

Inside the JBuilder OpenTools API

Keith Wood

Cataloging-in-Publication Data

Wood, Keith, 1961–

Inside the JBuilder OpenTools API / by Keith Wood.

ISBN 1-59457-427-8 (pbk.)

1. JBuilder (Computer tool). 2. Java (Computer file). 3. Computer software—Development. I. Title

© 2004, Keith Wood

All rights reserved

kbwood@iprimus.com.au

Published by BookSurge
5341 Dorchester Road Suite 16
North Charleston, SC, 29418

No part of this book may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the author.

ISBN 1-59457-427-8

10 9 8 7 6 5 4 3 2 1

JBuilder is a registered trademark of Borland Software Corporation in the United States and other countries. Other products mentioned are used for identification purposes only and may be trademarks of their respective companies.

Contents

Contents	i
Dedication	v
Preface.....	vi
What is in the book?	vi
Code Downloads	vi
Conventions	vi

Part III

The User Experience

Chapter 9	3
Utility Classes.....	3
AuralImage Class	5
ButtonStrip Class	6
CheckTree Class.....	7
CheckTreeNode Class	8
Classes Class.....	10
ClipPath Class	11
ClipPathRenderer Class.....	12
ColorCombo Class	12
ColorPanel Class	14
CompositelIcon Class	15
Debug Class	16
DefaultDialog Class	19
DialogValidator Interface	21
Diff Class.....	21
DiffEntry Class	23
DummyPrintStream Class.....	23
Icons Class	24
Icons.IconFactory Class.....	25
Images Class	25
JBuilderInfo Class	26
KeyStrokeDialog Class.....	30
KeyStrokeEditorPanel Class	31
KeyStrokeEditorTextField Class.....	33
ListPanel Class.....	34
PackageBrowserDialog Class	38
PackageBrowserFilter Interface	40
PackageBrowserTree Class.....	41
PathSet Class.....	43
Platform Class	49
ProjectPathSet Class.....	49
RegularExpression Class	53
RegularExpression.MatchResult Class.....	55
SearchTree	56
Streams Class	58
Strings Class	59
Strings.StringEncoding Class.....	62

TableSorter Class.....	62
Text Class.....	65
TextFile Class.....	66
TexturePanel Class	66
VetoException Class.....	67
ZipIndex Class.....	67
ZipIndexEntry Class	71
UISampler Example.....	72
Summary	76

Part V

The Editors and Viewers

Chapter 20	79
Component Modeling Tool	79
CmtComponents Interface.....	82
CmtComponent Interface	83
CmtComponentSource Interface.....	85
CmtComponentListener Interface.....	88
CmtSubcomponent Interface	88
CmtFeature Interface.....	92
CmtProperty Interface.....	92
CmtPropertySource Interface	93
CmtPropertyState Interface	93
CmtPropertySetting Interface	95
CmtModel Interface.....	95
CmtModelNode Interface.....	97
CMT Example	97
Summary	104
Chapter 21	105
UI Designers	105
DesignerManager Class	107
DesignerListener and DesignerReleaseListener Interfaces	108
DesignerEvent Class	109
Designer Interface.....	111
InternetBeansDesigner Example.....	113
InternetBeansModel and InternetBeansModelNode Examples	119
InternetBeansViewer Example	124
Summary	129
Chapter 22	130
Layout Assistants	130
LayoutAssistant Interface.....	131
DesignView Class.....	136
SelectBoxes Interface and SelectNib Class.....	137
BasicLayoutAssistant Class	140
BorderCornerLayoutAssistant Example.....	142
Summary	150

Part VI

The Wizard Framework

Chapter 24	155
Java Object Toolkit	155
JotPackages Interface	157
JotFile Interface	159
JotSourceFile Interface	159
JotMarker Interface	161
JotFileListener Interface	162
JotFileEvent Class	162
JotClass Interface	163
JotClassSource Interface	167
JotType Interface	169
JotMethod Interface	169
JotMethodSource Interface	170
JotCodeBlock Interface	171
JotSourceElement Interface	174
JotCommentable Interface	174
JotComment Interface	175
JotExpression Interface	176
JotStatement Interface	177
JSPTagWizard Example	179
Generating a Java Class	183
Summary	193

Part VIII

External Systems

Chapter 27	197
Version Control Systems	197
VCSFactory Class	198
VCS Class	199
RevisionInfo Class	202
AbstractRevisionNumber Class	203
VCSFileInfo Class	203
VCSFileStatus Class	204
VCSUtils Class	205
CommitAction Class	211
SourceSafeVCS Example	212
Summary	223
Chapter 28	224
Application Servers	224
JBoss Example	225
ServerManager Class	225
Service.Type Class	230
Service Class	232
Server Class	236
ServerLauncher Class	248
AppServerTargeting Class	255
AbstractDeploymentDescriptor Class	259
DeploymentDescriptor Class	260

AbstractDescriptorConversion Class	261
SetupManager Class	265
Setup Class	267
JBossSetup24 Example	268
SetupPage Interface	269
SetupPropertyPage Class.....	270
NestingSetupPropertyPage Class.....	270
Summary	271

Appendices

Appendix B.....	275
JBuilder Documentation Version Differences	275
Appendix C.....	282
Help Topics.....	282
Appendix D.....	287
XML Tools OpenTool	287
XMLValidator Interface.....	287
XSLTViewerFactory Class	290
XSLTNodeViewer Class.....	294
Index	297

List of Examples

UISampler Example.....	72
CMT Example	97
InternetBeansDesigner Example.....	113
InternetBeansModel and InternetBeansModelNode Examples	119
InternetBeansViewer Example	124
BorderCornerLayoutAssistant Example.....	142
JSPTagWizard Example.....	179
Generating a Java Class	183
SourceSafeVCS Example.....	212
JBoss Example.....	225
JBossSetup24 Example	268
XMLValidator Interface.....	287
XSLTViewerFactory Class	290
XSLTNodeViewer Class.....	294

Dedication

For James and Bronte

who wondered what I was doing all those nights.

Preface

This book is designed as an introduction to the OpenTools API within JBuilder, from version 7 through version 10, and as a reference for building your own tools. JBuilder is Borland's Java development environment and has been rated as the best Java IDE in several surveys of developers.

Due to space restrictions in the printed version of this book, several chapters were not able to be included. They are presented here instead.

What is in the book?

Part III examines several miscellaneous classes that contribute to the overall user experience within JBuilder. These let you manage persistent properties for controlling various aspects of the environment, provide help for your tools, and maintain the look-and-feel and functionality of JBuilder within your own tools.

Part V inspects the editors and viewers that let you interact with the content of the nodes in the project. Graphically manipulating a node is achieved through designers. You are free to develop new designers for specific elements within the node. Layout managers are central to UI development in Java, but they are not suited to use with graphical tools. The layout assistants provide the bridge between the two, letting you drag-and-drop for your own layout.

Part VI discusses the framework available for creating wizards within JBuilder. These tools guide you through a complex or repetitive process by asking simple questions, and often generate code as a result. To assist in writing that code, the Java Object Toolkit (JOT) classes give you access to the contents of a source file in an object-oriented manner. Conversely, you can parse existing code with JOT and extract information from it.

Part VIII talks about how JBuilder can interface with external systems – specifically Version Control Systems (VCS) and third-party application servers. Application servers provide a hosting environment for EJBs and other enterprise components. Through their integration with JBuilder you can easily deploy your classes to the server, start it running, and then debug your code within it.

Code Downloads

Code for all of the examples presented in this book is available for download on the accompanying Web site.

<http://home.iprimus.com.au/kbwood/JBOpenTools>

Ready-to-run versions of the tools (JAR files) are also included, so that you can make the most of them immediately.

You will also find links to several repositories of OpenTools at the Web site, along with links to other information about JBuilder and the OpenTools API.

Conventions

The main text of the book is set in a proportional font (like this), while terms introduced for the first time appear in *italics*, as do emphasized items. Code

samples, references to Java classes and methods, the names of directories and files, and text entered at a command line are presented in a `fixed font`. The names of menu items and other UI controls appear in a `sans-serif font`.

In many places throughout the book, references are made to the directories where JBuilder is installed. Since this may change for different versions of JBuilder and according to individual preferences, the notation “{JBuilder}” refers to the main JBuilder directory. Also, all path names use a forward slash (/) as the separator regardless of the operating system preference, in keeping with JBuilder’s own conventions. So, the following file name refers to one of the sample tools that comes with JBuilder and is found beneath the main installation directory.

```
{JBuilder}/samples/OpenToolsAPI/actions/Actions.jpx
```

The actual path name may be something like this on a Windows machine:

```
C:\JBuilderX\samples\OpenToolsAPI\actions\Actions.jpx
```

Furthermore, wherever project files are referred to in their `.jpx` format, it should be understood that the equivalent `.jpr` format would work just as well.

Certain additional points are highlighted within the book, as shown below. The purpose of the different categories is to make it easier to pick out this important or relevant information and its impact on the OpenTools API.



NOTE

Items of general interest look like this.



TIP

Tips for using JBuilder and its OpenTools API appear like this.



WARNING

Information necessary for the correct functioning of a portion of the API is brought to your attention in this manner. It includes class attributes that have been deprecated.



VERSION

Differences between versions of JBuilder are highlighted in this fashion. Items without this notation can be assumed to be the same throughout all the JBuilder versions covered by this book.



UNDOCUMENTED

Borland has not documented some sections of the OpenTools API because they may change in future versions of JBuilder. Those areas for which documentation does exist can be regarded as relatively static – you can expect them not to change, or at least be backward compatible, until a major update of the OpenTools API. Although you have access to the additional functionality of the undocumented features and can develop some very nifty tools using it, you may find that your tool does not work when you upgrade to a later version of JBuilder. These areas are highlighted throughout the book as shown here.



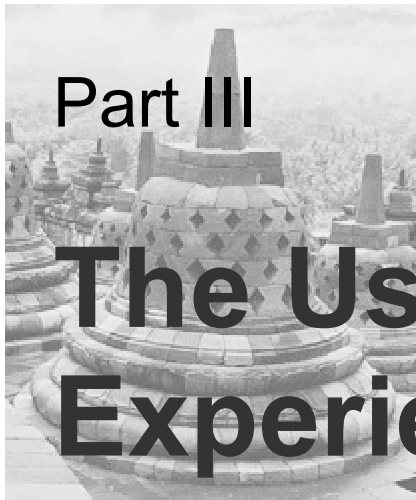
DOWNLOAD

Due to space constraints in the printed version, several chapters are included only in electronic format on the accompanying Web site. They are indicated by this format. It is also used to indicate other information that is available on the Web site.



BIO

Short biographies of the authors of various OpenTools appear in this style. These tools help to illustrate the classes and concepts within the book, show the breadth of possible tools, and act as a base for your own efforts. If you like a tool or have suggestions, please contact the author and tell them that. Most do it on their own time and with no payment.



