`.

“Hello World!” partially appears in the label. Don’t worry if it doesn’t completely show in the label. You’ll fix that after changing the font.

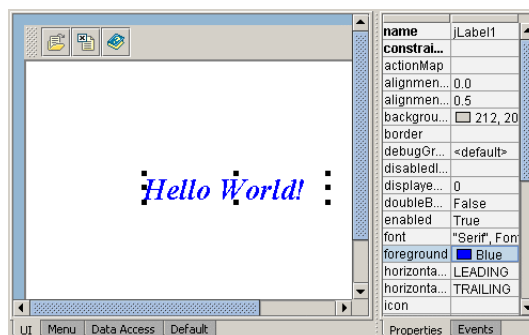


- b Click the column to the right of the `font` property to set the font. Click the ellipsis button to open the Font dialog box.
- c Choose `Serif` from the list of fonts and check `Bold` and `Italic`. Enter `28` in the `Size` box, then click `OK`.



- d Resize `jLabel1` by dragging the black handles until “Hello World!” is completely visible.
- e Click the column to the right of the `foreground` property in the Inspector to set the color of the “Hello World!” text. Click the *Down arrow* and select `Blue` from the drop-down list of colors.

Your design now looks similar to this:



- 5 Choose `File|Save All` or click the `Save All` button on the toolbar to save the project.

Step 6: Editing your source code

You can change information in the About box by directly editing the code. The default application version created by the Application wizard is version 1.0.

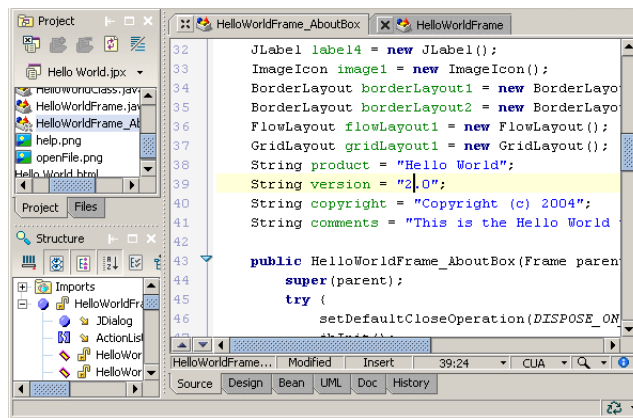
- 1 Double-click `HelloWorldFrame_AboutBox.java` in the project pane to open the file.

The content pane changes to the source view where you can edit the code in the editor.

- 2 Choose Search|Find to open the Find/Replace Text dialog box.
- 3 Type the following line of code in the Text to Find list box:

```
String version = "1.0";
```

- 4 Click Find. The editor finds the selected text.
- 5 Select 1.0 and enter 2.0 inside the quotes.



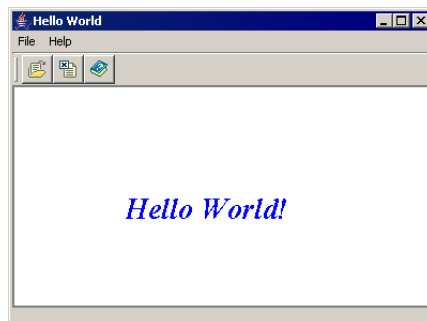
- 6 Choose File|Save All.

Step 7: Compiling, building, and running your application

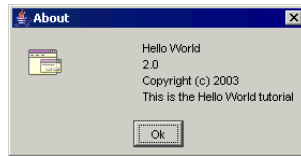
Now you can compile, build and run the application.

- 1 Choose Project|Make Project “HelloWorld.jpx” to compile the project.
- 2 Choose Run|Run Project.

The “Hello World” application is displayed:



- 3 Choose Help|About. The version data you changed is now displayed in the About dialog box.



- 4 Click OK to close the dialog box.
- 5 Choose File|Exit in your “Hello World” application to close it.

Step 8: Running your application from the command line

You can also run the application outside the JBuilder environment from the command line.

Note The `<jdk>/bin/` directory which contains the **java** command must be on your path. If **java** is on your path, when you type `java` at the command line, information explaining the command should display. If it's not on your path, you need to run the application from within the `<jdk>/bin/` directory.

To run the application, assuming **java** is on your path,

- 1 Open the command-line window.
- 2 Run the application with the following command on one line at the command prompt:

```
java -classpath
/<home>/jbproject/HelloWorld/classes helloworld.HelloWorldClass
```

Note Your particular version of Windows may require a backslash (\).

This command should be in the following form:

```
java -classpath classpath package-name.main-class-name
```

In this example,

- `java` = the launcher for the Java application.
- `-classpath` = an option that sets the search path for application classes and resources.
- `classpath` = `/<home>/jbproject/HelloWorld/classes`

The classpath should point to the directory containing the compiled classes. In this example, the `classes` directory was set as the output path for compiled classes on Step 1 of the Project wizard.

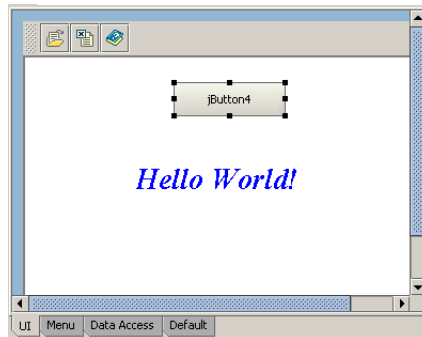
- `<home>` = your home directory, for example, `c:\winnt\profiles\<username>`
- `package-name` = `helloworld`
- `main-class-name` = `HelloWorldClass`

- 3 Close the “Hello World” application.

Step 9: Adding an event to a button

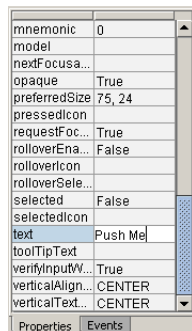
Next, you'll add another Swing component to your application.

- 1 Open `HelloWorldFrame.java` and click the Design tab to switch to the UI designer.
- 2 Click the `JButton` component on the Swing page of the component palette and drop it on `jPanel1` in either the component tree or the panel on the design surface. `jButton4` is added below `jPanel1` in the component tree.
- 3 Click `jButton4` in the design and drag it to the top center of the design as shown in the image below.



- 4 Change the `Text` property in the Inspector for `jButton4` to `Push Me`. Press `Enter`. Enlarge the button by dragging the black handles until "Push Me" shows completely.

The Inspector should look like this:



- 5 Click the Inspector's `Events` tab to define what happens when `jButton4` is pressed. We want the color of the text (`Foreground` property) to change when the button is pressed.
- 6 Double-click the column to the right of the `ActionPerformed` event.



JBUILDER switches to the editor where the following skeleton code has been added for the `ActionPerformed` event.

```
void jButton4_actionPerformed(ActionEvent e) {

}
```

You can now enter code that defines the event.

- 7 Type the following code indicated in bold:

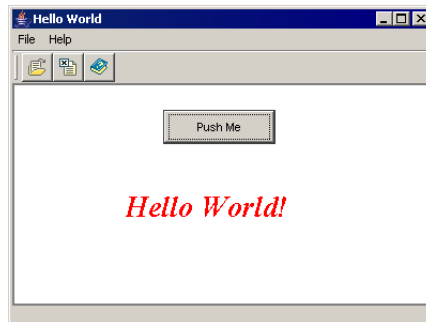
```
void jButton4_actionPerformed(ActionEvent e) {
    jLabel1.setForeground(new Color(255,0,0));
}
```

Tip

Use CodeInsight to complete the code for you. Type `jLabel1.` and wait for the pop-up window or press *Ctrl+spacebar* to invoke it. If CodeInsight fails to appear, check the settings on the CodeInsight page of the Preferences dialog box (Tools|Preferences|Editor). Type `setfor` to highlight `setForeground(Color)` in the pop-up window or use the arrow keys. Press *Enter* and type in the remainder of the above code.

Now, when you run the application and push the “Push Me” button, “Hello World!” should turn red.

- 8 Choose File|Save All.
- 9 Choose Project|Make Project “HelloWorld.jpx”.
- 10 Choose Run|Run Project.
- 11 Click the “Push Me” button. The color of the “Hello World!” text turns red.



- 12 Choose File|Exit to close the “Hello World” application.

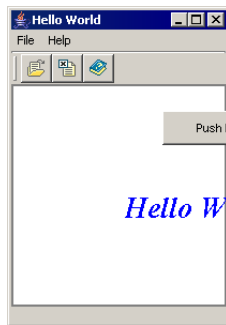
Step 10: Completing your UI

As the final step in completing your application, you need to change your layout to a portable layout. Remember, if you leave `JPanel1` as null or no layout, the components in your UI will not resize nor reposition when the user resizes the application window at runtime. Because `null` uses absolute positioning, the components remain in the same location, no matter what size the window. Therefore, `null` is not a good layout manager for final deployment of your application.

To demonstrate this window resizing problem with `null`, run the application as follows:

- 1 Right-click `HelloWorldClass.java`, which contains the `main()` method, and choose Run using “HelloWorldClass”.
- 2 Resize the “Hello World” application window, making it larger and smaller, and observe the behavior of the components.

Notice that the label and button components do not move as you resize the window, because their position is absolute rather than relative to the window size.



- 3 Close the “Hello World” application.

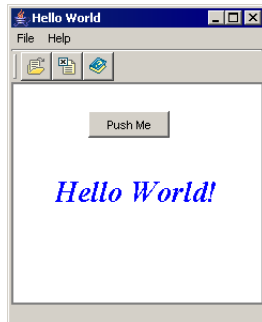
Portable layouts, unlike `null`, use relative positioning which is dynamic. Components in portable layouts reposition correctly when the user resizes the application window. In this example, you’ll change the layout to `GridBagLayout`.

To modify the layout behavior,

- 1 Return to the `HelloWorldFrame.java` file in the content pane and click the Design tab to switch to the designer.
- 2 Choose `jPanel1` in the component tree.
- 3 Choose the Inspector’s Properties tab.
- 4 Change `jPanel1`’s layout property to `GridBagLayout` in the Inspector. Select each component in the designer and notice the grid area each component fills.

`GridBagLayout` is a good choice for laying out components of varying sizes in a grid.

- 5 Save and run the application.
- 6 Resize the application window and notice how the components reposition as the window changes in size.



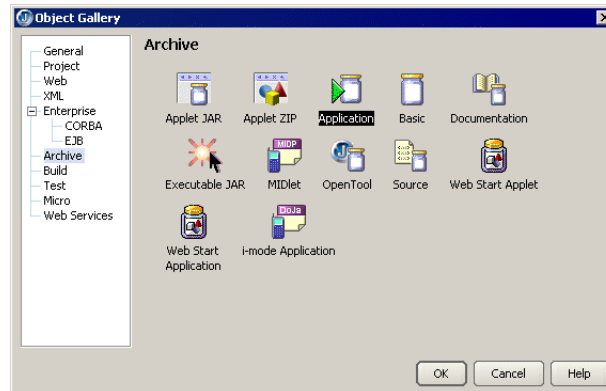
- 7 Close the “Hello World” application.

Step 11: Preparing your application for deployment

The Archive Builder collects all the files needed to distribute your program and can archive them into a Java archive file (JAR file).

To deploy your application:

- 1 Choose File|New|Archive and double-click the Application icon open the Application Archive Builder.



- 2 Click Finish to accept all the defaults for this wizard. Or, if you are curious to see what the rest of the steps contain, click Next on each step of the Archive Builder, then Finish on the last step. For more information about the remaining steps, click the Help button on each step.

See “Deploying Java Programs” and “Using the Archive Builder” in *Building Applications with JBuilder* to learn about archiving and deploying applications.

An archive node called `Application` appears in the project pane. You can modify this file by right-clicking and selecting Properties.

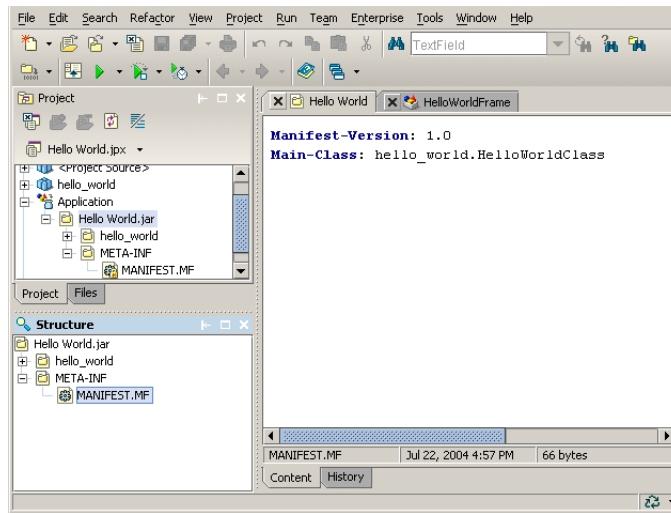
- 3 Right-click the `Application` node and choose Make or choose Project|Make Project “HelloWorld.jpx” to compile your application and create the JAR file.

The Archive Builder gathers all the files in the project’s output path (Project|Project Properties|Paths) into the JAR file.

- 4 Click the expand icon next to the `Application` archive node to expand the node and see the `HelloWorld.jar` archive file.
- 5 Double-click the JAR file in the project pane.