Part III

The User Experience

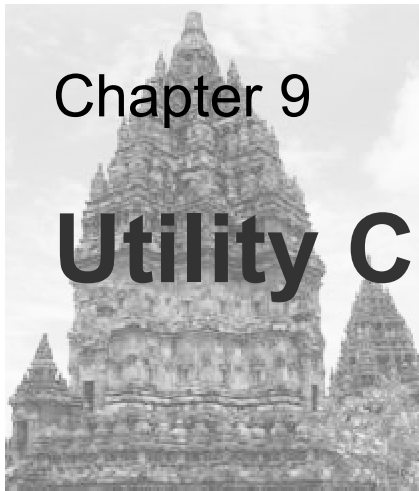
Like most applications, JBuilder has a certain look-and-feel associated with it. Much of the user experience comes from the capabilities of Java and the Swing components. The basic appearance and behavior of the controls is set by the selection of the Swing look-and-feel – Metal, Windows, Mac, or CDE/Motif. You set it by choosing the Tools | IDE Options or Preferences menu item, moving to the Browser or Look & Feel tab, and selecting the appropriate value from the drop-down list.

Other aspects come from additional functionality built into JBuilder's UI, such as the incremental searching in all tree controls. To assist you in integrating your tool into the IDE, JBuilder provides several sets of classes that make it easier to achieve that look-and-feel.

Chapter 9 contains a number of miscellaneous classes that supply useful functionality or a consistent user-experience. Browse through these as there may be something that already does what you are after.



Part III: The User Experience



Chapter 9

Utility Classes

This chapter presents a collection of miscellaneous classes that come with JBuilder. They provide basic functionality that is used throughout JBuilder itself, and are available for your use as well. Some of the classes define user interface components that can help you retain the look-and-feel of JBuilder in your tools. Table 9-1 summarizes the classes described. They come from the `com.borland.primetime.ui` (`p.ui` below), `com.borland.primetime.ui.table` (`p.ui.table` below), `com.borland.primetime.util` (`p.util`), `com.borland.jbuilder.ide` (`j.ide`), `com.borland.jbuilder.info` (`j.info`), and `com.borland.jbuilder.paths` (`j.paths`) packages, but are not the only ones in them. As usual, the Version column indicates which JBuilder version they first appeared in.



UNDOCUMENTED

Borland has not documented many of the classes described in this chapter. So, make use of them, but be aware that they may not be available, or may change, in the future.

Table 9-1. Summary of utility classes.

Class	Package	Purpose	Vers
AuralImage	p.util	Enhance an image by creating a colored aura around it	4
ButtonStrip	p.ui	Manage a set of buttons on a panel	4
CheckTree	p.ui	A tree view that displays a checkbox against each node	4
CheckTreeNode	p.ui	Each node in a CheckTree, providing support for determining the checked status	4
Classes	p.util	Retrieve classpath information and find classes on a classpath	4
ClipPath	p.util	Shorten a file name by removing path details	4
ClipPathRenderer	p.ui	Draw a shortened file name in a list cell	4
ColorCombo	p.ui	A combobox to select a color	4

Class	Package	Purpose	Vers
ColorPanel	p.ui	A panel showing colors to choose from	4
CompositeIcon	p.ui	Combine icons by overlaying them	4
Debug	p.util	Simple debugging output	4
DefaultDialog	p.ui	Dialog boxes in keeping with JBuilder's appearance	4
DialogValidator	p.ui	Validation interface for dialog boxes	4
Diff	p.util	Compare two sets of strings to find any differences	4
DiffEntry	p.util	A difference between two sets of strings	4
DummyPrintStream	p.util	A print stream that goes nowhere	4
Icons	p.util	Manage icons, including loading them, extracting them from a larger image, and generating disabled versions	4
Icons.IconFactory	p.util	Retrieve icons by extracting them from a larger image	4
Images	p.util	Work with images, including loading them, and generating disabled versions	4
JBuilderInfo	j.info	Details about the version and edition of JBuilder	4
KeyStrokeDialog	p.ui	A dialog for obtaining a keystroke combination	4
KeyStrokeEditorPanel	p.ui	A panel for obtaining a keystroke combination	4
KeyStrokeEditorTextField	p.ui	A text field for obtaining a keystroke combination	4
ListPanel	p.ui	A panel for managing lists of objects that have an ordering	4
PackageBrowserDialog	j.ide	A dialog for selecting a package or class	4
PackageBrowserFilter	j.ide	A filter for excluding packages or classes	5
PackageBrowserTree	j.ide	The tree display of packages or classes	4
PathSet	j.paths	Keeps track of a collection of paths	4
Platform	p.util	Details about the platform running JBuilder	4
ProjectPathSet	j.paths	Provides information about the paths for a JBuilder project	4
RegularExpression	p.util	Simple regular expressions	4
RegularExpression. MatchResult	p.util	Details about a match based on a regular expression	4
SearchTree	p.ui	An enhanced tree that allows searching by typing	4
Streams	p.util	Copy or read a stream	4

Class	Package	Purpose	Vers
Strings	p.util	Work with strings, including encoding and decoding them, and formatting them	4
Strings.StringEncoding	p.util	Escape or un-escape characters within a string	4
TableSorter	p.util p.ui.table	Sort data from a table model for display within a table	4
Text	p.util	Work with text, including removing whitespace and replacing tabs with spaces	6
TextFile	p.util	An enhanced file that lets you easily read or write straight text	6
TexturePanel	p.ui	A panel that repeats a background image across its surface	4
VetoException	p.util	Stop an action from continuing	4
ZipIndex	p.util	Details about entries in a Zip file	4
ZipIndexEntry	p.util	An individual Zip file entry	4

Have a look through the other classes in these packages that you can also use. However, most of them are undocumented and will require some experimentation to make the most of them.

AuralImage Class

Highlighting an image or an icon can be achieved through the `com.borland.primetime.util.AuraImage` class. It basically generates an outline (an aura) of a particular color around a given image, as shown in Figure 9–1. This experimental effect was decided against in JBuilder itself.

Figure 9–1.
An AuralImage.



UNDOCUMENTED

Borland has not yet documented the `AuraImage` class, so take care when using it.

The available methods of this class are:

```
public AuraImage(Image image);
public AuraImage(Image image, int auraColor);
public AuraImage(int[] pixels, int width, int height, int
    auraColor);
```

Create a new image with a colored outline.

`image` is the image to be outlined.

`auraColor` is the color (in RGB format, like `Color.blue.getRGB()`) to use in drawing the outline. If not specified, it defaults to green.

`pixels` is an array of color values from which to create an image. Although presented as a one-dimensional array, it actually contains a two-dimensional image of the size given below, with rows following each other sequentially along the single dimension. Therefore the length of this array is `width * height`.

height. To access pixel x, y you would use `pixels[y * width + x]`. See the `java.awt.image.PixelGrabber` class for more information.

`width` and `height` are the dimensions of the image encoded in the pixel array.

```
public static Image createAuraImage(Image image);
public static Image createAuraImage(Image image, int
    auraColor);
```

An alternate way to create an image with an aura, these methods return an outlined image for further processing.

`image` is the image to be outlined.

`auraColor` is the color (in RGB format) to use in drawing the outline. If not specified, it defaults to green.

```
public void dumpPixelMaps();
```

For debugging purposes, this method prints out the two maps used to mask out the original image and the calculated outline.

```
public int getAlphaThreshold();
```

Retrieve the current cutoff point for transparency when determining the extent of the image to outline.

```
public Image getAuraImage();
```

Obtain the image used for the aura. This is just the outline generated by this class.

```
public int getAuraRGB();
```

Return the color of the outline as a RGB value.

```
public Image getBlendedImage();
```

Get the combined image – the original plus the aura.

```
public Image getSourceImage();
```

Retrieve the original image provided to this object to be outlined.

```
public void setAlphaThreshold(int threshold);
```

Establish the cutoff point for transparency when determining the extent of the image to outline.

`threshold` is the maximum setting for the alpha value for a pixel for it to be considered transparent.



WARNING

The alpha threshold value does not seem to be used currently.

```
public void setAuraRGB(int rgb);
```

Set the color used for the aura outline.

`rgb` is the outline color encoded as a single integer value. Use the `getRGB` method of the `Color` class to obtain this value.

See the `UISampler` class that accompanies this chapter for examples of the `AuraImage` in action.

ButtonStrip Class

Arranging buttons on a panel is the purpose of the `com.borland.primetime.ui.ButtonStrip` class. It extends `JPanel` and lets you easily add a variety of buttons to it, as seen in Figure 9-1.

Figure 9–1. A ButtonStrip.**UNDOCUMENTED**

The `ButtonStrip` class is currently undocumented.

Its abilities are shown below:

```
public ButtonStrip();
public ButtonStrip(boolean separate);
public ButtonStrip(boolean separate, boolean reversed);
```

Generate a new button strip with one of these constructors.

`separate` is true (the default) to have the buttons on the strip inset by a small gap, or false to have them abut the edge.

`reversed` is true if the buttons are displayed in the reverse order to which they are added, or false if the creation order is used.

```
public JButton createButton(String text, boolean add);
public JButton createButton(String text, char mnemonic,
    boolean add);
```

Produce an ad hoc button and optionally add it to the panel.

`text` is the prompt displayed on the button.

`add` is true to add the new button to the panel, or false to just create and return it.

`mnemonic` is the special character to use for the button.

```
public JButton createCancelButton(boolean add);
public JButton createHelpButton(boolean add);
public JButton createNoButton(boolean add);
public JButton createOkButton(boolean add);
public JButton createYesButton(boolean add);
```

Create a button of the appropriate type and optionally add it to the panel. A reference to the button is returned for you to attach listeners to it. I believe that the text for the buttons is localized, but have not been able to verify this.

`add` is true to add the new button to the panel, or false to just create and return it.

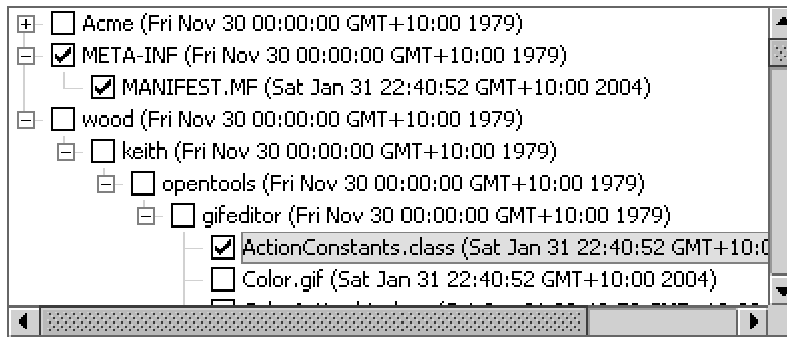
```
public void setOrientation(int orientation);
```

Establish the orientation for the panel.

`orientation` is the orientation setting. Use one of `HORIZONTAL` or `VERTICAL` from `javax.swing.SwingConstants`.

CheckTree Class

Displaying checkmarks next to items in a tree is easily achieved with the `com.borland.primetime.ui.CheckTree` class. It derives directly from `JTree` and adds support for displaying and updating the checkboxes shown in the UI, as Figure 9–2 shows. Just replace your normal trees with this version. It has no new methods, but relies on custom renderers, editors, and tree nodes.

Figure 9–2. A *CheckTree*.**UNDOCUMENTED**

All of the classes surrounding the `CheckTree` class are currently undocumented.

Its one method is shown here:

```
public CheckTree()
```

Just create a `CheckTree` where you would have used a normal `JTree`. The difference is in the nodes that make up the tree (as described in the next section) and the model used (which should be `CheckTreeModel`).

See the `UISampler` class at the end of this chapter for an example of the `CheckTree` in action, showing the contents of a Zip file.

CheckTreeNode Class

To get the most benefit from your `CheckTree` object, you should make all of its nodes instances of the `com.borland.primetime.ui.CheckTreeNode` class. Derived from `DefaultMutableTreeNode`, it adds methods to set and determine the checked state of a node, as well as to update the values used to display that node.

**UNDOCUMENTED**

Be aware that the `CheckTreeNode` class is not yet documented.

The methods of this class are listed below:

```
public CheckTreeNode(Object object);
public CheckTreeNode(Object object, boolean
    checkableAndChecked);
public CheckTreeNode(String text, boolean checkable,
    boolean checked);
```

Create new nodes for your `CheckTree` with one of these constructors.

`object` is the user object associated with this node. It allows additional information to be carried along with the node.

`checkableAndChecked` sets both the checkable and checked options to the same value. It defaults to false.

`text` is the display text for this node.

`checkable` determines whether the item appears with a checkbox.

`checked` indicates the initial state of the item.

```
public Icon getExpandedIcon();
    Retrieve the icon shown against this node when it is expanded. If not
    specified, it defaults to the normal icon for the node.
public Icon getIcon();
    Retrieve the normal icon shown against this node.
public String getText();
    Find the text displayed for this node.
public boolean isAffectedByParentEnabled();
    Discover whether or not this node is also enabled when its parent is enabled.
    Returns true if it is affected, or false if it is not.
public boolean isCheckable();
    Returns true if a checkbox appears against this node, and false if it cannot be
    checked at all.
public boolean isChecked();
    Determine the current state of this node.
public boolean isEnabled();
    Find out whether this node is enabled, returning true if it is, or false if it is
    not.
public boolean isEnablementAffectedByParent();
    Returns true (the default) if this node is enabled or disabled based on the
    checked status of its parent, or false if its enabled status is not altered.
```

**VERSION**

The `isEnablementAffectedByParent` method is only available in JBuilder 10.

```
public boolean isLocked();
    Discover whether this node may have its state altered, returning true if it
    cannot, or false (the default) if it can.
```

**VERSION**

The `isLocked` method is not available in JBuilder 7.

```
public void setAffectChildrenEnabled(boolean affected);
    When this node is enabled or disabled, this setting determines whether that
    change is sent on to its children.
    affected is true to pass this node's enabled status on to its children, or false
    to only update this node.
public void setAffectedByParent(boolean affected);
    Alter how this node responds when its parent is enabled or disabled.
    affected is true to have this node follow its parent's enabled status, or false
    to make it independent.
public void setChecked(boolean checked);
    Update the current state of this node. Depending on the values for other
    properties, this change may also affect the children of this node.
    checked sets the state of the displayed checkbox.
public void setEnabled(boolean enabled);
    Enable or disable this node through this method. The setting may be passed
    on to children of this node depending on the value established via set-
```

`AffectChildrenEnabled` (on this node) and `setAffectedByParent` (on its children).

`enabled` is `true` to enable this node, or `false` to disable it.

```
public void setEnablementAffectedByParent(boolean
    affected);
```

Indicate whether this node's enabled property is affected by its parent's.

`affected` is `true` (the default) if this node's enabled status changes in line with its parent's, or `false` if it is separate.



VERSION

The `setEnablementAffectedByParent` method is only available in JBuilder 10.

```
public void setExpandedIcon(Icon icon);
```

Set the icon shown against this node when it is expanded.

`icon` is the new image.

```
public void setIcon(Icon icon);
```

Modify the normal icon shown for this node.

`icon` is the new image.

```
public void setLocked(boolean locked);
```

Use this method to change whether this node can have its state updated.

`locked` is `true` to prevent the node from changing, or `false` (the default) to allow the user to alter its state.



VERSION

The `setLocked` method is not available in JBuilder 7.

```
public void setText(String text);
```

Alter the text displayed for this node.

`text` is the new display value.

```
public String toString();
```

Returns the same value as `getText`.

The `UISampler` class at the end of this chapter uses `CheckTreeNodes` when displaying the contents of a Zip file.

Classes Class

Utility methods that deal with classes and classpaths are the province of the `com.borland.prime.time.util.Classes` class. All of its methods are static so that they can be called directly. They are:

```
public static Url findPathUrl(Url[] classpath, String
    className);
```

Find the first entry in the classpath that contains a particular class. If the class cannot be found anywhere on the classpath it returns `null`.

`classpath` is the list of classpath entries to search. If `null` or empty the method always returns `null`.

`className` is the full name of the class to find.

```
public static String getRootEntryFromClasspath(Class
class);
public static String getRootEntryFromClasspath(String
className);
```

Retrieve the location from which a class was loaded by searching through the classpath. It is either the name of a directory or of a JAR file. The returned value is ready to be inserted directly into another classpath. If the class cannot be found it returns `null`.

`class` is a reference to the class being searched for.

`className` is the full name of the class to find.

```
public static String getShortName(Class class);
```

Extract just the name of the class provided by removing the package name. For example, if passed this class, the return value would be “Classes”. If an array reference is passed in, the name of its elements is returned instead.

`class` is the class whose name is desired.

```
public static boolean pathContainsClass(Url[] classpath,
String className);
```

Discover whether or not a class exists on a given classpath. The method returns true if the class can be found and false if it cannot.

`classpath` is the list of classpath entries to search. If `null` or empty the method always returns false.

`className` is the full name of the class to find.

```
public static String toPath(Class class);
```

Convert the full class name into a path name by replacing periods (`.`) with slashes (`/`).

`class` is the class whose name is converted.

ClipPath Class

The `com.borland.primetime.util.ClipPath` class provides two static methods that let you trim down a long path and file name so that it can be displayed in a limited area. It does this by removing one or more path entries, starting in the center of the full name, and replacing them with an ellipsis (`...`).

For example, to fit the path name into a label you could use the following:

```
label.setText(
ClipPath.clipCenter(label.getFont(), fileName, label.getWidth()));
```



UNDOCUMENTED

The `ClipPath` class, although mentioned in one of the help pages on OpenTools, is not otherwise documented by Borland.

The methods (all static) available in this class are:

```
public static String clipCenter(Font font, String path, int
width);
```

Trim the provided path and file name to fit.

`font` is the font to use when measuring the size of the text.

`path` is the file name to be displayed in a restricted area.

`width` is the number of pixels that the path has to fit into.

```
public static String clipMenuItemPath(String path, int width);
```

Shorten the given path and file name based on the font used for menu items.

path is the file name to be displayed in a restricted area.

width is the number of pixels that the path has to fit into.

The `UISampler` class described at the end of this chapter illustrates how the `ClipPath` class is used.

ClipPathRenderer Class

The `com.borland.primetime.ui.ClipPathRenderer` class makes use of the `ClipPath` class to draw an item in a list via the `ListCellRenderer` interface. Just create an instance of it through its no-argument constructor and assign that to the list or combobox (see Figure 9-3).

```
clipCombo.setRenderer(new ClipPathRenderer());
```

Figure 9-3. A `ClipPathRenderer` in a combobox.



UNDOCUMENTED

Borland does not document the `ClipPathRenderer` class, but then there is basically nothing to know about it anyway.

ColorCombo Class

Selecting a color can be performed via the `com.borland.primetime.ui.ColorCombo` class. It extends `JComboBox` to display a swatch of the current color and pops up a `ColorPanel` to select a new one, as shown in Figure 9-4.

Figure 9-4. A `ColorCombo`.



UNDOCUMENTED

The `ColorCombo` class is undocumented so far.

Its abilities are described below:

```
public ColorCombo();
public ColorCombo(Color[] colors);
public ColorCombo(Color[] colors, int height);
public ColorCombo(Color[] colors, int height, int alignment);
```

Build a new color combo component with the given initial values.

colors is the list of custom colors to show. It must contain eight entries.

`height` is the height of the popup color panel in cells. It should be a power of two. It defaults to `DEFAULT_POPUP_GRID_HEIGHT` (2).

`alignment` is the alignment of the popup color panel. It defaults to `RIGHT`.

```
public static Color decodeColor(String colorText);
```

Convert a single color from a text representation to a `Color` object.

`colorText` is the text form of the color.

```
public static Color[] decodeColors(String colorList);
```

Convert a list of colors, specified in a string and separated by colons (:), into an array of `Color` objects.

`colorList` is the list of colors to convert. A suitable list may be obtained via the `encodeColors` method.

```
public static String encodeColor(Color color);
```

Transform a color into a text version of itself. The returned value is a set of three integers, separated by commas (,), being the red, green, and blue levels (0 to 255) of the color.

`color` is the color to translate.

```
public static String encodeColors(Color[] colors);
```

Transform a list of colors into a corresponding string representation. A colon (:) separates each color from its neighbors.

`colors` is the list of colors to convert.

```
public Color[] getCustomColors();
```

Return the list of colors shown in the right-hand portion of the popup color panel.

```
public int getPopupAlignment();
```

Retrieve the alignment for the popup color panel.

```
public int getPopupGridHeight();
```

Obtain the height of the popup color panel in cells.

```
public Color getSelectedColor();
```

Find the color selected by the user, or black if none has yet been selected or set.

```
public void setCustomColors(Color[] colors);
```

Load the list of custom colors to show in the right-hand portion of the popup color panel.

`colors` is the list of colors to show. It must be an array of eight `Color` objects.

```
public void setPopupAlignment(int alignment);
```

Set the alignment for the popup color panel.

`alignment` is the new setting. Use either of the constants `LEFT` or `RIGHT` from this class.

```
public void setPopupGridHeight(int height);
```

Establish the height of the popup color panel in cells.

`height` is the new setting. It should be a multiple of two. The default value is held in the `DEFAULT_POPUP_GRID_HEIGHT` constant.

```
public void setSelectedColor(Color color);
```

Change the chosen color to the given value.

`color` is the new color to use for this control.

An example of the `ColorCombo` class appears in the `UISampler` class discussed at the end of the chapter.

ColorPanel Class

Displaying a set of colors to choose from is the purpose of the `com.borland.prime.time.ui.ColorPanel` class, as can be seen in Figure 9–5. It can be used standalone, or as part of the `ColorCombo` component.

Figure 9–5. A ColorPanel.



UNDOCUMENTED

Be aware that the `ColorPanel` class is undocumented so far.