Step 12: Running your deployed application from the JAR file

The manifest file appears in the content pane and the contents of the JAR file appear in the structure pane.



6 Save your project.

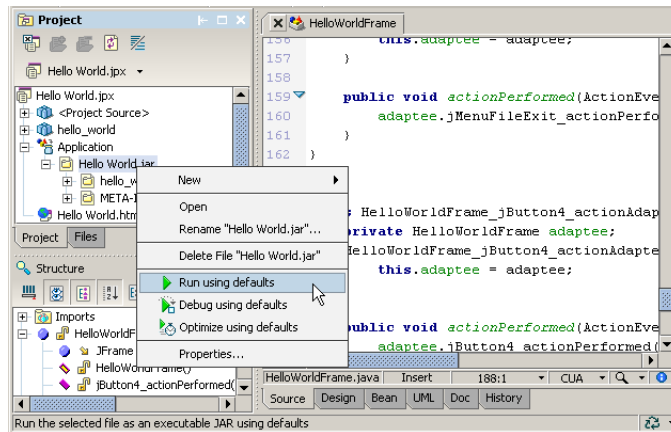
Step 12: Running your deployed application from the JAR file

**This is a feature of
JBuilder Developer
and Enterprise**

Use a simple process to test your deployed application: run it from the executable JAR file created with the Archive wizard in the previous step. The project pane displays the JAR file, `HelloWorld.jar`, within the `Application` node.

- 1 Expand the `Application` node in the project pane.
- 2 Right-click the `HelloWorld.jar` file.
- 3 Choose `Run using defaults`.

The Hello World application loads and runs, (the Hello World dialog box, with the Push Me button in the middle pane should appear).



Step 13: Running your deployed application from the command line

To test the deployed application, you can run the JAR file from the command line.

Note The `<jdk>/bin/` directory which contains the **java** command must be on your path. If **java** is on your path, when you type `java` at the command line, information explaining the command should display. If it's not on your path, you need to run the application from within the `<jdk>/bin/` directory.

- 1 Open the command-line window.
- 2 Enter the following command on one line at the prompt:

```
java -jar /<home>/jbproject/HelloWorld/HelloWorld.jar
```

Note For Windows, use a backslash (\).

The command must have the following form:

```
java -jar classpath
```

- 3 The “Hello World” application loads and runs.

In this example,

- `java` = the launcher for the Java application
- `jar` = an option that sets the search path for the JAR file (which contains application classes, including the main class, and resources)
- `<home>` = your home directory, for example, `c:\winnt\profiles\<username>` or `c:\Documents and Settings\<username>`
- `classpath` = `/<home>/jbproject/HelloWorld/HelloWorld.jar`

The classpath should include the JAR file and its correct location. In this example, the JAR file is located in the project directory, `HelloWorld`. The Archive Builder saves it there by default.

For more information about using the Archive Builder wizard, see “Using the Archive Builder” in *Building Applications with JBuilder*.

For information on JAR files, see the JAR tutorial at <http://java.sun.com/docs/books/tutorial/jar/index.html>.

Congratulations, you’ve created your first application with JBuilder. Now that you’re familiar with JBuilder’s development environment, you’ll find its many time-saving features make your programming easier.

For other tutorials involving UI design in JBuilder, see “Building a Java text editor,” “Creating a GridBagLayout in JBuilder,” and “Creating a UI with nested layouts” in *Designing Applications with JBuilder* in online help.

HelloWorld source code

Select a link to view the code:

- [Source code for “Source code for HelloWorldFrame.java” on page 212](#)
- [Source code for “Source code for HelloWorldClass.java” on page 215](#)
- [Source code for “Source code for HelloWorldFrame_AboutBox.java” on page 216](#)

Source code for HelloWorldFrame.java

```

package hello_world;

import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JToolBar;
import javax.swing.JButton;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.*;
import javax.swing.border.*;
import com.borland.jbcl.layout.*;

/**
 * <p>Title: Hello World</p>
 *
 * <p>Description: This is the Hello World tutorial</p>
 *
 * <p>Copyright: Copyright (c) 2004</p>
 *
 * <p>Company: My Company</p>
 *
 * @author My Name
 * @version 1.0
 */
public class HelloWorldFrame extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    JMenuBar jMenuBar1 = new JMenuBar();
    JMenu jMenuFile = new JMenu();
    JMenuItem jMenuItemFileExit = new JMenuItem();
    JMenu jMenuHelp = new JMenu();
    JMenuItem jMenuItemHelpAbout = new JMenuItem();
    JToolBar jToolBar = new JToolBar();
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();
    JButton jButton3 = new JButton();
    ImageIcon image1 = new ImageIcon(hello_world.HelloWorldFrame.class.
        getResource("openFile.png"));
    ImageIcon image2 = new ImageIcon(hello_world.HelloWorldFrame.class.
        getResource("closeFile.png"));
    ImageIcon image3 = new ImageIcon(hello_world.HelloWorldFrame.class.
        getResource("help.png"));
    JLabel statusBar = new JLabel();
    JPanel jPanel1 = new JPanel();
    Border border1 = BorderFactory.createLineBorder(Color.gray, 2);
    JLabel jLabel1 = new JLabel();
    JButton jButton4 = new JButton();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    public HelloWorldFrame() {
        try {
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            jbInit();
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

```

```

/**
 * Component initialization.
 *
 * @throws java.lang.Exception
 */
private void jbInit() throws Exception {
    contentPane = (JPanel) getContentPane();
    contentPane.setLayout(borderLayout1);
    setSize(new Dimension(400, 300));
    setTitle("Hello World");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new
        HelloWorldFrame_jMenuFileExit_ActionAdapter(this));
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new
        HelloWorldFrame_jMenuHelpAbout_ActionAdapter(this));
    jPanell.setBackground(Color.white);
    jPanell.setAlignmentY((float) 0.5);
    jPanell.setDebugGraphicsOptions(0);
    jPanell.setLayout(gridBagLayout1);
    jPanell.setBorder(border1);
    jLabell.setFont(new java.awt.Font("Serif", Font.BOLD | Font.ITALIC, 28));
    jLabell.setForeground(Color.blue);
    jLabell.setText("Hello World!");
    jLabell.setBounds(new Rectangle(112, 119, 151, 37));
    jButton4.setBounds(new Rectangle(87, 96, 74, 24));
    jButton4.setText("Push Me");
    jButton4.addActionListener(new
        HelloWorldFrame_jButton4_actionAdapter(this));
    jMenuBar1.add(jMenuFile);
    jMenuFile.add(jMenuFileExit);
    jMenuBar1.add(jMenuHelp);
    jMenuHelp.add(jMenuHelpAbout);
    setJMenuBar(jMenuBar1);
    jButton1.setIcon(image1);
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    contentPane.add(jToolBar, BorderLayout.NORTH);
    contentPane.add(statusBar, BorderLayout.SOUTH);
    contentPane.add(jPanell, java.awt.BorderLayout.CENTER);
    jPanell.add(jLabell, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
        , GridBagConstraints.WEST, GridBagConstraints.NONE,
        new Insets(23, 121, 103, 117), 7, 15));
    jPanell.add(jButton4, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
        , GridBagConstraints.CENTER, GridBagConstraints.NONE,
        new Insets(48, 152, 0, 169), 1, 0));
}

/**
 * File | Exit action performed.
 *
 * @param actionEvent ActionEvent
 */
void jMenuFileExit_actionPerformed(ActionEvent actionEvent) {
    System.exit(0);
}

```

```

    /**
     * Help | About action performed.
     *
     * @param actionEvent ActionEvent
     */
    void jMenuItemHelpAbout_actionPerformed(ActionEvent actionEvent) {
        HelloWorldFrame_AboutBox dlg = new HelloWorldFrame_AboutBox(this);
        Dimension dlgSize = dlg.getPreferredSize();
        Dimension frmSize = getSize();
        Point loc = getLocation();
        dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x,
            (frmSize.height - dlgSize.height) / 2 + loc.y);
        dlg.setModal(true);
        dlg.pack();
        dlg.show();
    }

    public void jButton4_actionPerformed(ActionEvent e) {
        jLabel1.setForeground(new Color(255, 0, 0));
    }
}

class HelloWorldFrame_jMenuItemFileExit_ActionAdapter implements ActionListener {
    HelloWorldFrame adaptee;

    HelloWorldFrame_jMenuItemFileExit_ActionAdapter(HelloWorldFrame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuItemFileExit_actionPerformed(actionEvent);
    }
}

class HelloWorldFrame_jButton4_actionAdapter implements ActionListener {
    private HelloWorldFrame adaptee;
    HelloWorldFrame_jButton4_actionAdapter(HelloWorldFrame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e) {
        adaptee.jButton4_actionPerformed(e);
    }
}

class HelloWorldFrame_jMenuItemHelpAbout_ActionAdapter implements ActionListener {
    HelloWorldFrame adaptee;

    HelloWorldFrame_jMenuItemHelpAbout_ActionAdapter(HelloWorldFrame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuItemHelpAbout_actionPerformed(actionEvent);
    }
}

```


Source code for HelloWorldClass.java

```

package hello_world;

import java.awt.Toolkit;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Dimension;

/**
 * <p>Title: Hello World</p>
 *
 * <p>Description: This is the Hello World tutorial</p>
 *
 * <p>Copyright: Copyright (c) 2004</p>
 *
 * <p>Company: My Company</p>
 *
 * @author My Name
 * @version 1.0
 */
public class HelloWorldClass {
    boolean packFrame = false;

    /**
     * Construct and show the application.
     */
    public HelloWorldClass() {
        HelloWorldFrame frame = new HelloWorldFrame();
        // Validate frames that have preset sizes
        // Pack frames that have useful preferred size info, e.g. from their layout
        if (packFrame) {
            frame.pack();
        } else {
            frame.validate();
        }

        // Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
            (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }

    /**
     * Application entry point.
     *
     * @param args String[]
     */
}

```

```

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.
                        getSystemLookAndFeelClassName());
                } catch (Exception exception) {
                    exception.printStackTrace();
                }

                new HelloWorldClass();
            }
        });
    }
}

```

Source code for HelloWorldFrame_AboutBox.java

```

package hello_world;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;

/**
 * <p>Title: Hello World</p>
 *
 * <p>Description: This is the Hello World tutorial</p>
 *
 * <p>Copyright: Copyright (c) 2004</p>
 *
 * <p>Company: My Company</p>
 *
 * @author My Name
 * @version 1.0
 */
public class HelloWorldFrame_AboutBox extends JDialog implements ActionListener {
    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();
    JPanel insetsPanel1 = new JPanel();
    JPanel insetsPanel2 = new JPanel();
    JPanel insetsPanel3 = new JPanel();
    JButton button1 = new JButton();
    JLabel imageLabel = new JLabel();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    ImageIcon image1 = new ImageIcon();
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    FlowLayout flowLayout1 = new FlowLayout();
    GridLayout gridLayout1 = new GridLayout();
    String product = "Hello World";
    String version = "2.0";
    String copyright = "Copyright (c) 2004";
    String comments = "This is the Hello World tutorial";
}

```

```

public HelloWorldFrame_AboutBox(Frame parent) {
    super(parent);
    try {
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        jbInit();
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

public HelloWorldFrame_AboutBox() {
    this(null);
}

/**
 * Component initialization.
 *
 * @throws java.lang.Exception
 */
private void jbInit() throws Exception {
    image1 = new ImageIcon(hello_world.HelloWorldFrame.class.getResource(
        "about.png"));
    imageLabel.setIcon(image1);
    setTitle("About");
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
    button1.setText("OK");
    button1.addActionListener(this);
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
    setResizable(true);
}

/**
 * Close the dialog on a button event.
 *
 * @param actionEvent ActionEvent
 */
public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == button1) {
        dispose();
    }
}
}

```


Chapter 25

Tutorial: Building an applet

This tutorial steps you through creating an AWT applet using the JBuilder integrated development environment (IDE). For more information on the IDE and its components, see “JBuilder’s work environment” available from the Help menu.

The tutorial shows how to

- Create a project for the applet.
- Use the Applet wizard to create an AWT applet.
- Build and run the applet.
- Customize the applet’s UI.
- Add AWT components, such as Choice, Label, and Button.
- Edit the source code.
- Deploy the applet.
- Modify the HTML file.
- Run the deployed applet from the command line
- Test the applet.

See [“Applet source code” on page 245](#) to view source code associated with this tutorial.

Important Before beginning this tutorial, read the [“Overview” on page 220](#) which discusses important applet issues.

The Accessibility options section in the JBuilder Quick Tips contains tips on using JBuilder features to improve JBuilder’s ease of use for people with disabilities.

For information on documentation conventions used in this tutorial and other JBuilder documentation, see [“Documentation conventions” on page 31](#).

To suggest ways to improve this tutorial, send e-mail to jgpubs@borland.com.

Overview

It's important to remember when designing applets that browser support for Java can be limited (depending on the browser you use). In this tutorial you design your applet using Java AWT components, which are generally supported by most browsers, such as Netscape Navigator and Microsoft Internet Explorer. However, if you want to design an applet using Swing components, you need to determine what Java capabilities your browser supports. Generally, you can find browser support information for Java applets by looking in the Tools, Preferences, or Help menu contents of Netscape or Internet Explorer. You can troubleshoot your applet in the browser by checking the Java Console error messages. The safest way to design your applet is by using AWT components and the JDK that the browser supports.