Its abilities are listed below:

- ```
public ColorPanel();
public ColorPanel(Color[] colors, int value);
```
- Create a new panel containing color swatches for selection with these constructors.
- `colors` is the list of custom colors to show. It must contain eight entries.
- `value` appears to be ignored.
- ```
public void addActionListener(ActionListener listener);
```
- Add a new object to be notified of changes to the selected color.
- `listener` is the object to inform.
- ```
public String getActionCommand();
```
- Return the command string that gets passed along to registered listeners when a new color is selected.
- ```
public Color[] getCustomColors();
```
- Retrieve the set of custom colors displayed in the right-hand portion of the panel.
- ```
public Color getSelectedColor();
```
- Get the color selected by the user.
- ```
public void removeActionListener(ActionListener listener);
```
- Delete an object from the list of those notified of changes to the selected color.
- `listener` is the object to remove.
- ```
public void setActionCommand(String command);
```
- Set the command string that gets passed along to registered listeners when a new color is selected.
- `command` is the text to send on.
- ```
public void setCustomColor(int index, Color color);
```
- Update a custom color to a new value.
- `index` is the position within the custom color array to alter. It must be offset by the number of fixed colors in the panel. Thus the index for the first

custom color is given by `colorPanel.fixedColors.length` (currently 16), while the last is found by adding `colorPanel.GetCustomColors().length` to that and subtracting one (resulting in 23 at present).
`color` is the new color to show.

**WARNING**

Setting an individual custom color does not automatically update the display. What you need to do is to set the entire collection of colors at once. You can achieve the same effect with the following code:

```
colorPanel.setCustomColor(16, Color.pink);
colorPanel.setCustomColors(colorPanel.getCustomColors());
```

```
public void setCustomColors(Color[] colors);
```

Set the custom colors to display in the right-hand portion of the panel.

`colors` is the list of colors to show. It must contain eight entries.

```
public void setPanelGridHeight(int height);
```

Establish the height of the popup color panel in cells.

`height` is the new setting. It should be a power of two. The default value is two.

```
public void setSelectedColor(Color color);
```

Initialize the color currently selected.

`color` is the chosen color. The default color is black.

Several static fields are also defined in this class:

```
public static final Color[] fixedColors;
```

This array contains the colors shown on the left in the panel – the ones that do not change.

```
public static final Color BLUE_GREEN;
public static final Color DARK_BLUE;
public static final Color DARK_GREEN;
public static final Color DARK_MAGENTA;
public static final Color DARK_RED;
public static final Color DARK_YELLOW;
```

These constants are the additional colors (beyond those available in the `Color` class) that appear in the main part of the panel.

See the `UISampler` class at the end of the chapter for an example of the `ColorPanel` class.

Compositelcon Class

Combine multiple icons to produce a single version with the `com.borland.prime.time.ui.CompositeIcon` class, as shown in Figure 9-6.

Figure 9-6. A *CompositeIcon*.



It implements the `Icon` interface, adding these methods:

```
public CompositeIcon(Icon[] icons);
```

Create a new icon made up of several components with this constructor. The icons are painted in the order that they appear in this list. Each icon is centered in the space allocated for the composite.

`icons` is the list of icons to combine.

```
public CompositeIcon(Icon[] icons, int separation, int orientation);
```

Create a new icon made up of several components with this constructor. In this case, the icons create a grid based on the orientation supplied.

`icons` is the list of icons to combine.

`separation` is the pixel distance between neighboring icons in the grid.

`orientation` is the direction in which to create the grid. It is one of the `HORIZONTAL` or `VERTICAL` constants from this class.

```
public Icon findHit(Point point);
```

Determine which icon from the collection corresponds to a particular location – perhaps as the result of the user clicking on the icon. A `null` is returned if no hit is found.

`point` is the position within the combined icon to locate.

See the `UISampler` class at the end of the chapter for an example of the `CompositeIcon` class.

Debug Class

To assist you with your debugging efforts by logging details from your tools, the `com.borland.primetime.util.Debug` is provided. By default it writes messages to the standard error stream, but this can be redirected to any print stream.

All the methods of the class are static, and are shown below:

```
public static void addTraceCategory(Object category);
```

Filter trace and warn method calls by specifying categories that are to be logged.

`category` is a unique object that is registered via this method. Thereafter trace or warn calls are only displayed if the category specified there is one that has been previously registered here. All categories should generate a meaningful value via their `toString` method since this is displayed with each logging event.

```
public static void debugRect(Graphics g, int x, int y, int width, int height);
```

Each time that this method is called it draws a different colored and patterned rectangle on the graphics surface, providing visual feedback that it has been called.

`graphics` is the surface to paint onto.

`x` and `y` are the top-left coordinates of the rectangle to draw.

`width` and `height` determine the extent of the rectangle to draw.

```
public static void enableAssert(boolean enable);
```

Enable or disable the use of assertions within this class.

`enable` is true to cause the `ensure` methods to throw exceptions when their test conditions are not true, or false to have the `ensure` methods never throw an exception.

```
public static void enableOutput(boolean enable);
```

Enable or disable the logging as a whole through this method.

`enable` is true to send subsequent output to the standard error stream, or false to suppress the output (through the use of a `DummyPrintStream` object).

```
public static void ensure(boolean condition) throws
    AssertionError;
```

```
public static void ensure(boolean condition, String
    description) throws AssertionError;
```

Throw an exception if the condition is not true and assertions are enabled (see the `enableAssert` method).

`condition` is the expression to test with the expectation that it is true.

`description` is the text to pass to the `AssertionException` should it be thrown.

```
public static void flush();
```

Flush out the output stream.

```
public static void print(String message);
```

```
public static void println(String message);
```

```
public static void printlnc(String message);
```

Send the message to the output stream, with or without a final newline character. The last version precedes the text with a counter that is incremented on each call.

`message` is the text to display.

```
public static void printProfiler(Object key);
```

Display all the details acquired by a profiler, including minimum, maximum, and average timings.

`key` is some identifying object.



VERSION

The `printProfiler` method is not available in JBuilder 7.

```
public static void printStackTrace();
```

```
public static void printStackTrace(Throwable exception);
```

Display a stack trace on the output stream.

`exception` is the exception to print the stack trace for. If no exception is specified, a default one is generated and dumped.

```
public static void removeTraceCategory(Object category);
```

Delete a category from the list of those checked for trace and warn calls.

`category` is the object that was previously registered for logging via the `addTraceCategory` call.

```
public static void setLogStream(PrintStream log);
```

Redirect the logging output to another stream with this method.

`log` is the new print stream instance to send subsequent output to.

```
public static void startProfiler(Object key);
```

Begin, or restart, a profiler.

`key` is some identifying object.

**VERSION**

The `startProfiler` method is not available in JBuilder 7.

```
public static void stopProfiler(Object key);
public static void stopProfiler(Object key, int
    warningThreshold);
```

Stop a previously started profiler.

`key` is some identifying object.

`warningThreshold` is the number of milliseconds for the current timing beyond which a warning message is displayed.

**VERSION**

The `stopProfiler` methods are not available in JBuilder 7.

```
public static void trace(Object category, String
    description);
public static void trace(Object category, Throwable
    exception);
```

Send an informational message to the output stream, but only if the supplied category is one that was previously registered via the `addTraceCategory` method and if output is enabled (see `enableOutput`).

`category` is the object identifying the category to log against.

`description` is the message to print out if everything is enabled.

`exception` is the exception to print a stack trace for if everything is enabled.

```
public static void warn(Object category, String
    description);
public static void warn(Object category, boolean condition,
    String description);
```

Send a warning message to the output stream, but only if the supplied category is one that was previously registered via the `addTraceCategory` method and if output is enabled (see `enableOutput`).

`category` is the object identifying the category to log against.

`description` is the message to print out if everything is enabled.

`condition` is an additional test to evaluate before printing the message. Only if this is true, as well as everything else being enabled, is the message displayed.

Two fields round out the functionality provided by the `Debug` class:

```
public static int count;
```

This is the counter variable that is incremented and printed by the `println` method.

```
public static PrintStream out;
```

This is the current output stream that the log is being written to. Initially this is `System.err`.

A debugging section in your code may look like the following:

```
String tracing = "trace";
String warning = "warning";
Date date = null;
:
```

```

public void setStartDate(Date date) {
    Debug.trace(tracing, "date=" + date);
    Debug.warn(warning, date.before(new Date()),
        "Start date must not be in the past");
    this.date = date;
}
:
Debug.addTraceCategory(tracing);
Debug.addTraceCategory(warning);
:
Date now = new Date();
setStartDate(new Date(now.getTime() - 10000));
setStartDate(new Date(now.getTime() + 10000));

```

which would produce this output:

```

[trace] date=Thu May 16 18:17:24 GMT+10:00 2002
warn[warning] Start date must not be in the past
[trace] date=Thu May 16 18:17:44 GMT+10:00 2002

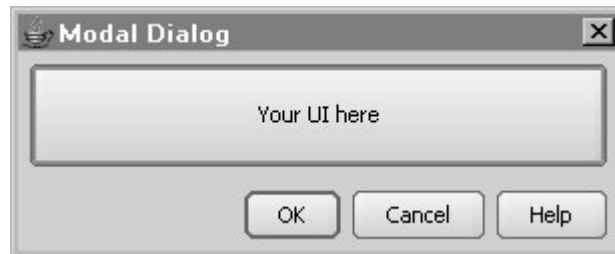
```

You can then control the level of debugging output by calling `addTraceCategory` only for those levels that you want.

DefaultDialog Class

To retain the appearance of JBuilder's dialog boxes, you can use the `com.borland.prime.time.ui.DefaultDialog` class (see Figure 9-7) that extends the standard `JDialog`. It provides static methods so that you can easily create and display a dialog, as well as instance methods that let you customize the dialog before showing it. Features include auto-centering, finding an owning frame, and modifying buttons.

Figure 9-7. A *DefaultDialog*.



Its methods are:

```

public DefaultDialog(Component owner, String title, boolean
    modal);

```

```

public DefaultDialog(Component owner, String title, boolean
    modal, boolean autoSize);

```

Create a new dialog to allow for customization before its display. You must add your own content to the dialog and set the buttons that it displays.

`owner` is a component on the frame that owns this dialog.

`title` is the caption for the new dialog.

`modal` is true for a modal dialog, or false for a non-modal version.

`autoSize` is true (the default) to have the dialog resize itself based on its contents, or false if the size is set manually.

```

public void centerOnScreen();

```

Move the dialog so that it is centered on the screen.

```
public void doDefaultClick();
```

Trigger the `doClick` method of the default button in the dialog.

```
public static Frame findFrame(Component component);
```

Search the containership hierarchy to find the frame that holds a particular component. This method is usually used to locate an owning frame for the dialog.

`component` is the control from which to start searching.

```
public String getBoundsAsString();
```

Retrieve the bounds for this dialog as a comma-separated string in the format “left,top,width,height”.

```
public boolean isAutoCenter();
```

Determine whether the dialog is automatically centered when displayed. It returns true if it is centered or false if it is not.

```
public void setAutoCenter(boolean autoCenter);
```

Alter where the dialog appears.

`autoCenter` is true (the default) to automatically center the dialog, or false to use the standard dialog placement.

```
public void setBoundsAsString(String location);
```

Given the bounds as a string in the format “left,top,width,height”, apply them to the dialog.

`location` is the bounds in the above format.

```
public void setCancelButton(JButton cancelButton);
```

Set the button to activate if the dialog is cancelled.

`cancelButton` is the cancel button in your UI.

```
public void setDefaultButton(JButton defaultButton);
```

Establish which button in your UI is the default one.

`defaultButton` is the button to use as the default.

```
public void setHelpButton(JButton helpButton);
```

Link a button to the help action in the dialog (pressing F1).

`helpButton` is the button that brings up help for the dialog.

```
public void show();
```

If you created the dialog yourself (rather than using one of the static methods below), use this method to display it to the user.

```
public static boolean showModalDialog(Component owner,
    String title, JComponent component, HelpTopic
    helpTopic);
```

```
public static boolean showModalDialog(Component owner,
    String title, JComponent component, HelpTopic helpTopic,
    DialogValidator validator);
```

```
public static void showSimpleModalDialog(Component owner,
    String title, JComponent component, HelpTopic
    helpTopic);
```

```
public static void showSimpleNonModalDialog(Component
    owner, String title, JComponent component, HelpTopic
    helpTopic);
```

Display a standard dialog box with OK and Cancel buttons, and an optional Help button. The simple versions do not have a Cancel button. You supply the remaining contents of the dialog. The dialog is automatically sized by calling `pack`.

`owner` is a component on the frame that owns this dialog.

`title` is the caption for the dialog.

`component` is the component that holds the contents of the dialog, excluding the standard buttons.

`helpTopic` is the help topic to display when help is requested by the user, or `null` for no help. The Help button only appears if this is not `null`.

`validator` is an object that implements the `DialogValidator` interface (see below) and verifies the contents of the dialog before closing. If not specified, the contents are assumed to be valid, unless the component supplied for the UI implements this interface, in which case it is used instead. No validation is performed for the simple version of this method.

See the `UISampler` class at the end of the chapter for examples of the `DefaultDialog` class' static methods.

DialogValidator Interface

Ensuring that the contents of a dialog are valid before allowing it to be closed is the task of the `com.borland.primetime.ui.DialogValidator` interface. Its one method simply verifies its associated dialog:

```
public boolean validateDialog();
```

Check the contents of the associated dialog box (usually a `DefaultDialog` object) and return `true` if everything is fine, or `false` to prevent the dialog from closing. You should display your own messages to the user indicating what is wrong with the dialog's contents.

A basic implementation of this interface appears in the `UISampler` class at the end of the chapter.

Diff Class

Comparing two text files is made easier by using the `com.borland.primetime.util.Diff` class. It takes in two sets of text, either as files, as readers, or as arrays of strings, and generates the differences as an array of `DiffEntry`s. The resulting array is cached within the class and is accessed via an iterator. Differences can also be presented as an edit script in the format of the “--rcs” output from the Unix `diff.exe` utility.

```
public Diff();
```

Create a new differencing object.

```
public void addAddition(int startLine, String[] lines);
```

Manually update the differences by including an addition, meaning that lines are inserted into the original text.

`startLine` is the number of the line after which the new lines are added. Use zero to insert at the beginning of the text.

`lines` is the array of new lines to insert.

```
public void addChange(int startLine, String[] lines);
```

Alter the list of differences by specifying a set of lines from the original that change.

`startLine` is the number of the first line (one-based) that is changed.

`lines` is the array of changed lines to replace the existing lines. A matching number of lines in the original are affected.

```
public void addDeletion(int startLine, int count);
```

Update the differences manually by including a deletion, meaning that lines are removed from the original text.

`startLine` is the number of the first line (one-based) that is deleted.

`count` is the number of lines to remove.

```
public boolean diff(File file1, File file2);
```

```
public boolean diff(String[] lines1, String[] lines2);
```

```
public boolean diff(BufferedReader reader1, BufferedReader
    reader2);
```

```
public boolean diff(BufferedReader reader1, BufferedReader
    reader2, boolean preformat, String contentType);
```

Compare the two texts and cache any differences internally. Any existing differences are cleared out before dealing with this call. The methods return true if the difference processing succeeded, or false if it did not.

`file1` and `file2` are the two files to compare.

`lines1` and `lines2` are the two arrays of strings to compare.

`reader1` and `reader2` are the two readers to compare.

`preformat` is true to apply a formatting standard before comparison, or false (the default) to leave the readers' contents as they are.

`contentType` is the type of content within the readers. It helps to determine which formatting standard is applied, and defaults to "text/plain".



VERSION

The `diff` method that takes the `preformat` and `contentType` parameters is not available in JBuilder 7.

```
public boolean editScriptToDiff(Iterator diffFile);
```

If you already have a set of differences in the "--rcs" output format of `diff.exe`, you can transform it into a cached set of `DiffEntry` objects through this method. It returns true if it is successful in the conversion, or false if it fails.

`diffFile` is the existing differences in the appropriate format, accessed via an iterator over an array of strings.

```
public Iterator iterator();
```

Obtain an iterator that steps through the cached differences in normal order. Each item returned by the iterator is a `DiffEntry` instance.

```
public void reset();
```

Clear out any differences in this object in preparation for another comparison. Normally you would not call this directly since all the `diff` methods automatically use it before their processing. If you are manually updating the list of differences then you may want to use it.

```
public Iterator reverseIterator();
```

Retrieve an iterator that steps through the cached differences in reverse order. Each item returned by the iterator is a `DiffEntry` instance.

```
public int size();
```

Return the number of differences currently cached in this object.

```
public String[] toEditScript();
```

Use this method to convert the cached set of differences into a “diff.exe -rcs” style edit script stored as an array of strings. The result only contains add and delete instructions.

There is an example tool that comes with JBuilder that demonstrates the `Diff` class:

```
{JBuilder}/samples/OpenToolsAPI/DiffViewer/DiffViewer.jpx
```

A node viewer that lets you compare the current Java source file with another file on disk to see what differences there are between them. It appears as the Diff View tab in the Content Pane, but only for Java source nodes. It also demonstrates the use of an editor and the highlighting of lines within it, and uses the `Diff` class to locate the differences between the two files.

DiffEntry Class

The differences found by the `Diff` class are cached internally as an array of `com.borland.primetime.util.DiffEntry` objects.



UNDOCUMENTED

Be aware that the `DiffEntry` class is undocumented.

This class has several fields that you can access directly:

```
public int count;
```

The number of lines affected by this difference.

```
public String[] newLines;
```

The set of lines that are added or changed, depending on the type of this difference. For deletions, this field is `null`.

```
public int startLine;
```

The number of the first line affected by this difference.

```
public int type;
```

The type of difference – being one of the class’ constants: `ADD`, `CHANGE`, or `DELETE`.

DummyPrintStream Class

A print stream that does nothing, `com.borland.primetime.util.DummyPrintStream` can be useful when a print stream is required but you are not interested in the output. Use it wherever a normal print stream would be used, since it has all the standard methods.



UNDOCUMENTED

The `DummyPrintStream` class is undocumented, but it does not do anything anyway.

Icons Class

Managing sets of icons is the task of the `com.borland.primetime.util.Icons` class. It typically works with a composite image that contains a large number of same-sized icons arranged in either a single row, or in a grid. Rather than loading each icon individually, the main image is retrieved and then each icon that it contains can be extracted from memory.



UNDOCUMENTED

So far the `Icons` class is undocumented.

The abilities of the `Icons` class are listed below, all being static methods:

```
public static Icon getBlankIcon(int width, int height);
```

Return an empty icon of the given size.

`width` and `height` are the sizes of the blank icon to generate.

```
public static synchronized Icon getDisabledIcon(Icon icon);
```

Create a disabled version of the given icon – converting it to a grayscale image.

`icon` is the original icon to be disabled.

```
public static Icon getIcon(URL url);
```

Retrieve the image from a specified location and then create an icon from it.

`url` is the location to load the image from.

```
public static Icon getIcon(Class class, String imagePath);
```

Create an icon from the image specified as a resource.

`class` is the class whose loader is used to locate the resource.

`imagePath` is the full path and file name for the resource file that contains the image.

```
public static Icon getIcon(Image image);
```

```
public static Icon getIcon(Image image, int left, int top,
int width, int height);
```

Construct an icon from an existing image, or portion thereof.

`image` is the picture to work from.

`left` and `top` are the starting coordinates for the portion of the image to extract. They default to zero if not specified.

`width` and `height` are the size of the image portion to extract. They default to the entire image if not specified.

```
public static IconFactory getIconFactory(Image image, int
size);
```

```
public static IconFactory getIconFactory(Image image, int
width, int height);
```

```
public static IconFactory getIconFactory(Image image, int
width, int height, int spacing);
```

Create an icon factory based around a composite image. Once obtained, use the factory to extract the individual icons.

`image` is the composite image containing multiple icons.

`size` is the width and height of the embedded icons (assumed to be square).

`width` and `height` are the dimensions of the embedded icons.

`spacing` is the vertical and horizontal gap between successive icons.

Icons.IconFactory Class

The `com.borland.primetime.util.Icons.IconFactory` class handles the actual extraction from a composite image, which contains a regular array of sub-images.



UNDOCUMENTED

As for the `Icons` class, this one is also undocumented.

Its methods are as follows:

```
public IconFactory(Image image, int size);
public IconFactory(Image image, int width, int height);
public IconFactory(Image image, int width, int height, int
    spacing);
```

Create a new icon factory based on the supplied image.

`image` is the composite image containing multiple icons.

`size` is the width and height of the embedded icons (assumed to be square).

`width` and `height` are the dimensions of the embedded icons.

`spacing` is the vertical and horizontal gap between successive icons.

```
public Icon getIcon(int column);
public Icon getIcon(int row, int column);
```

Retrieve a particular sub-image from the factory.

`column` is the position of the icon horizontally within the larger image, starting from zero.

`row` is the position of the icon vertically within the larger image, starting from zero. If not specified, the row is set to zero.

Images Class

Load images with the `com.borland.primetime.util.Images` class to ease the effort required.



UNDOCUMENTED

The `Images` class is undocumented.

Its static methods are described here:

```
public static Image getAuraImage(Image image);
```

Create a default (green) `AuraImage` from the given picture and return it. This method adds a halo around the original image.

`image` is the picture to outline.

```
public static synchronized Image getDisabledImage(Image
    image);
```

Build a disabled version (grayed out) of the given picture and return it.

`image` is the picture to be disabled.

```
public static Image getImage(URL url);
```

```
public static synchronized Image getImage(URL url, boolean
    wait);
```

Retrieve the image from a specified location.

`url` is the location to load the image from.

`wait` is true (the default) to not return until the image is available, or false to return immediately.

```
public static Image getImage(Class class, String
    imagePath);
public static Image getImage(Class class, String imagePath,
    boolean wait);
```

Load an image that is specified as a resource.

`class` is the class whose loader is used to locate the resource.

`imagePath` is the full path and file name for the resource file that contains the image.

`wait` is true (the default) to not return until the image is available, or false to return immediately.

```
public static synchronized void waitForImage(Image image);
    Wait for an image to be loaded.
    image is the image being read in.
```

JBuilderInfo Class

Details about the version of JBuilder running are available through the `com.borland.jbuilder.info.JBuilderInfo` class.