Important For more information on running applets in JBuilder, see “Running applets” in the *Developing Web Applications*.

Good Evening applet

The “Good Evening” applet you create in this tutorial contains a drop-down list of language choices. When you select a language, such as German, the panel below the selection changes to the German translation of “Good Evening”: “Guten Abend.”

When you finish the tutorial, your applet will look similar to this when running in JBuilder's applet viewer:



If you run the finished applet in Sun's applet viewer, it will look like this:



For the complete applet code, see [“Applet source code” on page 245](#).

For in-depth information on applets and deployment, see “Working with applets” and “Deploying Java programs” in *Building Applications with JBuilder*.

Step 1: Creating the project

Before beginning this tutorial, read the [“Overview” on page 220](#) which discusses such important applet issues as browser support, JDK versions, and applet components.

The first step in creating your applet in JBuilder is to create a project in which to store it. The Project wizard creates one for you:

- 1 Choose File|New Project to open the Project wizard.
- 2 Make the following changes to the appropriate fields in Step 1 of the Project wizard:
 - a Type `MyFirstApplet` in the Name field.

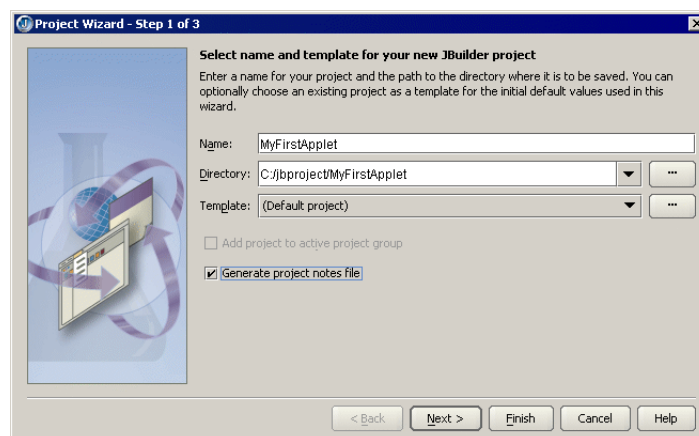
Note

As you type the project name in the Name field, the same name is entered in the Directory field. By default, JBuilder uses this project name to create the project's directory name and the package name for the classes. Projects in JBuilder are saved by default in the `/home/jbproject/` directory. For information about the home directory, see [“Documentation conventions” on page 31](#). For more information on projects, see [Chapter 6, “Working with projects.”](#)

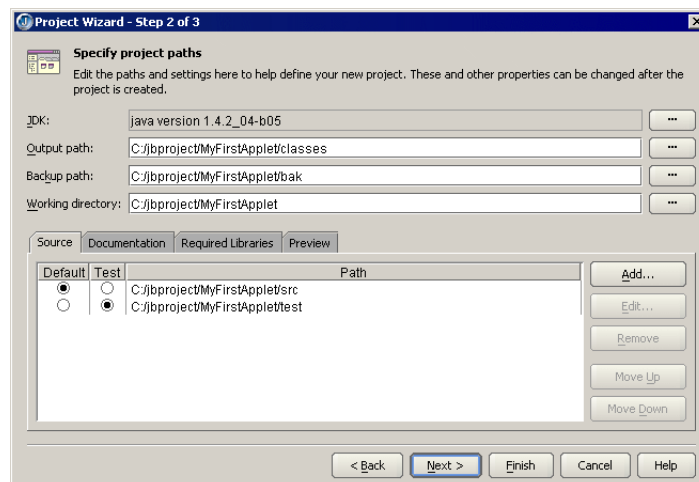
- b Choose `jpx` from the Type drop-down menu.
- c Check the Generate Project Notes File option.

When you check this option, the Project wizard creates an HTML file for project notes and adds it to the project.

- 3 Accept all other defaults in Step 1.



- 4 Click Next to go to Step 2 of the Project wizard.



- 5 Accept the default paths in Step 2. Note where the compiled class files, project files, and source files will be saved. For more information on paths and how JBuilder saves files, see “Managing paths” in *Building Applications with JBuilder*.

Tip

If you prefer to create your applet using an earlier version of the JDK, change the JDK version in this step. Although JBuilder Foundation does not support JDK switching, you can edit the existing JDK in the Configure JDKs dialog box (Tools | Configure JDKs). For JDK 1.1.x, you also need to download the JDK 1.1.x-specific version of the JFC.

See also

- “Setting project properties” in *Building Applications with JBuilder* for information on switching or editing the JDK
- “Running applets” in *Developing Web Applications* for information on running applets in JBuilder
- “How JBuilder constructs paths” on *Building Applications with JBuilder*
- “Where are my files?” in *Building Applications with JBuilder*

- 6 Click Next to go to Step 3 of the wizard.

- 7 Make the following changes in the appropriate fields of Step 3:

- a Accept the Encoding and Automatic Source Packages defaults.
- b Type `Good Evening` in the Title field of the class Javadoc fields.
- c Enter your name, company name, and a description of your applet in the appropriate optional fields.

Note

The information in the class Javadoc fields appears in the project HTML file and as optional header comments in the source code.

Project Wizard - Step 3 of 3

Specify general project settings
Enter settings here to help define your new project. These and other properties can be changed after the project is created.

Encoding: Default

Automatic source packages

☒ Enable source package discovery and compilation

Deepest package exposed:

Class Javadoc fields:

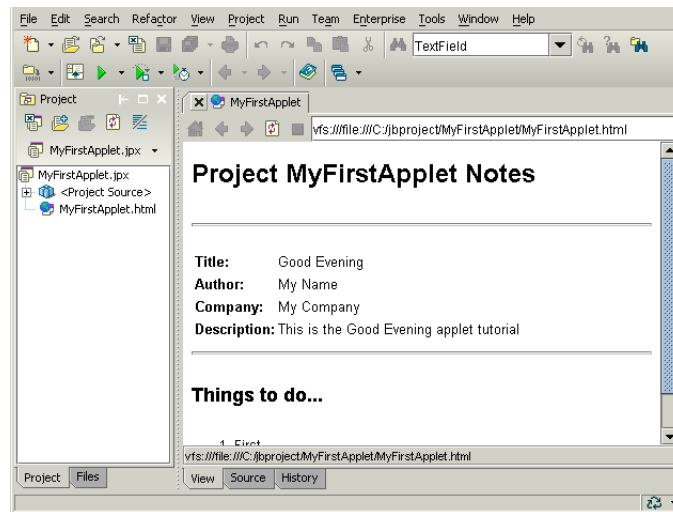
Label	Text
Title:	Good Evening
Description:	This is the Good Evening applet tutorial
Copyright:	Copyright (c) 2004
Company:	My Company
@author	My Name
@version	1.0

☐ Include references from project library class files

☐ Diagram references from generated source

< Back Next > Finish Cancel Help

- 8 Click the Finish button. The wizard creates a project file and a project notes file, `FirstApplet.jpx` and `FirstApplet.html`, that appear in the project pane of the JBuilder's IDE.
- 9 Double-click the HTML file to see the project notes in the content pane.



Step 2: Generating your source files

The Applet wizard creates a `.java` file and an applet HTML file and places them in the project you just created with the Project wizard.

To generate these source files for your applet, follow these steps:

- 1 Choose File/New and choose the Web page of the object gallery.
- 2 Double-click the Applet icon to open the Applet wizard.
- 3 Type `GoodEveningApplet` in the Class Name field.

This is a case-sensitive Java class name.

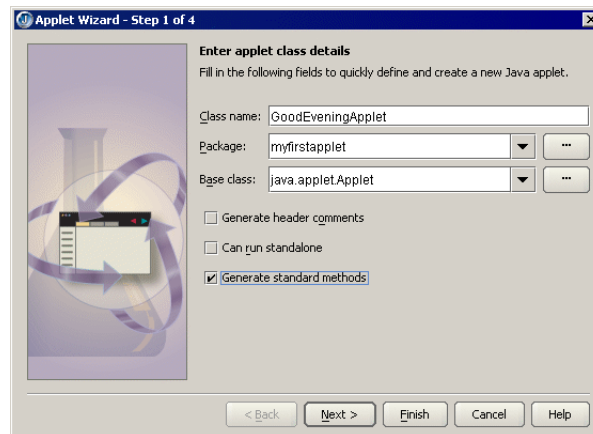
Note The fully qualified class name (package name + class name) is `firstapplet.GoodEveningApplet.class`.

- 4 Accept the default package name, `firstapplet`, in step 1. By default, the wizard takes the package name from the project file name, `FirstApplet.jpx`.
- 5 Accept the default Base Class, `java.applet.Applet`.

Caution If you create your applet using `javax.swing.JApplet`, your applet won't run in most web browsers. At the time of this writing, Swing is not yet supported by the web browsers. See [“Overview” on page 220](#) for more information.

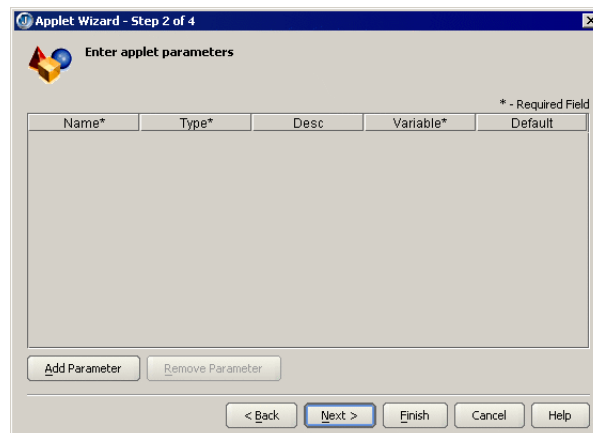
6 Check Generate Standard Methods.

Step 1 of the Applet wizard should look like this:



7 Click Next to go to step 2.

In this step, you can add parameters to your applet. The wizard adds the `<param>` tags inside the `<applet>` tags in the applet's HTML file and also inserts code for handling parameters in the source code. Applet parameters, the equivalent of command-line arguments for applications, allow you to customize your applet.



See also

- “Defining and using parameters” at <http://www.java.sun.com/docs/books/tutorial/applet/appletonly/param.html>

8 Do not add any parameters for this tutorial. Click Next to go to step 3 of the Applet wizard.

9 Make the following changes in step 3:

- a Accept the default option, Generate HTML Page.

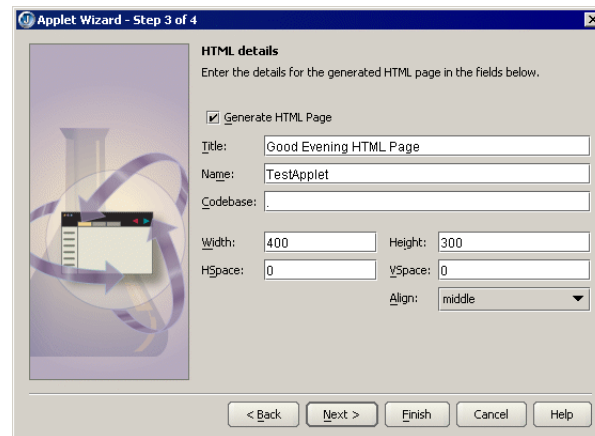
When you select this option, the HTML file that calls the applet is created. This HTML file contains the `<applet>` tag and various attributes.

- b Enter Good Evening HTML Page in the Title field.

The title displays in the web browser window when the applet is running.

c Accept the default values for all other attributes.

Step 3 of the Applet wizard should look like this:



The `<applet>` tag in the HTML file can include these attributes:

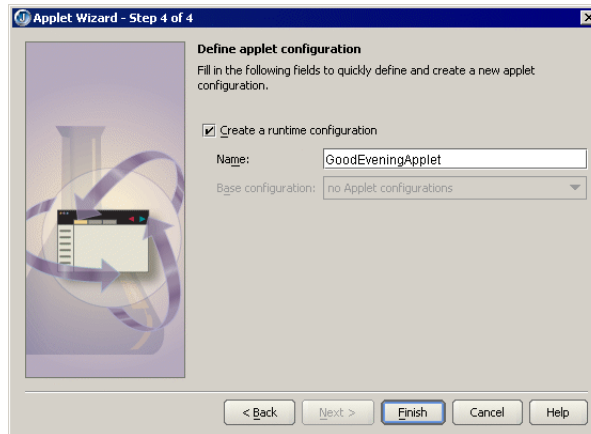
<code>codebase</code>	This optional attribute specifies the path relative to the applet HTML file that the browser searches to find any necessary class files. A value of <code>"."</code> specifies the same directory as the HTML file running the applet. The <code>codebase</code> attribute is required when the class files are in a different directory than the HTML file.
<code>code</code>	This required attribute, which is automatically inserted by JBuilder's Applet wizard, is the fully qualified class name (package name + class name) of the applet class that contains the <code>init()</code> method. You'll see it in the HTML file when it is generated. In this example, the fully qualified class name is <code>firstapplet.GoodEveningApplet.class</code> .
<code>archive</code>	This optional attribute, which is not included in the code generated by the Applet wizard, is required when the applet is deployed in a JAR, ZIP, or CAB file. Archive files must be in the directory specified by <code>codebase</code> .
<code>name</code>	This optional attribute names the applet.
<code>width/height</code>	These required attributes determine the width and height of the applet display area in pixels.
<code>hspace/vspace</code>	These optional attributes determine the horizontal padding (left and right margins) and vertical padding (top and bottom margins) around the applet in pixels.
<code>align</code>	This optional attribute determines the alignment of the applet on the HTML page.