UNDOCUMENTED

The `JBuilderInfo` class remains undocumented, and has indeed changed over time.

Use its static methods (not all are included) to obtain this information, as listed below:

```
public static final void actionVerify(UpdateAction action,
    int edition);
```

Enable or disable an updateable action based on the edition of JBuilder in use.

`action` is the action to enable or disable.

`edition` is one of the constants from the `com.borland.primetime.actions.UpdateAction` class indicating which level of JBuilder is required for this action's functionality: `PERSONAL`, `PROFESSIONAL`, or `ENTERPRISE`.



VERSION

The `actionVerify` method does not appear in JBuilder 10.

```
public static String getBuildNumber();
```

Obtain the current build number, such as "10.0.176.0". Compare with the `getRawBuildNumber` method.



VERSION

The `getBuildNumber` method is only available in JBuilder 10.

```
public static String getCompanyName();
```

Return the company name specified by the user during registration.

```
public static int getDaysLeft();
```

If this is a time-limited trial version of JBuilder, find the number of days left before it expires through this method. It returns zero if no time limit applies.

VERSION

The `getDaysLeft` method has been dropped in JBuilder 10.

```
public static String getDescription();
```

Obtain the full description of this JBuilder version, such as: "JBuilder X Enterprise".

```
public static String[] getExpansionPackNames()
```

Get the list of expansion packs installed in JBuilder. The method returns an empty array if none are present.

```
public static String[] getExtraDescriptions();
```

Retrieve any extra descriptions for this edition of JBuilder. An empty array is returned if no other messages apply. For example, the Personal edition may return: "Not for commercial use".

```
public static String getRawBuildNumber();
```

Obtain the raw build number for this version of JBuilder, like "010.000.176.000". Compare with the `getBuildNumber` method.

VERSION

The `getRawBuildNumber` method is only available in JBuilder 10.

```
public static int getSKU();
```

Retrieve the SKU number for this version of JBuilder. Use the `getSKUName` method to translate this into text.

VERSION

The `getSKU` method is only available in JBuilder 10.

```
public static String getSKUDescription();
```

```
public static final String getSKUDescription(int SKU);
```

```
public static String getSKUName(int SKU);
```

Get the description of this edition of JBuilder: Trial, Professional, Enterprise, etc.

SKU is the identifying number for that edition.

VERSION

The `getSKUDescription` method that takes an integer parameter is only available in JBuilder 9. The `getSKUName` method is only available in JBuilder 10.

TIP

You can easily downgrade JBuilder's SKU for testing purposes by including a system property in its start up process. In your `jbuilder.config` file, add the line:

```
vmparam -Dcom.borland.jbuilder.sku=Personal
```

and then restart JBuilder to restrict it to the Personal/Foundation edition. Use a value of "Developer" for the Professional/SE/Developer edition.

```
public static String getUsername();
```

Return the name of the user as given during registration.

```
public static boolean isBeaEnabled();
```

Find out if this is the BEA version of JBuilder. It returns true if it is, or false if it is not.



VERSION

The `isBeaEnabled` method is not available in JBuilder 7.

```
public static boolean isComponentEnabled(String className);
```

Determine whether or not the given class is enabled under this version of JBuilder. It returns true if the component can be used, or false if it cannot.

`className` is the full name of the class being checked.

```
public static boolean isEntEnabled();
```

```
public static boolean isEnterpriseEnabled();
```

Find out if this is an Enterprise edition of JBuilder. It returns true if it is, or false if it is not.



VERSION

The `isEntEnabled` and `isEnterpriseEnabled` methods are not available in JBuilder 10. Use `isGenericPremiumEnabled` instead in JBuilder 10.

```
public static boolean isFoundationOnlyEnabled();
```

Returns true if using the Foundation edition of JBuilder, or false if using one of the other editions.



VERSION

The `isFoundationOnlyEnabled` method is only available in JBuilder 10.

```
public static boolean isGenericPremiumEnabled();
```

Find out whether the premium (enterprise) edition of JBuilder is being used, returning true if it is, or false if it is not.



VERSION

The `isGenericPremiumEnabled` method is only available in JBuilder 10.

```
public static boolean isGenericValueEnabled();
```

Discover whether the value (professional) edition of JBuilder is being used, returning true if it is, or false if it is not.



VERSION

The `isGenericValueEnabled` method is only available in JBuilder 10.

```
public static boolean isLibraryEnabled(Url url);
```

Discover whether a particular library of classes is available under this edition of JBuilder, returning true if it is, or false if it is not.

`url` is the location of the library to check.

```
public static boolean isLicensed();
```

Get a flag indicating whether or not this copy of JBuilder is licensed. The method returns true if it is licensed, or false if it is not.

```
public static boolean isProEnabled();
```

```
public static boolean isSeEnabled();
```

Discover if this is a Professional/SE edition of JBuilder. It returns true if it is, or false if it is not. This method also returns true if it is an Enterprise edition.

**VERSION**

The `isProEnabled` method is not available in JBuilder 7 and 8, while `isSeEnabled` is only available in JBuilder 7 through 9. Use `isGenericValueEnabled` in JBuilder 10.

```
public static boolean isStudio();
```

Returns true if the Studio edition of JBuilder is being used, or false if it is not.

**VERSION**

The `isStudio` method is only available in JBuilder 10.

```
public static boolean isSybaseEnabled();
```

Find out if this is the Sybase version of JBuilder. It returns true if it is, or false if it is not.

**VERSION**

The `isSybaseEnabled` method is not available in JBuilder 10.

```
public static boolean isTermLicense();
```

Determine whether the license expires. It returns true if it does, or false if it does not. Use the `getDaysLeft` method to determine when the copy expires.

```
public static boolean isTrial();
```

Find out if this is a Trial version of JBuilder. It returns true if it is, or false if it is not.

```
public static void launchWizard();
```

```
public static void launchWizard(boolean immediate);
```

Bring up the JBuilder Registration Wizard with this method.

`immediate` is true to reload settings immediately, or false to wait until the next restart.

**VERSION**

The first version of the `launchWizard` method above is only available in JBuilder 7, while the last version is only available in JBuilder 8 and 9.

```
public static void setStatusMsg(String text, int type, int timeout);
```

Display a message in the Status Pane (if it is available).

`text` is the string to display, or an empty string to clear the Status Pane.

`type` indicates the format for the displayed text. Use one of these values from the `StatusView` class (see Chapter 13): `TYPE_NORMAL`, `TYPE_WARNING`, or `TYPE_ERROR`.

`timeout` is the time in milliseconds after which the message is cleared. It defaults to the value of `TIMEOUT_DEFAULT` (10 seconds). You can also use the `TIMEOUT_NONE` value (from the `StatusView` class) for a permanent message.

**VERSION**

The `setStatusMsg` method is only available in JBuilder 10.

```
public static void showInfoDialog();
```

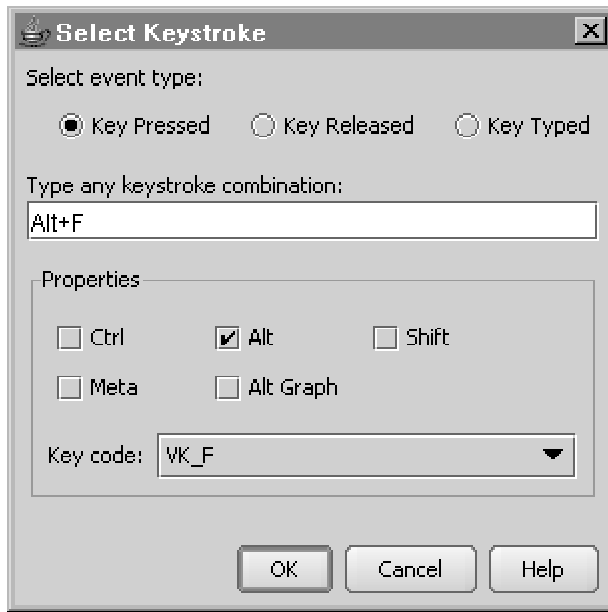
Bring up the JBuilder Licensing dialog with this method.

Several of these methods are demonstrated in the `UISampler` class described at the end of this chapter.

KeyStrokeDialog Class

Asking the user to select a keystroke combination can be easily achieved with the `com.borland.primetime.ui.KeyStrokeDialog` class. It displays a popup dialog box (see Figure 9–8) that lets the user enter a keystroke and select any modification keys, such as Shift, Ctrl, or Alt. It is based around the `KeyStrokeEditorPanel` class described later.

Figure 9–8. A *KeyStrokeDialog*.



UNDOCUMENTED

Borland has not yet documented the `KeyStrokeDialog` class.

Use the dialog as shown in the following code:

```
KeyStrokeDialog dialog =
    new KeyStrokeDialog(frame, "Select Keystroke", true);
dialog.show();
KeyStroke keystroke = dialog.getKeyStroke();
```

The dialog's methods are:

```
public KeyStrokeDialog();
public KeyStrokeDialog(Frame frame, boolean modal);
public KeyStrokeDialog(Frame frame, String title, boolean
    modal);
public KeyStrokeDialog(Component component, String title,
    boolean modal);
```

Create a new dialog box to ask the user for a keystroke combination.

`frame` is the parent frame for the dialog.

`component` is a control whose encompassing frame becomes the parent for the dialog.

`title` is the title of the dialog.

`modal` is true to make the dialog modal, or false to make it non-modal.

```
public KeyStroke getKeyStroke();
```

Retrieve the keystroke selected by the user.

```
public void setDisplayOptions(boolean showEventType,
    boolean showKeyReleased, boolean showKeyTyped);
```

Set all the display options in one go. These make the corresponding controls on the editor panel visible or not.

`showEventType` is true to show the entire event type panel, or false to hide it.

`showKeyReleased` is true to display the key released option, or false to hide it.

`showKeyTyped` is true to show the key typed option, or false to hide it.

```
public void setKeyStroke(KeyStroke keystroke);
```

Set the initial keystroke to display to the user.

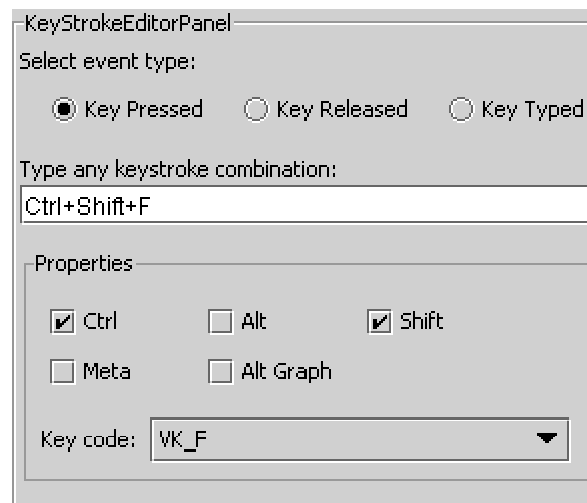
`keystroke` is the keystroke to show.

See the `UISampler` class at the end of this chapter for an example of the dialog's use.

KeyStrokeEditorPanel Class

The `com.borland.primetime.ui.KeyStrokeEditorPanel` class provides for the full entry of a keystroke along with the event that triggers it (see Figure 9-9). This panel forms the main contents of the `KeyStrokeDialog`, and is a wrapper around a `KeyStrokeEditorTextField` object.

Figure 9-9. A KeyStrokeEditorPanel.



UNDOCUMENTED

Borland has not yet documented the `KeyStrokeEditorPanel` class.

The methods of the editor panel are listed below:

```

public KeyStrokeEditorPanel();
public KeyStrokeEditorPanel(KeyStrokeEditor
    keystrokeEditor);
public KeyStrokeEditorPanel(KeyStroke keystroke, boolean
    showKeyReleased, boolean showKeyTyped);
public KeyStrokeEditorPanel(boolean showEventType, boolean
    showKeyReleased, boolean showKeyTyped);
    Create a new keystroke entry panel with one of these constructors.
    keystrokeEditor is an existing editor to use.
    keystroke is the initial keystroke to show.
    showEventType, showKeyReleased, and showKeyTyped are passed
    through to the setDisplayOptions method. setEventType is true if not
    specified.

public static KeyStroke decodeKeyStroke(String
    keystrokeText);
    Convert a text description of a keystroke into a KeyStroke object.
    keystrokeText is the text version of the keystroke.

public static String encodeKeyStroke(KeyStroke keystroke);
    Convert a keystroke into a corresponding text version, such as
    "CTRL+SHIFT+VK_F".
    keystroke is the keystroke to convert.

public String getJavaInitializationString();
    Retrieve the keystroke value as a string suitable for inserting into Java source
    code as an initial value.

public static String getKeyName(int keyCode);
    Convert a key value into a text description using its name, such as
    "VK_COMMA".
    keyCode is the code for the character to convert.

public KeyStroke getKeyStroke();
    Obtain the keystroke entered into this panel.

public static String getKeyStrokeName(KeyStroke keystroke);
public static String getKeyStrokeName(KeyStroke keystroke,
    boolean showEvent);
    Convert a keystroke into a corresponding text description using the key's
    name, such as "Ctrl+Shift+VK_F typed".
    keystroke is the keystroke to convert.
    showEvent is true (the default) to include the event with the text, of false to
    ignore it.

public static String getKeyStrokeText(KeyStroke keystroke);
public static String getKeyStrokeText(KeyStroke keystroke,
    boolean showEvent);
public static String getKeyStrokeText(KeyStroke keystroke,
    boolean showEvent, boolean useText);
    Convert a keystroke into a corresponding text description using the key's
    text, such as "Ctrl+Shift+F typed".
    keystroke is the keystroke to convert.
    showEvent is true (the default) to include the event with the text, of false to
    ignore it.

```

`useText` is true (the default) to display the key text, or false to display the key name.

```
public static String getKeyText(int keyCode);
```

Convert a key into a text description using its text value, such as “,”.

`keyCode` is the code for the character to convert.

```
public KeyStroke getResult();
```

Obtain the keystroke entered into this panel.

```
public String getValueText();
```

Retrieve the keystroke value as text.

```
public static String getVKText(int keyCode);
```

Convert a key into a text description using its name, such as “VK_COMMA”.

`keyCode` is the code for the character to convert.

```
public void resetFocus();
```

Move focus to the keystroke entry field and select all of its contents.

```
public void setDisplayOptions(boolean showEventType,
    boolean showKeyReleased, boolean showKeyTyped);
```

Set all the display options in one go. These values make the corresponding controls visible or not.

`showEventType` is true to show the entire event type panel, or false to hide it.

`showKeyReleased` is true to display the key released option, or false to hide it.

`showKeyTyped` is true to show the key typed option, or false to hide it.

```
public void setKeyStroke(KeyStroke keystroke);
```

Establish the keystroke to show initially.

`keystroke` is the new keystroke value to display.

```
public boolean stopEditing();
```

Call this method to translate the text entered by the user into a keystroke value. It returns true if it can perform the conversion, or false if it cannot or if there is no text to transform.

Look at the `UISampler` class at the end of the chapter for a demonstration of this panel’s use.

KeyStrokeEditorTextField Class

For a text field that allows entry of a keystroke combination and displays it as text, use the `com.borland.primetime.ui.KeyStrokeEditorTextField` class.



UNDOCUMENTED

Borland has not yet documented the `KeyStrokeEditorTextField` class.

```
public KeyStrokeEditorTextField();
```

```
public KeyStrokeEditorTextField(KeyStroke keystroke);
```

Generate a new keystroke entry field.

`keystroke` is the initial value to set.

```
public KeyStroke getKeyStroke();
```

Retrieve the entered keystroke.

```
public boolean getShowEventType();
public boolean getShowKeyReleased();
public boolean getShowKeyTyped();
```

Get the current settings for these values. They all default to true.

```
public void setDisplayOptions(boolean showEventType,
    boolean showKeyReleased, boolean showKeyTyped);
```

Set all the display options in one go. These are used in the editor panel for a keystroke.

`showEventType`, `showKeyReleased`, and `showKeyTyped` are the values for the corresponding options.

```
public void setKeyStroke(KeyStroke keystroke);
```

Establish the current keystroke value.

`keystroke` is the new setting.

```
public void setShowEventType(boolean show);
public void setShowKeyReleased(boolean show);
public void setShowKeyTyped(boolean show);
```

Set the new values for these settings. These are used in the editor panel for a keystroke.

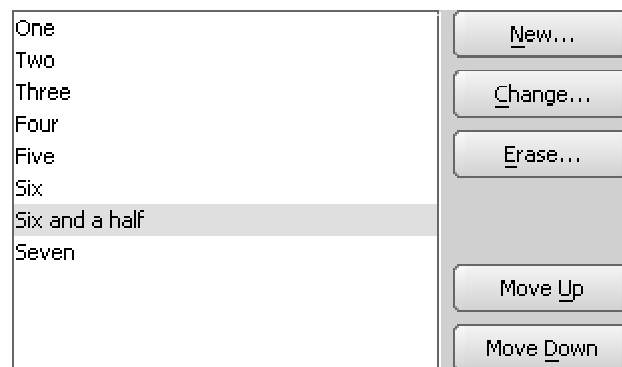
`show` is the new setting value.

For an example of this field, see the `UISampler` class at the end of this chapter.

ListPanel Class

Managing lists of items in a GUI is easily achieved by using the `com.borland.pruntime.ui.ListPanel` class. This abstract class provides for the display of the list, the movement of items in the list, adding and removing items, and editing items, as can be seen in Figure 9–10. You need to override at least the `editElement` and `promptForElement` methods in your subclass. The rest of the functionality is already supplied.

Figure 9–10. A *ListPanel*.



UNDOCUMENTED

The `ListPanel` class is undocumented.

Its methods are shown below:

```

public ListPanel();
public ListPanel(boolean border);
public ListPanel(String addText, String editText, String
    removeText);
public ListPanel(String addText, char addMnemonic, String
    editText, char editMnemonic, String removeText, char
    removeMnemonic);

```

Create a new list panel with one of these constructors. Settings for the buttons on the panel may be supplied, with appropriate defaults being used otherwise. I believe the button labels are localized, but have not been able to confirm this. The defaults given below apply to English.

`border` is true (the default) to have an empty border around the panel, or false to have no border.

`addText` is the label for the Add button, defaulting to “Add...”. Passing a null for this, or one of the other labels, simply sets the button label to null, but leaves the button visible and enabled.

`addMnemonic` is the accelerator character for the Add button. It defaults to ‘A’ if not specified.

`editText` is the label for the Edit button, defaulting to “Edit...”.

`editMnemonic` is the accelerator character for the Edit button. If not specified it defaults to ‘E’.

`removeText` is the label for the Remove button, defaulting to “Remove...”.

`removeMnemonic` is the accelerator character for the Remove button. It defaults to ‘R’ if not specified.



VERSION

The `ListPanel` constructor version that takes a single boolean value is not available in JBuilder 7.

```

public void addChangeListener(ChangeListener listener);
    Register an object to be informed of changes to the list.
    listener is the object to notify of changes.
public void addListElement();
    Ask the user for a new item to add to the list, via the promptForElement
    method, and then add it.
protected void addListElement(Object object);
    Add the specified object to the end of the list.
    object is the new list element. If this is an array, then each item in the array
    is added separately.
protected boolean canAdd();
    Determine whether a new item can be added to the list. The default version
    always returns true, but this could be overridden in your subclass.
protected boolean canEdit(int index);
    Decide whether an item can be modified. This class always returns true,
    although you can modify this by overriding it in your descendent.

```

```
protected boolean canMoveDown(int index, int count);
protected boolean canMoveUp(int index, int count);
```

Control whether an item can be moved down or up within the list. Usually these return true if not at the end or start of the list respectively. Override this behavior if you want something different.

`index` is the current position of the item within the list.

`count` is the total number of items in the list.

```
protected boolean canRemove(int index);
```

Confirm that an item can be deleted from the list. All items may be removed by default, however you could change this in your subclass.

`index` is the position of the item to be deleted.

```
protected void doubleClickElement(Object object);
```

By default, double-clicking an item causes it to be edited. Change this behavior by overriding this method in your subclass.

`object` is the selected item.

```
protected abstract Object editElement(Object object);
```

You must override this method in your subclass to provide a meaningful way of altering an item from the list. Return the updated item for replacement in the list, or `null` to cancel the operation.

`object` is the item to edit.

```
public void editSelectedListElement();
```

This method calls the previous one for the currently selected item.

```
public void enableControls(boolean enabled);
```

Enable or disable all the controls in the panel through this method.

`enabled` is true to enable the controls, or false to disable them.

```
protected JButton getAddButton();
```

```
protected JButton getEditButton();
```

Get a reference to the named button on the panel.

```
protected String getElementName(Object object);
```

Retrieve the name of the given element. This returns the object's `toString` value unless overridden.

`object` is the item to be named.

```
public ArrayList getList();
```

Obtain a reference to the entire list of items.

```
public Component getListCellRendererComponent(JList list,
Object value, int index, boolean isSelected, boolean
cellHasFocus);
```

Return the component that renders the items within the list control. This returns the `defaultListCellRenderer` object.

```
protected JScrollPane getListScrollPane();
```

```
protected JButton getMoveDownButton();
```

```
protected JButton getMoveUpButton();
```

```
protected JButton getRemoveButton();
```

Get a reference to the scroll pane or to the named button on the panel.

```
public int getSelectedIndex();
```

```
public int[] getSelectedIndices();
```

Retrieve the indexes of one or all of the selected entries. The methods return `-1` or an empty array if nothing is selected.

**VERSION**

The `getSelectedIndex` and `getSelectedIndices` methods are only available in JBuilder 9 and 10.

```
public Object getSelectedListElement();
public Object[] getSelectedListElements();
```

Find the currently selected item(s) within the list, or `null` or an empty array if no item has been chosen.

```
public ListSelectionModel getSelectionModel();
```

Get the selection model underlying the list panel.