Important The values for `codebase`, `code`, `archive`, and `name` must be in quotation marks and are case-sensitive.

See also

- “`<applet>` tag attributes” in the *Developing Web Applications*.

- 10 Click Next to go to step 4, then click Finish to automatically create a runtime configuration and close the Applet wizard.

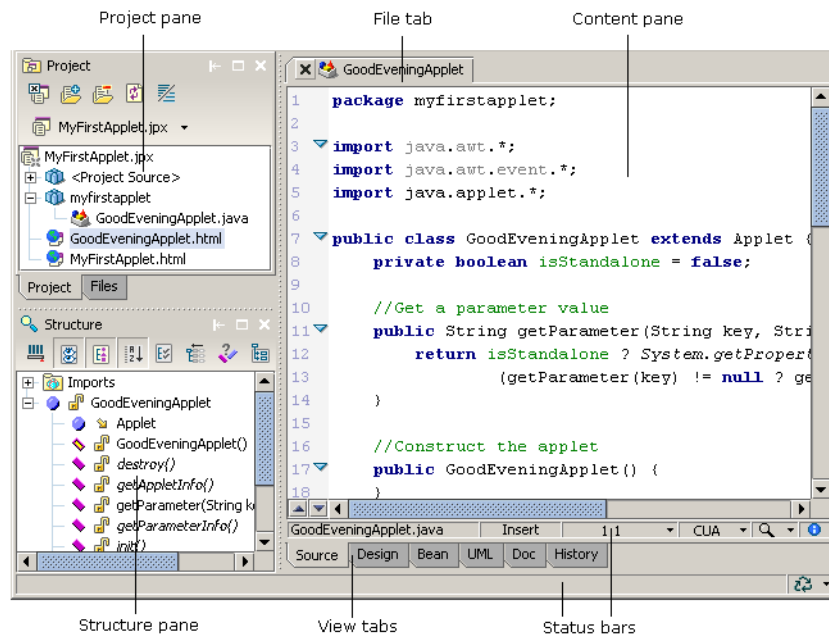


Two files are created and added to the project, `GoodEveningApplet.java` and `GoodEveningApplet.html`. `GoodEveningApplet.java` is open in the content pane.

Note

An automatic source package node named `firstapplet` also appears in the project pane if the Enable Source Package Discovery And Compilation option is enabled on the General page of the Project Properties dialog box (Project|Project Properties).

The JBuilder IDE displays the First Applet project, files, structure, and source code in the project, structure, and content panes.



Now, look at the source code for the two files.

1 Look at `GoodEveningApplet.java` and note the following:

- It contains the `init()` method. The applet HTML file must call the class that contains the `init()` method for the applet to run.
- The package name `firstapplet` is the first line of code. The class file is saved in a `firstapplet` directory according to Java conventions.
- The import statements import AWT packages, not Swing:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
```

2 Double-click `GoodEveningApplet.html` in the project pane and choose the Source tab at the bottom of the content pane to see the source code.

Notice that the wizard inserted the code value, `firstapplet.GoodEveningApplet.class`.

3 Choose File|Save All to save the source files and the project file.

Note JBuilder saves the source files to:

```
<JBuilder_projects>/FirstApplet/src/firstapplet/
```

The applet HTML file is saved to the `classes` directory:

```
<JBuilder_projects>/FirstApplet/classes/
```

The class files, after compiling, are saved to the output path:

```
<JBuilder_projects>/FirstApplet/classes/firstapplet/
```

JBuilder always follows the package hierarchy when saving files. In this example, the source and class files are saved within a `firstapplet` directory on the source and output paths to reflect the `firstapplet` package structure. These paths are set for the project by the Project wizard and can be viewed in the Project Properties dialog box (Project|Project Properties). In this tutorial, you accepted the default JBuilder paths in step 2 of the Project wizard.

Step 3: Building and running your applet

Now, build and run the applet. Building includes build tasks such as preparing non-Java files for compiling, compiling Java source files, copying resources, archiving, deploying, and so on. Compiling is the process of running the Java compiler. The compiler, which translates source code into Java bytecode, generates `.class` files.

Important For information on running JDK 1.1.x and 1.2/1.3 applets in JBuilder, see “Running applets” in the *Developing Web Applications*.



- 1 Choose Run|Run Project, or click the Run button to compile and run your applet. The Run menu and the Run button run the applet in JBuilder’s applet viewer, `AppletTestbed`.

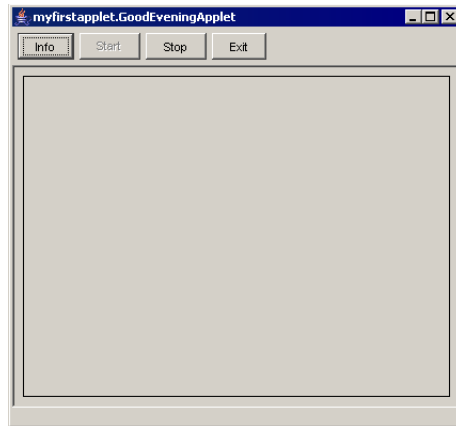
Tip

You can also right-click `GoodEveningApplet.html` in the project pane and select Run. This runs your applet in Sun’s `appletviewer` command-line tool.

Currently, the applet window is empty because the applet doesn’t have a user interface yet.

When you run your applet, the message pane appears at the bottom of the IDE where any compile-time errors are displayed. Correct any errors and run the applet again.

Your applet is displayed and should look like this in JBuilder's AppletTestbed:



You can change the applet's run settings in the Runtime Configurations dialog box. To access this dialog box quickly, choose Run|Configurations. Select the `GoodEveningApplet` on the Run page, and click Edit. The settings on the Runtime Configuration dialog box control the behavior of the Run menu and the Run icon on the tool bar. Choose the Main Class option to run your applet in JBuilder's AppletTestbed. Choose the HTML option to run your applet in Sun's `appletviewer`. When you use the Applet wizard to create your applet, the Main Class option is set by default.

Important

Applets run from the HTML file, which calls the class containing the `init()` method, not from the `.java` file. Any attempt to run the `.java` file results in an error message (unless the Can Run Standalone option was selected in Step 1 of the Applet wizard):

```
java.lang.NoSuchMethodError: main
Exception in thread "main"
```

- 2 Click Exit in the "Good Evening" applet to close it.
- 3 Right-click the `GoodEveningApplet` tab in the message pane and choose Remove "GoodEveningApplet" Tab to close any runtime messages.

Step 4: Customizing your applet's user interface

Now that the Applet wizard has generated the applet shell, you'll customize it with various components using the following steps.

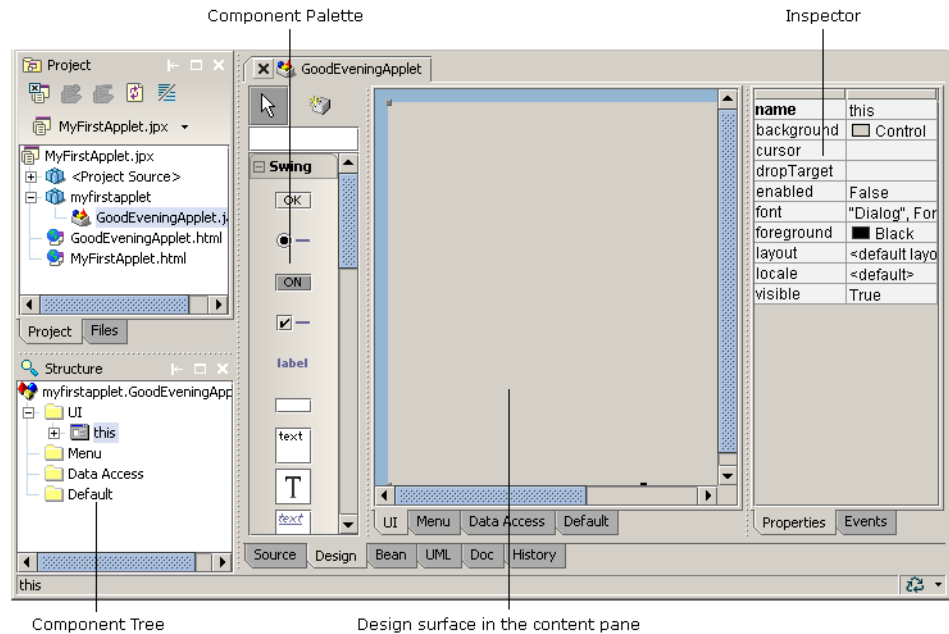
- 1 Click the Design tab at the bottom of the `GoodEveningClass.java` which should still be opened in the content pane. This switches the project to the design view and displays the various elements of the UI designer.

The UI designer consists of a drawing surface in the content pane, with a component palette at the left of it from which you'll choose components to add to the design. The structure pane now displays a component tree which shows the elements in the class you're designing and contains such folders as `UI`, `Menu`, and `Data Access`, and `Default`. The Inspector is to the right of the design surface and is used for setting component properties and adding events to your code.

Note

The JBuilder workspace (that is, the position of the various panes in the browser) is customizable, so the layout shown in the image below may not match yours exactly.

Figure 25.1 UI designer



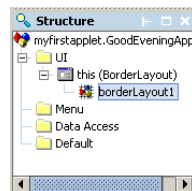
- 2 Change the layout of the main UI Panel to BorderLayout in the Inspector:
 - a Select the main UI Panel in the structure pane by clicking the `this` node in the component tree.
 - b Click to the right of the `layout` property on the Properties page of the Inspector.
 - c Choose `BorderLayout` from the drop-down list of layouts.

`BorderLayout` arranges a container's components in areas named North, South, East, West, and Center. Use `BorderLayout` when you want to force components to one or more edges of a container or fill the center of the container with a component. It is also the layout you want to use to cause a single component to completely fill its container.

Caution

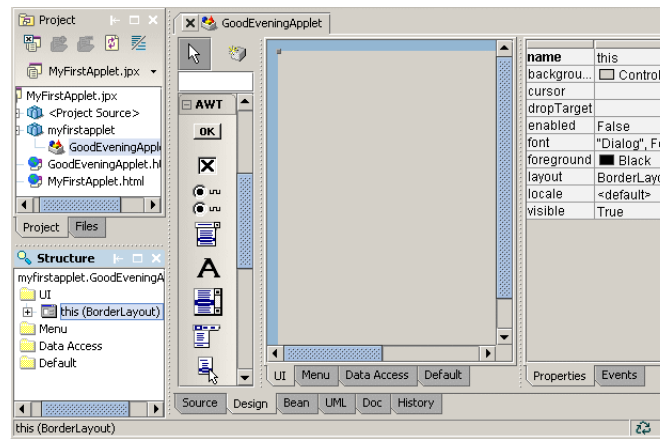
If at any time you choose `XYLayout` (a custom Borland layout) from the Inspector's drop-down list, JBuilder adds this import statement to the source code:
`com.borland.jbcl.layout.*`. This import statement is *not* removed when you change to a more portable layout before deployment. Your deployed applet won't run unless you remove this import statement, because it will be looking for the `jbcl` layout classes to import. You need to remove the import statement manually before deploying.

JBuilder's UI designer uses a default layout manager for each container, usually the layout of the AWT parent container. In the Java AWT, all panels use `FlowLayout` by default. To see the panel's layout manager, click the expand icon in the component tree to expand the selection. The layout manager displays as an item in the tree just below the parent container.



- 3 Collapse the Swing page on the component palette and expand the AWT page. You might need to scroll down on the component palette to find the AWT tab. AWT components, unlike Swing components, are supported by most web browsers.

Caution The UI designer always opens with the Swing components on the first page of the component palette. If you select these Swing components by mistake, your applet won't run in the web browsers. Be very careful to choose components from the AWT page of the palette.



- 4 Add two panels to the main applet panel (called `this` in the structure pane) following the steps below. The top panel will contain a drop-down list of languages to select from and a label to identify the list. The bottom panel will contain “Good Evening” in various languages.

- a Click the AWT `Panel` component, `java.awt.Panel`, on the component palette.

Tip Place the cursor over a component to see its name in a tool tip.

- b Click on `this` in the component tree in the structure pane to add the panel. `panel1` is added below `this` in the component tree.

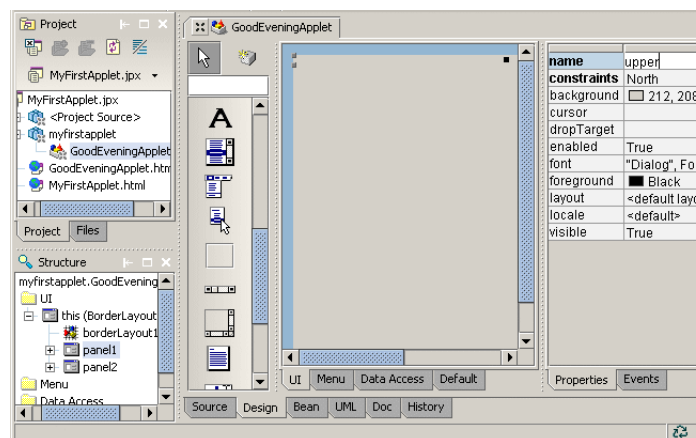
- c Repeat the previous two steps to add a second panel to `this` called `panel2`.

- 5 Modify the `constraints` property for each of the new panels to control their placement as follows:

- a Select `panel1` in the component tree or in the UI designer. (When a component is selected in the designer, small nibs show at each corner of the component.) Click the column to the right of the `constraints` property in the Inspector, and choose North from the drop-down list.

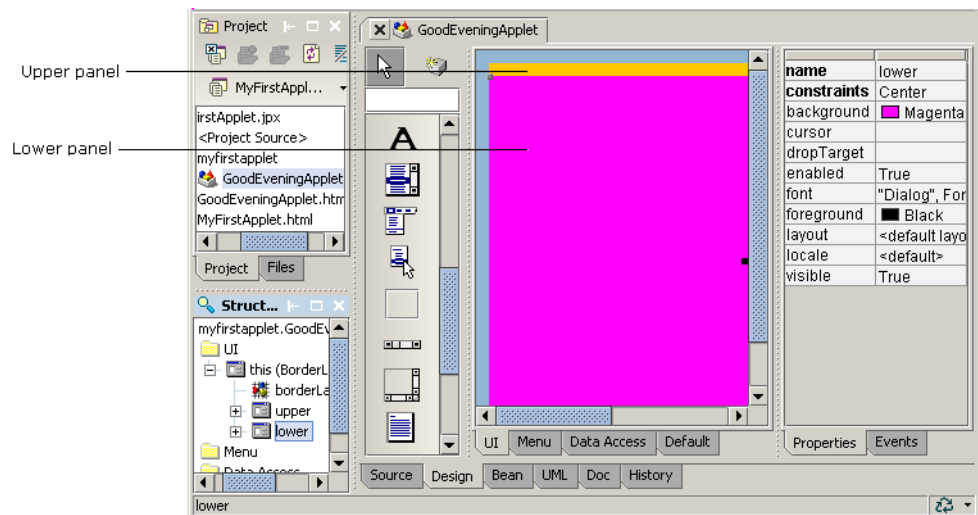
- b Now repeat the process for `panel2`, setting it to Center.

- 6 Select `panel1` again, and in the Inspector, select the `name` property. Highlight the value `panel1` in the right column, and replace it by typing `upper`.



Tip You can also rename a component by right-clicking a component in the component tree and selecting Rename from the menu.

- 7 Rename the bottom panel to `lower`.
- 8 Change the background color of the `upper` panel to Orange in the Inspector:
 - a Select the `upper` panel in the component tree or the designer.
 - b Click the column to the right of the `background` property in the Inspector.
 - c Click the *Down arrow* to open the color drop-down list and choose Orange from the list.
- 9 Change the background color of `lower` to Magenta. Now your UI should look like this:



- 10 Select the `lower` panel and change its layout to `CardLayout` in the Inspector.

The `CardLayout` panel will contain 5 panels, each with “Good Evening” in a different language.

`CardLayout` places components (usually panels) on top of each other in a stack like a deck of cards. You see only one at a time, and you can flip through the panels by using another control to select which panel comes to the top. `CardLayout` is usually associated with a controlling component, such as a check box or a list. The state of the controlling component determines which component the `CardLayout` displays. The user makes the choice by selecting something on the UI.

- 11 Add five panels (`panel1` through `panel5`) to the `lower` panel. To ensure all of them get added to the correct panel, click on the `lower` panel node in the component tree rather than on the design surface.

Note If you drop the component in the wrong location, select it in the component tree and press *Delete*. Then add it again.

- 12 Change the layout for `panel1` through `panel5` to `BorderLayout`. You can do this for all the panels at the same time:

Use *Shift+Click* or *Ctrl+Click* to select all five panels in the component tree, then change the `layout` property to `BorderLayout` in the Inspector.

- 13 Select another component in the component tree or the UI designer to deselect all the panels.

- 14 Change the background color of each of these 5 panels to a different color in the Inspector. The color selection is up to you.

Tip Click the column to the right of the `background` property in the Inspector, and click the ellipsis button to the right of the drop-down list to open the Background dialog box. Use the color sliders in the Background dialog box to create a color.

15 Save the file and the project.

16 Right-click `GoodEveningApplet.html` and choose Run using “GoodEveningApplet.”

When the applet runs in Sun’s appletviewer, you’ll only see `upper` and the top panel of the `CardLayout`. The other language panels display later after you add the drop-down list and add the events to the list selections.

17 Exit the applet.

18 Close the message pane by right-clicking the `GoodEveningApplet` tab and choosing Remove “GoodEveningApplet” Tab.

Step 5: Adding AWT components to your applet

Now you’ll use the component palette to add a `Label` and a `Choice` component to the `upper` panel in your design.



1 Click the `Choice` component on the AWT page of the palette.

2 Drop the component into your design’s top orange panel, `upper`. Use one of the following two methods:

- Click the `upper` panel in the component tree.
- Click the `upper` panel in the UI designer.

Note that `choice1` is added to `upper` in the component tree.



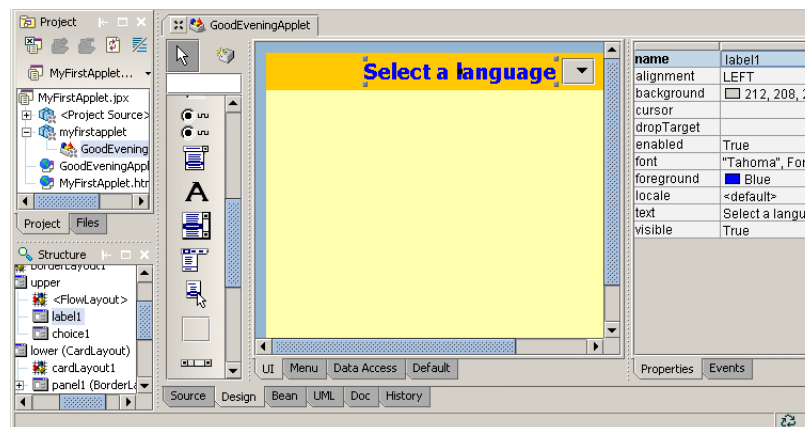
3 Select the `Label` component on the AWT page of the component palette and place it on the `upper` panel to the left of the `Choice` component.

Note that `label1` is added to `upper` in the component tree.

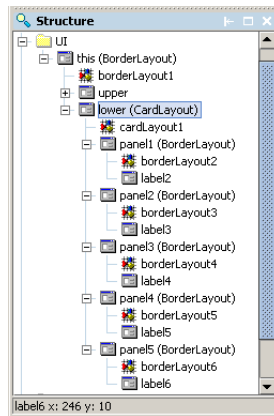
4 Select `label1` in the component tree and complete the following steps:

- a Double-click the column to the right of the `text` property in the Inspector to highlight the existing text.
- b Enter `Select a language` and press *Enter*. “Select a language” now appears in the label next to the `Choice` component.
- c Click the column to the right of the `font` property to set the font. Click the ellipsis button to open the Font dialog box.
- d Check **Bold**. Enter `20` in the `Size` box, then click **OK**.
- e Click the column to the right of the `foreground` property in the Inspector to set the text color. Click the *Down arrow* and select `Blue` from the drop-down list of colors.

Your design should look similar to this (except for the colors):



- 5 Now add an AWT label component to each panel node in the `lower CardLayout` panel (`panel1` through `panel5`). To make this process faster, after you click the `label` on the palette, just click on the target `panel` node in the component tree to drop the label.



Each of these labels is going to display “Good Evening” in a different language.

- 6 Change each label to “Good Evening” in a different language. First, select the label under each panel in the component tree and enter “Good Evening” in the appropriate language in the `text` property of the Inspector. Use these languages for the labels or choose your own:
- `label2`: Good evening! (English)
 - `label3`: Guten abend! (German)
 - `label4`: Buona sera! (Italian)
 - `label5`: Bonsoir ! (French)
 - `label6`: Goede avond! (Dutch)
- 7 Select `label2` through `label6` using **Ctrl+Click** and change the `font` properties for all the labels to **Bold** and the font size to 24. Leave the foreground property of the labels (the text color) **Black**. Click any component in the component tree or in the UI designer to deselect the labels.
- 8 Change the position of each label by changing its `constraints` property in the Inspector to **North**, **South**, **East**, **West**, or **Center** as follows:
- `label2`: **North**
 - `label3`: **South**
 - `label4`: **East**
 - `label5`: **West**
 - `label6`: **Center**

Note Notice the position of each label (as indicated by the selection nibs) in its `BorderLayout` panel. **Center** fills the entire panel, although the text is on the left because the alignment is left-justified, while **North**, **South**, **East**, and **West** fill only a portion of the panel.

- 9 Now select `panel1` through `panel5`, and change their layout to `FlowLayout`, which is actually the default layout for an AWT `Panel`.

Notice that the labels move to the center top of the panels. `FlowLayout` is a good layout choice when you want your components in rows.

Next, you'll add a button to one of these panels, which `FlowLayout` will place in the same row as the label. `FlowLayout` arranges components from left to right in a row. When the row is full, the remaining components move to a new row. This button will ultimately change the color of the `label` text on this panel when it's pushed.



- 10 Click the AWT `Button` component on the palette, and drop it into `panel2` to the left of the “Guten abend!” label.

The panel should look similar to this:



Try moving the button to another position in the designer. `FlowLayout` always forces it back into the row with the label, either before or after the label.

- 11 Choose `File|Save All` to save the project.

Step 6: Editing your source code

In this step, you'll add the languages to the `Choice` drop-down list, then add events to hook up each language panel to the `Choice` component.

To create the items for the drop-down list,

- 1 Click the Source tab in the content pane to change to the source code in the editor.
- 2 Select the `init()` method in the structure pane. The `init()` method code is highlighted in the editor.

Tip Search for a method in the structure pane by clicking in the structure pane and typing the method name.

- 3 Position the cursor after the opening curly brace and before the `try/catch` statement and press *Enter* to create an extra blank line.

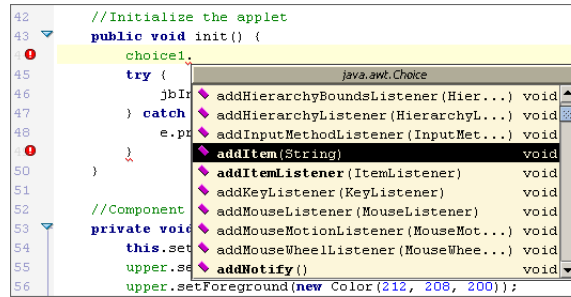
Tip To expand the editor and hide the project and structure pane, select `View|Show All`.

- 4 Add the following code block indicated in bold to the `init()` method:

```
//Initialize the applet
public void init() {
    choice1.addItem("English");
    choice1.addItem("German");
    choice1.addItem("Italian");
    choice1.addItem("French");
    choice1.addItem("Dutch");

    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

Tip Use CodeInsight to complete the code for you. Enter `choice1.` and wait for the pop-up window or press `Ctrl+Spacebar` to invoke it. Be sure to include the dot (.) after `choice1.` Use the arrow keys to select `addItem(String)` from the pop-up window. Press `Enter`. You can configure CodeInsight in the Preferences dialog box (Tools | Preferences | Editor | CodeInsight).



If there are any syntax errors in your code, an **Errors** folder appears in the structure pane as you type in the editor. Open the folder and select the error message to highlight the error in the source code.

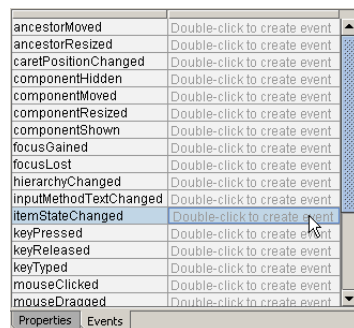
See also

- “About error and warning messages” in *Building Applications with JBuilder* in online help

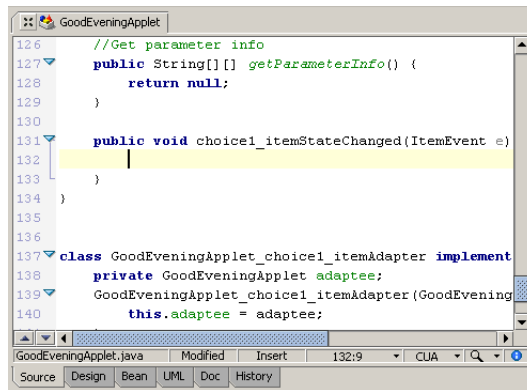
Next, you’ll hook up the events to the choice items. When you select a language from the drop-down list `Choice` component, “Good Evening” appears in the `cardLayout` panel in the selected language.

To hook up the `Choice` list events,

- 1 Return to the UI designer.
- 2 Select `choice1` located under `upper` in the component tree.
- 3 Select the Events tab in the Inspector.
- 4 Double-click to the right of the `itemStateChanged` event.



JBuilder generates the method code and takes you to the source code with the cursor inserted in the method.



- 5 Add the following code indicated in bold to the body of this new event. This connects the correct language panel to the language choice.

```

void choice1_itemStateChanged(ItemEvent e) {
    if ("English".equals(choice1.getSelectedItem())){
        cardLayout1.show(lower, "panel1");
    }
    else if ("German".equals(choice1.getSelectedItem())){
        cardLayout1.show(lower, "panel2");
    }
    else if ("Italian".equals(choice1.getSelectedItem())){
        cardLayout1.show(lower, "panel3");
    }
    else if ("French".equals(choice1.getSelectedItem())){
        cardLayout1.show(lower, "panel4");
    }
    else if ("Dutch".equals(choice1.getSelectedItem())){
        cardLayout1.show(lower, "panel5");
    }
}
  
```

Tip You can use code templates to generate code. Type `if` and press `Ctrl+J` to access the code templates pop-up window. Use the arrow keys to navigate the selections. Select the `if-else if` template and press `Enter`. The code is generated:

```

if () {

}
else if{

}
  
```

- 6 Choose `File|Save All`.

- 7 Run the applet by right-clicking `GoodEveningApplet.html` in the project pane and selecting Run using “GoodEveningApplet.”

The “Good Evening” applet runs in Sun’s appletviewer:



If there are any errors, they appear in the message pane at the bottom of the JBuilder’s IDE. Select an error message and press *F1* for Help. Select the error message to highlight the code in the editor. Sometimes the error may be before or after the highlighted line of code. Fix the errors, save the project, and run the applet again.

- 8 Test the drop-down list. The language selected from the list should match the language on the panel below it.
- 9 Exit the applet.

Now, add a button label and a button event for `button1` on `panel2`. When you push the button, the “Guten abend!” text on `label6` change will change to yellow.

- 1 Add a button label and a button event as follows:

- a Switch to the UI designer again, and select `button1` on `panel2`. Change the button’s `label` property on the Properties page of the Inspector from `button1` to Push Me. Press *Enter*. Notice how the button automatically resizes to fit the text.
- b Click the Events tab for the button to define what happens when `button1` is pressed.
- c Double-click the column to the right of the `ActionPerformed` event.

JBuilder switches to the editor where the following skeleton code has been added for the `ActionPerformed` event just below the if-else if statements.

```
void button1_actionPerformed(ActionEvent e) {

}
```

Tip

Double-clicking the button in the designer has the same effect.

Now, enter the code that defines the button event which will change “Guten abend!” to yellow.

- d Type the following code indicated in bold:

```
void button1_actionPerformed(ActionEvent e) {
    label13.setForeground(new Color(255,255,0));
}
```

- 2 Save the project.

- 3 Run the applet and select "German" from the drop-down list. Click the "Push Me" button.

"Guten abend!" should change to yellow.

Your applet should look similar to this:



- 4 Exit the applet.

Step 7: Deploying your applet

Deploying a Java applet consists of bundling together the various Java class files, image files, and other files needed by your applet and copying them and the applet HTML file to a location on a server or client computer where they can be executed. You can deliver the files separately, or you can deliver them in compressed or uncompressed archive files. JAR files, Java archive files, are the most commonly used. JAR files provide the advantages of smaller file sizes and faster download times.

When deploying your applet, it's important to remember the following:

- Save and compile your project before deploying.
- Open `GoodEveningApplet.html` and click the Source tab to review the following:
 - Check that the `codebase` attribute specifies the correct location of the class file relative to the HTML file. In this example, the `codebase` value is `"."`, because the JAR file containing the class file will be in the same directory as the HTML file.
 - Check that the `code` attribute has the fully qualified class name, including the package name. In this example, the `code` value is `firstapplet.GoodEveningApplet.class`.

The applet tag should look like this:

```
<applet
  codebase = "."
  code     = "myfirstapplet.GoodEveningApplet.class"
  name     = "TestApplet"
  width    = "400"
  height   = "300"
  hspace   = "0"
  vspace   = "0"
  align    = "middle"
>
</applet>
```


- Maintain the existing directory structure. In this example, `GoodEveningApplet.class` must be in a `myfirstapplet` directory to reflect the package structure: `myfirstapplet/GoodEveningApplet.class`. If you're deploying to a JAR file, check the directory structure in the file and make sure it matches.
- Deliver all the necessary classes to the appropriate Internet server. The class files must be in the correct location relative to the HTML file and matching the `codebase` attribute.
- Deliver the applet HTML file to the appropriate Internet server.

Caution If you are creating your applets for older JDK 1.02-compliant browsers, keep in mind that JAR files are not supported by these older browsers. Create a ZIP archive file instead.

See also

- “Deploying Java programs” in *Building Applications with JBuilder*
- “Deploying the Text Editor application” in *Designing Applications with JBuilder* in online help

Deploying your applet with the Archive Builder

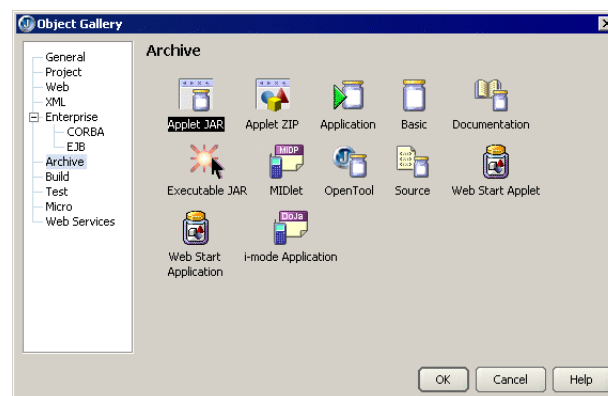
JBuilder's Archive Builder collects all the files needed to distribute your applet and can archive them into a JAR file.

To deploy your applet:

- 1 Save your project.
- 2 Choose Project|Make Project “MyFirstApplet.jpx” to compile your project.
- 3 Create an `applets` directory for your applet in your `/jbproject` directory.

This will be the testing directory where you'll put your applet HTML file and your JAR file.

- 4 Choose File|New and select the Archive page.

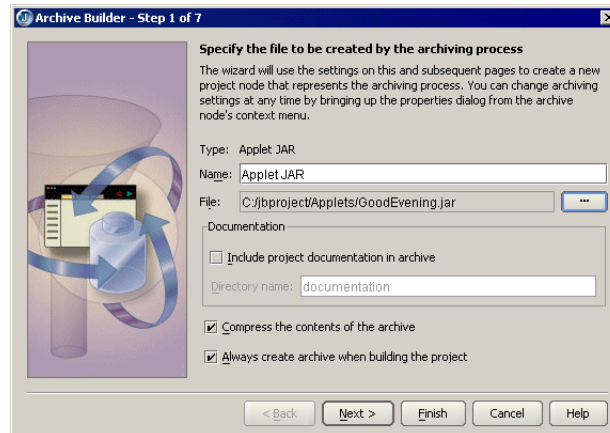


- 5 Double-click the Applet JAR icon to open the Archive Builder wizard.

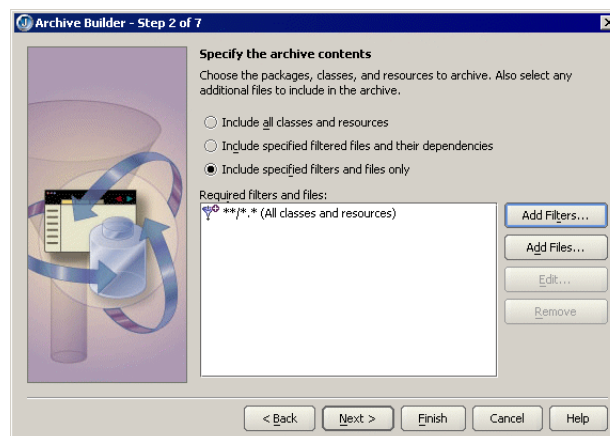
Note Select the Applet ZIP archive type if you are developing applets for older browsers that do not support JAR files.

- 6 Accept the default name of Applet JAR in the Name field on the first step of the wizard.
- 7 Click the ellipsis button next to the File field and browse to the `/<home>/jbproject/applets/` directory.
- 8 Change the JAR file name in the File field to `GoodEvening.jar` and click OK.

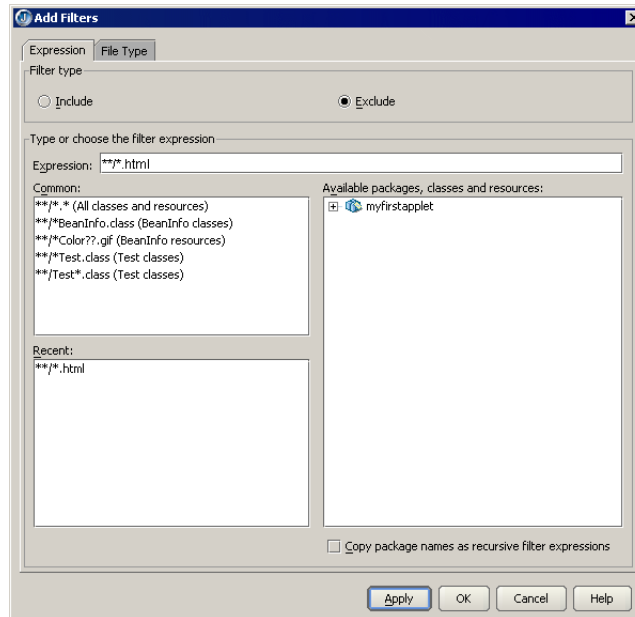
Step 1 should look like this:



- 9 Click Next to go to step 2.



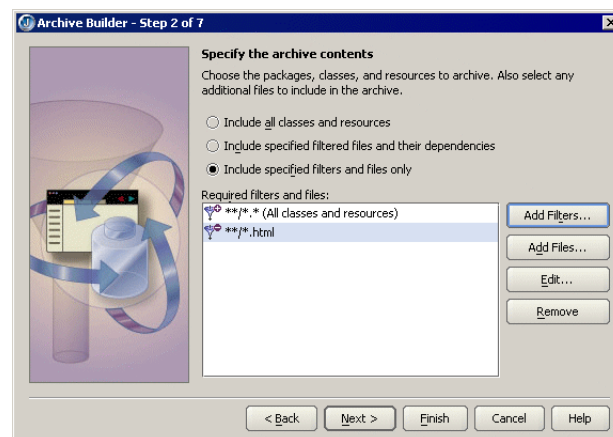
- 10 Click Add Filters on step 2 and make the following changes on the Expression page of the Add Filters dialog box:
 - a Click the Exclude radio button in the Filter Type area.
 - b Enter `**/*.html` in the Expression field located in the Type Or Choose The Filter Expression area.



- 11 Click Apply, then OK.

The Archive Builder adds the `**/*.html` entry to the Required Filters And Files list box, preceded by a minus sign (-). This filter will exclude the applet HTML file, `GoodEveningApplet.html`, from the JAR file. Later, you will copy the `GoodEveningApplet.html` file to the `applets` directory.

Step 2 should look like this:



- 12 Click Finish to accept the defaults on the remaining steps and exit the wizard. The Archive builder will create the archive JAR and manifest files.

Note

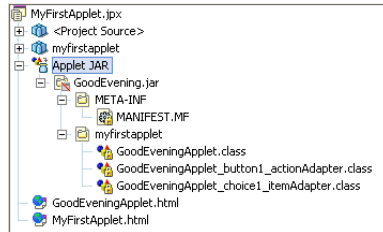
For information on manifest files, see “About the manifest file” in *Building Applications with JBuilder*.

An archive node called `Applet JAR` appears in the project pane. You can modify this file by right-clicking it and selecting Properties.

- 13 Right-click the `Applet JAR` archive node and choose **Make** to compile your project.

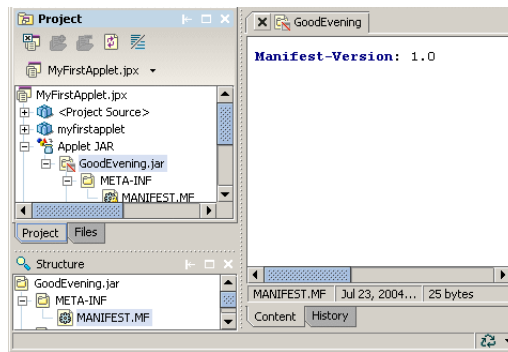
The Archive Builder gathers the `myfirstapplet` package into the JAR file.

- 14 Click the expand icon next to the `Applet JAR` archive node to see the `GoodEvening.jar` archive file.



- 15 Double-click the JAR file in the project pane.

The manifest file appears in the content pane and the contents of the JAR file appear in the structure pane. Select a file in the structure pane to view it in the content pane.



Note: If you are delivering multiple programs to the same location, you can deliver the redistributable files separately, rather than include them in each of your JAR files.

See also

- “Using the Archive Builder” in *Building Applications with JBuilder*
- “Deploying Java programs” in *Building Applications with JBuilder*

Step 8: Modifying the HTML file

Now that your applet is deployed in a JAR file, you need to modify the HTML file with the `archive` attribute and include the JAR file name. You'll also add a message inside the `<applet>` tags that tells users without Java-enabled browsers that they won't be able to see the applet unless they enable Java in their browser or upgrade their browser.

To modify the HTML file,

- 1 Open `GoodEveningApplet.html` in JBuilder and add the `archive` attribute as follows:
 - a Select the **Source** tab to view the HTML source code.
 - b Add the following HTML code inside the `<applet>` tag:

```
archive = "GoodEvening.jar"
```

The `<applet>` tag should look like this:

```
<applet
  codebase = "."
  code     = "myfirstapplet.GoodEveningApplet.class"
  archive  = "GoodEvening.jar"
  name     = "TestApplet"
  width    = 400
  height   = 300
  hspace   = 0
  vspace   = 0
  align    = top
>
</applet>
```

Tip If you have multiple JAR files for your applet, list them separated by a comma as shown here:

```
archive="file1.jar, file2.jar"
```

Important Some older web browsers do not support JAR files and multiple listings of archive files but do support a single ZIP file in the `archive` attribute.

Next, add a message that tells users without Java-enabled browsers that their browsers do not support Java; therefore, they can't see the applet.

2 Enter the following message between the open and close `<applet>` tags:

```
You need a Java-enabled browser running JDK 1.1.x or greater to view this
applet.
```

The `<applet>` tag looks like this:

```
<applet
  codebase = "."
  code     = "myfirstapplet.GoodEveningApplet.class"
  archive  = "GoodEvening.jar"
  name     = "TestApplet"
  width    = 400
  height   = 300
  hspace   = 0
  vspace   = 0
  align    = top
>
<strong>You need a Java-enabled browser running JDK 1.1.x or greater to view
this applet.</strong>
</applet>
```

Any web browser that does not support Java ignores the `<applet>` tags and displays everything between the tags. Because a Java-enabled browser recognizes the `<applet>` tags, anyone with a Java-enabled browser will see the applet and not the message.

Important Before saving the HTML file, check the `codebase` and `code` values again. If these values are incorrect, the applet won't run. Remember that the `codebase` value is the location of the applet code (CLASS or JAR file) in relation to the HTML file. The value, ".", means the class file is in the same directory as the HTML file. The `code` value must be the fully qualified class name for the applet, including the package name.

3 Save and close the file.

4 Copy the modified `GoodEveningApplet.html` from the project's `classes` directory to the `applets` directory. The `applets` directory should contain two files, `GoodEveningApplet.html` and `GoodEvening.jar`.

Caution Remember, JBuilder creates two HTML files: the project notes HTML file, `FirstApplet.html`, located at the root of the project directory and the applet HTML file containing the `<applet>` tag, `GoodEveningApplet.html`, located in the project `src` directory. Do not copy `FirstApplet.html` instead of `GoodEveningApplet.html` to the `applets` directory or your applet will not run.

Step 9: Running your deployed applet from the command line

It's a good idea to test the deployed applet locally before testing it on the Web. You can do this from the command line using Sun's `appletviewer`. This tells you if the web browser has everything it needs to run the applet. If any files are missing or if there are any errors in the HTML file, the applet won't run. You can then correct the errors before posting it on the Web.

To run the applet at the command line,

- 1 Be sure copies of `GoodEveningApplet.html` and `GoodEvening.jar` are in the applets directory.
- 2 Open the command-line window.
- 3 Clear any `CLASSPATH` variables to remove any class path settings for this session as follows:
 - Windows NT, 2000, and XP: `set CLASSPATH=`
 - UNIX:
 - in csh shell: `unsetenv CLASSPATH`
 - in sh shell: `unset CLASSPATH`
- 4 Change to the `applets/` directory.
- 5 Run the `appletviewer` by entering the following command:

```
/<jbuilder>/<jdk>/bin/appletviewer GoodEveningApplet.html
```

Where `<jbuilder>` represents the name of the JBuilder installation directory, `/jbuilder<version>/`.

Important

If JBuilder is on a different drive than the applet, include the drive letter.

Note

For Windows, use backslashes (`\`).

If the "Good Evening" applet loads and runs in the `appletviewer`, the deployment was successful and all the classes were found and included. If the applet doesn't run, check the error messages, correct them, recompile, deploy, and test again.

If you are having problems running your applet, check ["Applet source code" on page 245](#) and see these topics for common errors:

- "Solving common applet problems" at <http://www.java.sun.com/docs/books/tutorial/applet/problems/index.html>
- "Common mistakes in the `<applet>` tag" and "Additional tips for making applets work" in "Working with applets" in the *Developing Web Applications*

The final step in developing your applet is to test it by running it on the Web. This tells you if it really has all the files it needs. Although this tutorial doesn't go into details on applet testing, see ["Testing your deployed applet on the Web" on page 245](#) for an overview of applet testing.

Congratulations!! You've created your first applet with JBuilder. Now that you're familiar with JBuilder's development environment, you'll find its many time-saving features make your programming easier.

For other applet tutorials, see

- “The Java Tutorial” at <http://java.sun.com/docs/books/tutorial/index.html>
- Charlie Calvert’s Part II: Applets at <http://homepages.borland.com/ccalvert/JavaCourse/index.htm>
- John Moore’s “Applet design and deployment” at http://www.microps.com/mps/p_appletdesign.html

Testing your deployed applet on the Web

The final step in developing your applet is to run it on the Web. This tells you if it really has all the files it needs. It’s important to test your applet in a variety of browsers to be sure that the necessary files are included in the archive. Although this tutorial doesn’t go into details on applet testing, here are a few testing guidelines.

Complete these steps, then test your applet by running it on the Web in various browsers:

- 1 Transfer the applet’s HTML and JAR files to an Internet server or copy them to a Windows NT server.
 - a Use an FTP (file transfer protocol) utility to transfer the files to the server. Be sure to transfer the files as binary files.
 - b Verify that the HTML and JAR file locations on the server match the `codebase` attribute in the HTML file and that the `code` attribute has the fully qualified class name (including the package name).
- 2 Test the applet in various browsers. If the applet fails to load, check that the browser is Java-enabled. Also check the browser’s Java Console for error messages.

To open the Java Console,

- Select Communicator|Tools|Java Console in Netscape.

Java Console availability and access varies by browser version.

- 3 Correct any errors, redeploy your applet, and test again in the browsers.

See also

- “Testing applets” in *Developing Web Applications*

Applet source code

Select a link to view the Applet HTML source code or the Applet class source code.

- Applet HTML source code for “[Applet HTML source code for GoodEveningApplet.html](#)” on page 246
- Applet class source code for “[Applet class source code for GoodEveningApplet.java](#)” on page 246

Applet HTML source code for GoodEveningApplet.html

```
<html>
<head>
<title>
Good Evening HTML Test Page
</title>
</head>
<body>
myfirstapplet.GoodEveningApplet will appear below in a Java enabled browser.<br>
<applet
  codebase = "."
  code      = "myfirstapplet.GoodEveningApplet.class"
  archive   = "GoodEvening.jar"
  name      = "TestApplet"
  width     = "400"
  height    = "300"
  hspace    = "0"
  vspace    = "0"
  align     = "middle"
>
<strong>You need a Java-enabled browser running JDK 1.1.x or greater to view this
  applet.</strong>
</applet>
</body>
</html>
```

Applet class source code for GoodEveningApplet.java

```
package myfirstapplet;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class GoodEveningApplet extends Applet {
  private boolean isStandalone = false;
  BorderLayout BorderLayout1 = new BorderLayout();
  Panel upper = new Panel();
  Panel lower = new Panel();
  CardLayout cardLayout1 = new CardLayout();
  Panel panel1 = new Panel();
  Panel panel2 = new Panel();
  Panel panel3 = new Panel();
  Panel panel4 = new Panel();
  Panel panel5 = new Panel();
  Choice choice1 = new Choice();
  Label label1 = new Label();
  Label label2 = new Label();

  Label label3 = new Label(); //Get a parameter value

  Label label4 = new Label();
  Label label5 = new Label();
  Label label6 = new Label();
  FlowLayout flowLayout1 = new FlowLayout();
  FlowLayout flowLayout2 = new FlowLayout();
  FlowLayout flowLayout3 = new FlowLayout();
  FlowLayout flowLayout4 = new FlowLayout();
  FlowLayout flowLayout5 = new FlowLayout();
  Button button1 = new Button();
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) : def);
  }
}
```



```

//Construct the applet
public GoodEveningApplet() {
}

//Initialize the applet
public void init() {
    choice1.addItem("English");
    choice1.addItem("German");
    choice1.addItem("Italian");
    choice1.addItem("French");
    choice1.addItem("Dutch");

    try {
        jbInit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

//Component initialization
private void jbInit() throws Exception {
    this.setLayout(borderLayout1);
    upper.setBackground(Color.orange);
    upper.setForeground(new Color(212, 208, 200));
    lower.setBackground(Color.magenta);
    lower.setLayout(cardLayout1);
    panel1.setLayout(flowLayout1);
    panel2.setLayout(flowLayout2);
    panel3.setLayout(flowLayout3);
    panel4.setLayout(flowLayout4);
    panel5.setLayout(flowLayout5);
    panel1.setBackground(new Color(255, 255, 173));
    panel2.setBackground(new Color(192, 0, 192));
    panel3.setBackground(new Color(212, 208, 255));
    panel4.setBackground(new Color(206, 255, 255));
    panel5.setBackground(new Color(255, 206, 255));
    label1.setFont(new java.awt.Font("Tahoma", Font.BOLD, 20));
    label1.setForeground(Color.blue);
    label1.setLocale(java.util.Locale.getDefault());
    label1.setText("Select a language");
    label2.setFont(new java.awt.Font("Tahoma", Font.BOLD, 24));
    label2.setText("Good evening!");
    label3.setFont(new java.awt.Font("Tahoma", Font.BOLD, 24));
    label3.setText("Guten abend!");
    label4.setFont(new java.awt.Font("Tahoma", Font.BOLD, 24));
    label4.setText("Buona sera!");
    label5.setAlignment(Label.LEFT);
    label5.setFont(new java.awt.Font("Tahoma", Font.BOLD, 24));
    label5.setText("Bonsoir!");
    label6.setAlignment(Label.LEFT);
    label6.setFont(new java.awt.Font("Tahoma", Font.BOLD, 24));
    label6.setText("Goede avond!");
    button1.setLabel("Push me");
    button1.setLocale(java.util.Locale.getDefault());
    button1.addActionListener(new GoodEveningApplet_button1_actionAdapter(this));
    choice1.addItemListener(new GoodEveningApplet_choice1_itemAdapter(this));
    choice1.setFont(new java.awt.Font("Arial", Font.PLAIN, 12));
    choice1.setForeground(Color.black);
    this.add(upper, java.awt.BorderLayout.NORTH);
    this.add(lower, java.awt.BorderLayout.CENTER);
    lower.add(panel1, "panel1");
    lower.add(panel2, "panel2");
    lower.add(panel3, "panel3");
    lower.add(panel4, "panel4");
    lower.add(panel5, "panel5");
    upper.add(label1);
    upper.add(choice1);
    panel1.add(label2, null);
    panel2.add(label3, null);
    panel3.add(label4, null);

```

```

        panel4.add(label5, null);
        panel5.add(label6, null);
        panel2.add(button1);
    }

    //Start the applet
    public void start() {
    }

    //Stop the applet
    public void stop() {
    }

    //Destroy the applet
    public void destroy() {
    }

    //Get Applet information
    public String getAppletInfo() {
        return "Applet Information";
    }

    //Get parameter info
    public String[][] getParameterInfo() {
        return null;
    }

    public void choice1_itemStateChanged(ItemEvent e) {
        if ("English".equals(choice1.getSelectedItem())) {
            cardLayout1.show(lower, "panel1");
        } else if ("German".equals(choice1.getSelectedItem())) {
            cardLayout1.show(lower, "panel2");
        } else if ("Italian".equals(choice1.getSelectedItem())) {
            cardLayout1.show(lower, "panel3");
        } else if ("French".equals(choice1.getSelectedItem())) {
            cardLayout1.show(lower, "panel4");
        } else if ("Dutch".equals(choice1.getSelectedItem())) {
            cardLayout1.show(lower, "panel5");
        }
    }

    public void button1_actionPerformed(ActionEvent e) {
        label3.setForeground(new Color(255,255,0));
    }
}

class GoodEveningApplet_choice1_itemAdapter implements ItemListener {
    private GoodEveningApplet adaptee;
    GoodEveningApplet_choice1_itemAdapter(GoodEveningApplet adaptee) {
        this.adaptee = adaptee;
    }

    public void itemStateChanged(ItemEvent e) {
        adaptee.choice1_itemStateChanged(e);
    }
}

class GoodEveningApplet_button1_actionAdapter implements ActionListener {
    private GoodEveningApplet adaptee;
    GoodEveningApplet_button1_actionAdapter(GoodEveningApplet adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e) {
        adaptee.button1_actionPerformed(e);
    }
}

```

Index

Symbols

@todo tags 125

A

accessing code folding 135
accessing keyboard 42
actions, editor settings 109
active process 22
adaptive navigating, keyboard 42
adding
 characters to text blocks 111
 favorites 65
 import statements 112, 140
 macros, code template 140
 template macros 140
 todo tags 124
applets tutorial 219
applications tutorial 191
applying filters to packages 69
archiving runnable application 100
audits 60
auto imports for templates 138
 source code templates 138

B

bookmarks
 adding bookmarks 125
 editing 126
 editor 125
 Help Viewer 39
 navigating to 126
 viewing 126
books, JBuilder 29
Borland
 Community website 43
 contacting 43
 developer support 43
 e-mail 44
 newsgroups 44
 online resources 43
 reporting bugs 44
 technical support 43
 World Wide Web 44
braces, wrapping 116
browser
 hiding panes 69
 resizing content pane 69
 status bars 81
browsing symbol definitions 119
bugs, reporting 44
building projects 52, 84
 adding files 52

C

CaliberRM 91
changing 115
changing icons 77
class browser 74
class reference documentation 41

classes pane 74
 displaying 74
 hierarchy 74
 navigation 75
classes tab, project pane 17
ClassInsight 131
code
 applying formatting 115
 audits 60
 automating 57
 CodeInsight 130
 completing automatically 131
 creating in editor 107
 drilling into 78
 finding definition of 119
 finding references to 120
 formatting 113
 keyboard shortcuts 130
 shortcuts 129
code blocks, wrapping braces around 116
code errors 136
 ErrorInsight 136
code folding 134
 key bindings 134
 options 135
code formatting, changing 115
code navigation 134
 code folding 134
 Scopelnsight 133
code scoping 133
code shortcuts
 ClassInsight 131
 code folding 134
 code templates 138
 CodeInsight 130
 ErrorInsight 136
 JavadocInsight 122
 MemberInsight 131
 object gallery 58
 ParameterInsight 132
 Scopelnsight 133
 Smart MemberInsight 131
 SyncEdit 143
 TagInsight 145
 wizards 57
code symbols 119
 finding definition of 119
 finding references to 120
code templates 138
 adding to code 139
 auto imports 140
 creating 138, 140
 editing 142
 insert template macros 140
 navigating using tabs 143
 surrounding 141
CodeInsight 130
 configuring 133
 failing 131
 keyboard shortcuts 130
coding 57, 107
color customizing preferences 115

- comments, Javadoc shortcuts 121
- compiling projects 83
- component palette for tags 147
- configuring 167
 - IDE panes 167
 - See also* workspace configurations
 - JBuilder CodeInsight 133
- conflicts, resolving Javadoc 123
- content pane 14, 67
 - file tabs 68
 - file view tabs 14
 - hiding other panes 69
 - resizing 69
- content pane tabs, positioning 159, 171
- copying
 - from the Help Viewer 37
 - keymaps 154
 - text, message pane 80
- creating new projects 52
- curly braces, wrapping 116
- customizing 173
 - editor 160
 - editor display 156
 - IDE 173
 - JBuilder import settings 176
 - markup language editing 149
 - markup language formatting 149
 - markup language tags 149
 - screen elements 115
 - structure pane icons 77
 - SyncEdit elements 144
 - TagInsight 148
 - the IDE import settings 176

D

- dead code 86
- debugging 132
 - programs 87
- decorated files 70
- decorations 70
- decreasing code text size 112
- default project properties 179
- deleting
 - characters from text blocks 111
 - directories 54
 - files 54
- designer types, column designer 59
- designer, visual 59
- Developer Support 43
- directories, deleting 54
- displaying whitespace characters 158
- docking panes 170
- documentation 29
 - JBuilder 29
- documentation conventions 31
 - platform conventions 32
- drilling down 79
- drilling down in file structure 75, 78
- Dynamic Help 41

E

- editing
 - code templates 142
 - keymaps 152
 - SyncEdit 143
 - workspace configurations 169
- editor
 - actions 109, 156
 - adding characters to text blocks 111
 - bookmarks 125
 - code templates 138
 - color options 115
 - context menus 108
 - customizing 156, 160
 - deleting characters from text blocks 111
 - display 156
 - displaying line numbers 158
 - emulations 152
 - file tabs 108
 - gutter 108
 - Javadoc tips 121
 - JavadocInsight 122
 - keymaps 152
 - limiting open files 159
 - printing source code 128
 - searching 117
 - selecting text 110
 - selecting text blocks 111
 - settings 109
 - splitting view 157
 - text features 109
 - todo tags 124
 - using 107
 - whitespace characters 158
 - working with text 109
- editor, finding
 - code elements 119
 - definition of symbols 119
 - definitions 119
 - overridden methods 120
 - overriding methods 121
 - references to symbols 120
 - superclasses 120
 - unused variables 121
- enlarging code text 112
- error messages, viewing and finding 78
- ErrorInsight 136
- errors 136
 - ErrorInsight 136
 - solutions 136
- exporting keymaps 154
- ExpressionInsight 132
- expressions 132
 - auto-evaluating 132
 - ExpressionInsight 132

F

- favorite links
 - adding 65
 - organizing 65
- file 70

- file tabs, context menus 108
- file views 14
- files 52
 - adding to project 52
 - creating new empty 52
 - deleting 54
 - modified 70
 - removing 54
 - renaming 54
 - viewing 71
 - viewing multiple 171
 - viewing structure 75
- files pane 73
 - tool tips 70
- files tab, project pane 17
- filtering, structure pane 76
- finding 120
 - keymap actions 155
 - overridden method 120
 - references to 120
 - superclass 120
 - text 117
- finding text in Help 36
 - current topic 37
 - full-text search 36
- finding text, find/replace in path 118
- folding code
 - disabling 135
 - displaying tool tips 135
- fonts 31
 - JBuilder documentation conventions 31
 - resizing in editor 112
 - resizing in IDE 112
- formatting 113, 115
 - applying curly braces to code blocks 116
 - applying to code 113, 115
- formatting text, message pane 80
- free-floating message pane 81

G

- Go To Line 158
- gutter 108
 - context menus 108
- gutter margin, line numbers 158
- gutters, using to select text 110

H

- heap memory, monitoring 82
- Help
 - Dynamic Help 41
 - task-oriented 41
- help
 - embedded 35
 - for component in structure pane 79
 - standalone 35
 - using online 32
- Help Viewer
 - accessing 32
 - adding tabs 38
 - book subsets 37
 - bookmarks 39
 - copying text from 37
 - embedded 35

- finding text in current topic 37
- navigating in 40
- search 36
- search delimiters 37
- standalone 35
- synchronizing contents 36
- table of contents 35
- tasks tab 36
- toggling panes 38
- user interface 33
- using 34
- using index of 36
- using keyboard keys in 40
- zooming the view 38
- hiding message pane 81
- hierarchical views 75

I

- iconifying panes 170
- icons 108
- IDE 9
 - content pane 67
 - editor 107
 - elements 9
 - message pane 79
 - preferences 173
 - project pane 16, 69
 - structure pane 18, 75
 - toolbar 12
 - workspace configurations 167
- identifiers, finding definition 119
- identifying overriding methods 134
 - editor 121
 - Scopelnsight 134
- identifying unused variables 134
 - editor 121
- illegal syntax notification 130
- import options, ClassInsight 131
- importing settings 176
- increasing code text size 112
- inserting classes, ClassInsight 131
- integrated development environment (IDE) *See* IDE

J

- Java formatting 115
- Javadoc 121
 - adding tags 122
 - comments 121
 - custom tags 123
 - editing tags 122
 - folding comments 124
 - QuickHelp 124
 - resolving conflicts 123
 - todo comments 125
 - ToDo folder 77
 - todo tags 124
- Javadoc comments
 - conflicts 123
- Javadoc documentation
 - adding comments 121
 - See also* Javadoc, comments
 - shortcut commands 121
- Javadoc QuickHelp 124

- JavadocInsight 122
 - accessing 123
 - color options 123
 - pop-up timing 123

- JBuilder
 - customizing the editor 160
 - customizing the IDE 173
 - documentation for 29
 - introductory tour 43
 - menus 11
 - newsgroups 44
 - reporting bugs 44

K

- keyboard navigation 42
- keyboard shortcuts, code folding 134
- keymap actions, finding 155
- keymaps 152
 - binding new keystrokes 152
 - constructing new 153
 - copying 154
 - editing 152
 - editor emulations 152
 - exporting 154
 - printing 154
 - searching 155
- keystrokes 152
 - adding 152
 - binding new 152
 - changing 152
 - customizing 152

L

- library reference documentation 41
- line numbers 158
 - displaying in editor 158
 - go to specific line 158
 - selecting text 110
- locating text 117

M

- macros 140
 - adding to code templates 140
- managing workspace configurations 168
- manuals, JBuilder 29
- markup language
 - editing 149
 - formatting 149
 - tags 149
- MemberInsight 131
- memory
 - monitoring 82
 - problems 86
- menu bar 11
- menus 11
 - commands 11
 - JBuilder 11
 - toolbar shortcuts 12

- message pane 21, 79
 - copying text 80
 - docking 81
 - formatting text 80
 - hiding 81
 - showing 81
 - undocking 81
 - viewing todo comments 125
 - wrapping text 81
- modified files 70
- mouse, alternative navigation 42

N

- navigating
 - Help Viewer 40
 - without the mouse 42
- navigation, adding favorite links 65
- new keymaps 153
- newsgroups 44
 - Borland and JBuilder 44
 - public 44
 - Usenet 44

O

- object gallery 58
 - wizards 57
- Online help, accessing 32
- optimizing code 86
- organizing favorite links 65

P

- packages, adding to project 52
- pane
 - classes 74
 - files 73
 - structure 75
- panes
 - arranging 167
 - See also* workspace configurations
 - content 14
 - docking 170
 - iconifying 170
 - undocking 170
- parameter lists 132
- ParameterInsight 132
- pasting text 112
- positioning content pane tabs 159, 171
- preferences 173
 - color 115
 - JBuilder editor 160
- printing 128
 - keymaps 154
 - source code 128
- processes
 - active 22
 - starting 22
 - stopping 22
- profiling code 86
- project files, viewing structure 75

- project pane 16
 - classes pane 74
 - classes tab 17
 - context menu 18, 69
 - files tab 17
 - filtering packages 69
 - project tab 17
 - project toolbar 69
 - running an application 73
 - running applications 100
 - searching 69
 - tab for 17
 - tool tips 70
 - toolbar 70
- project properties, default 179
- Project wizard 52
- projects 23
 - adding files 52
 - adding Java source files 52
 - adding packages 52
 - building 84
 - closing 55
 - compiling 83
 - creating 52
 - managing requirements 91
 - opening 54
 - removing files and folders 54
 - renaming 54
 - viewing files 71
- properties, default project 179

R

- reference documentation 41
 - layout 41
- Release Notes 43
- reporting bugs 44
- requirements, managing projects 91
- resizing
 - code text 112
 - content pane 69
- resolving Javadoc conflicts 123
- runnable JAR 73, 100
- running
 - an application 73
 - archive application 100
 - processes 22
 - programs 85

S

- Scopelnsight 133
 - identifying unused variables 134
- screen elements, customizing 115
- search commands 117
- Search delimiters
 - Help Viewer 36, 37
- Search Help Viewer 36, 37
- Search menu 117
- searching
 - directories 117
 - find/replace in path 117
 - finding overridden method 120
 - finding superclass 120
 - in project pane 69

- in the editor 120
- keymap actions 155
- project, message, and structure panes 66
- text, find/replace in path 118
- using Find Definition 119
- with Find References 120
- Section 508 compliance 43
- selecting text
 - blocks 111
 - using line numbers 110
- settings, importing 176
- shortcut commands 129
 - coding 129
 - for Javadoc 121
- showing message pane 81
- smart key settings 109
- Smart MemberInsight 131
- source pane, splitting view 157
- splitter bar, browser 69
- starting processes 22
- status bar
 - changing editor font size 82
 - changing keymaps 82
 - displaying line numbers 82
 - navigating to a line 82
- status bars 22
 - browser 81
- stopping processes 22
- strings, finding 118
- structure pane 18, 75
 - error messages 78
 - filtering 76
 - filters 76
 - help 79
 - icons 77
 - navigating file structure 75
 - sorting order 76
 - ToDo folder 77
 - using to navigate source code 78
- SyncEdit 143
 - customizing 144
 - editing 143
- synchronizing contents with open Help file 36
- syntax errors, viewing and finding 78

T

- tabs 68
 - content pane 14, 68
 - template blocks 143
- tag errors
 - ErrorInsight 136
 - TagInsight 148
- tag inspector 147
- tagging
 - component palette 147
 - HTML, XML, JSP 147
- TagInsight 145
 - component palette 147
 - customizing 148
 - EntityInsight 146
 - HTML 145
 - inspector 147
 - JSP 145
 - XML 147

- tasks 36, 41
- template macros 140
- templates, code 138
- testing code 86
- testing, code
 - audits 60
- text
 - adding import statements 112
 - copying 110, 111, 112
 - deleting 110, 111
 - dragging and dropping 112
 - finding 118
 - formatting 113
 - pasting 110, 111, 112
 - selecting blocks 111
 - selecting in editor 110
- text searches 117
- thread contentions 86
- ToDo folder 77
- todo tags 124, 125
 - viewing 125
- tool tips 132
- toolbar, project pane 70
- toolbars, main window 12
- tools 57
 - object gallery 57, 58
 - visual design 57, 59
 - wizards 57
- tutorials 191
 - creating an applet 219
 - creating an application 191

U

- U.S. Government compliance 43
- undocking
 - message pane 81
 - panes 170
- unit testing 86
- Usenet newsgroups 44
- user interface elements, browser 9

V

- views, file tabs 14
- visual design tools *See* designer, visual

W

- Welcome Project 43
- wizards 57
 - Archive Builder 100
 - object gallery 57
- workspace configurations 167
 - editing 169
 - iconifying, docking, and undocking panes 170
 - managing 168
 - maximizing and restoring panes 169
 - menu locations 168
 - restoring original 167
 - splitting the content pane 171
 - tearing tabs 171
 - turning off 168
- wrapping text, message pane 81