**VERSION**

The `getSelectionModel` method is only available in JBuilder 9 and 10.

```
public void moveSelectedListElement(int offset);
```

Move the current item within the list.

`offset` is the distance from the current position to move, with negative numbers indicating movement up the list. If the `offset` places the item outside the bounds of the list, nothing happens.

```
protected abstract Object promptForElement();
```

You must override this method to create a new item. Return that item as the result of the method call, or `null` to cancel the action.

```
public void removeChangeListener(ChangeListener listener);
```

Delete a previously registered listener for changes in the list.

`listener` is the object to remove from the list of listeners.

```
public void removeSelectedListElements();
```

Delete all the selected items from the list.

```
public void selectValue(Object object);
```

Establish which item is the current one.

`object` is the item to select.

```
public void setAddButtonVisible(boolean visible);
```

Control the visibility of the Add button with this method.

`visible` is `true` to show the button or `false` to hide it.

```
public void setEditButtonVisible(boolean visible);
```

Control the visibility of the Edit button with this method.

`visible` is `true` to show the button or `false` to hide it.

```
public void setEnabled(boolean enabled);
```

Enable or disable the entire component.

`enabled` is `true` to enable all the controls, or `false` to disable them.

```
public synchronized void setList(ArrayList list);
```

Load the contents of the list control from the supplied list. Any existing items are deleted.

`list` is the list of items to be added.

```
public void setMoveButtonsVisible(boolean visible);
```

Control the visibility of the Move buttons with this method.

`visible` is `true` to show the buttons or `false` to hide them.

A single field is also available for use by a subclass:

```
protected DefaultListCellRenderer defaultListCellRenderer;
```

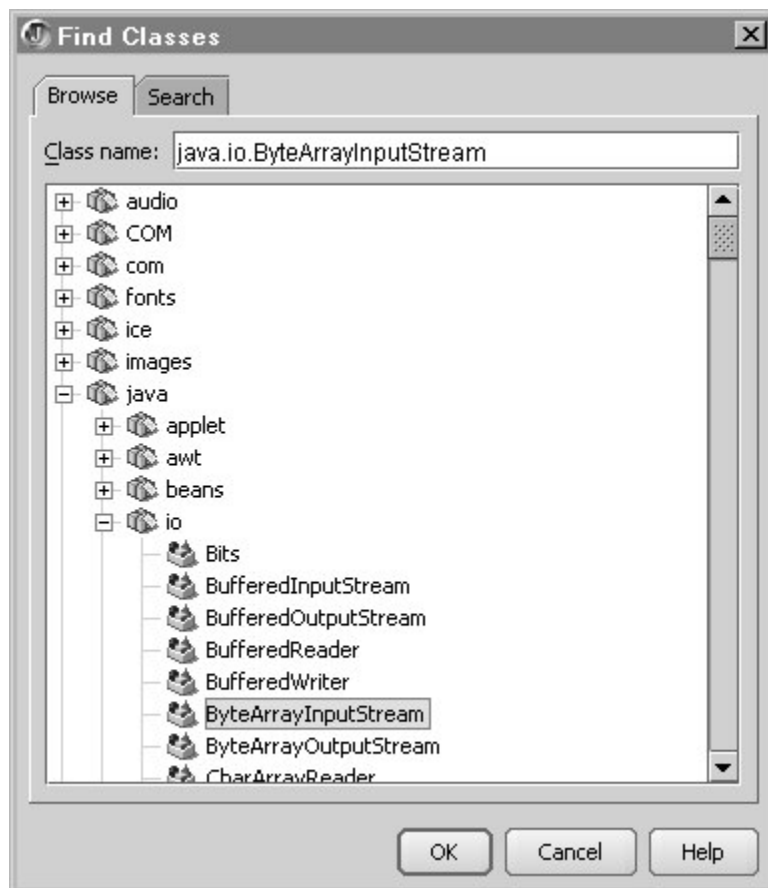
Access the renderer used for the items in the list through this field.

See the `UISampler` class at the end of the chapter for an example of the panel's use. Also, the `OpenTools` runner example in Chapter 26 makes use of a `ListPanel`.

PackageBrowserDialog Class

Searching for a class is made easier with the `com.borland.jbuilder.ide.PackageBrowserDialog` class, which builds on the `DefaultDialog` class. In fact, JBuilder uses this for the **Browse Classes** action as shown in Figure 9–11. You need to create a `PackageBrowserTree` instance for it to work with, or you can call the static methods on `PackageBrowserTree` to build the tree and then display the dialog. See the next section for more details.

Figure 9–11. A `PackageBrowserDialog`.



UNDOCUMENTED

This class is undocumented in JBuilder.

Its methods are shown below:

```
public PackageBrowserDialog(Component component, String
    title, PackageBrowserTree tree, boolean[] flags);
public PackageBrowserDialog(Component component, String
    title, PackageBrowserTree tree, boolean[] flags, String
    caption);
```

Create a new browser dialog.

`component` is a component whose frame becomes the owner of this dialog.

`title` is the caption for the dialog window.

`tree` is the tree that contains the class hierarchy.

`flags` is updated and returned. The first entry in the array is set to true if a package is selected, or false if the dialog is cancelled. You should pass in a boolean array with a single element.

`caption` is the text to display before the text box on the Browse tab. It defaults to "Class name:".



VERSION

The constructor that takes the `caption` parameter is not available in JBuilder 7.

```
public void closeDialog();
```

Close the dialog and dispose of it.

```
public boolean getAllowPackages();
```

Returns true if packages may be selected along with classes, or false if only classes can be chosen.

```
public boolean isAddToProjectEnabled();
```

Determine whether the Add to Project checkbox is enabled.

```
public boolean isAddToProjectSelected();
```

See whether the Add to Project checkbox is selected. It only returns true if the checkbox is visible as well as selected.

```
public boolean isAddToProjectVisible();
```

Find out whether the Add to Project checkbox is visible to the user.

```
public boolean isClassComboVisible();
```

Determine whether the combobox of previously selected class names is visible.

```
public void setAddToProjectEnabled(boolean enabled);
```

Enable or disable the Add to Project checkbox.

`enabled` is true to enable the checkbox, or false to disable it.

```
public void setAddToProjectSelected(boolean selected);
```

Set the initial state of the Add to Project checkbox.

`selected` is true to make it checked, or false to clear it.

```
public void setAddToProjectVisible(boolean visible);
```

Show or hide the Add to Project checkbox.

`visible` is true to display the checkbox, or false to hide it.

```
public void setAllowPackages(boolean allowPackages);
```

Establish what may be selected from the class hierarchy.

`allowPackages` is true to let the user select a package or a class, or false to only allow classes to be chosen.

```
public void setClassComboVisible(boolean visible);
```

Alter the visibility of the combobox that displays previously selected classes.

visible is true to have it appear, or false to remove it.

```
public static String showClassBrowserDialog(Component
    component, String title, PackageBrowserTree tree);
public static String showClassBrowserDialog(Component
    component, String title, PackageBrowserTree tree,
    boolean allowPackages);
public static String showClassBrowserDialog(Component
    component, String title, PackageBrowserTree tree,
    boolean allowPackages, String caption);
```

Use these static methods to easily bring up the dialog and return a selection. A null is returned if the dialog is cancelled. You can also use the static methods on the `PackageBrowserTree` class.

`component` is a component whose frame becomes the owner of this dialog.

`title` is the caption for the dialog window.

`tree` is the tree that contains the class hierarchy.

`allowPackages` is true to let the user select a package or a class, or false (the default) to only allow classes to be chosen.

`caption` is the text to display before the text box on the Browse tab. It defaults to "Class name:".



VERSION

The `showClassBrowserDialog` method that takes the `caption` parameter is not available in JBuilder 7.

```
public void showHelp();
```

Call up the help topic for this dialog.

```
public static String showPackageBrowserDialog(Component
    component, JBProject project);
public static boolean showPackageBrowserDialog(Component
    component, String title, PackageBrowserTree tree);
```

Display the dialog to select a package only. The first version returns the name of the package, while the latter returns true if a package is chosen, or false if the dialog is cancelled. For the latter, you would then have to query the tree for the selected package(s).

`component` is a component whose frame becomes the owner of this dialog.

`project` is the project to use for finding packages to be displayed.

`title` is the caption for the dialog window.

`tree` is the tree that contains the class hierarchy.

PackageBrowserFilter Interface

The `com.borland.jbuilder.ide.PackageBrowserFilter` interface allows you to select which nodes are shown in a `PackageBrowserTree` object.



UNDOCUMENTED

This interface is undocumented in JBuilder.

Its one method is:

```
public Node[] packageFilter(Node[] nodeArray);
```

Given an array of nodes, remove those that do not apply and return the result.

The `PackageOnlyPackageBrowserFilter` class in the same package provides a filter that selects only packages.

PackageBrowserTree Class

Searching for a class is made easier with the `com.borland.jbuilder.ide.PackageBrowserTree` class. It provides a tree view of the class hierarchy and extends `SearchTree` (see below) so that you can look for items by typing within the tree control. See Figure 9–12 in the `PackageBrowserDialog` class above for an example.



UNDOCUMENTED

This class is undocumented in JBuilder.

To use this class to have the user select a known class, you could use code like the following:

```
String className = PackageBrowserTree.browseClass(
    (JBProject)Browser.getActiveBrowser().getActiveProject(),
    null, "Find a Class", null);
if (className != null && className.length() > 0) {
    // Do something with it
}
```

The new methods of this class are shown below:

```
public PackageBrowserTree();
public PackageBrowserTree(Project project);
public PackageBrowserTree(Project project,
    PackageBrowserFilter filter);
public PackageBrowserTree(Project project,
    PackageBrowserFilter filter, int mode);
```

Create a new tree control that displays classes from a project's classpath.

`project` is the project whose classpath is used to locate classes. If not specified here you must use the `setProject` method to set it before using this tree.

`filter` is a filter that selects which entries in the tree are displayed.

`mode` is the mode of the package nodes that make up the tree. Use the `MODE_*` constants from the `JBProject` class (see Chapter 6).

```
public static String browseClass(JBProject project,
    Component component, String title, String initialPath);
public static String browseClass(JBProject Project,
    Component component, String title, String initialPath,
    boolean needExistingClass);
public static String browsePackageOrClass(JBProject
    project, Component component, String title, String
    initialPath);
```

Display a dialog box that lets the user search for a class, or a package, anywhere on a particular classpath. The return value is the full name of the class or package selected, or `null` if the dialog was cancelled.

`project` is the project to use to determine the classpath to search.

`component` is a component whose frame becomes the owner for this dialog. It may be `null` if there is no owner.

`title` is the caption for the dialog box.

`initialPath` is the class or package to display initially. If this is `null` or an empty string, the last class found is used.

`needExistingClass` is `true` (the default) if an existing class must be chosen, or `false` if a new name may be entered.



VERSION

The `browseClass` method that takes the `needExistingClass` parameter is only available in JBuilder 9 and 10.

```
public TreePath getFarthestPath(String path);
```

Find the tree path that matches or is just past the given path and return it. See the same method in `SearchTree` for more details and an example.

`path` is the full path to attempt to follow.



VERSION

The `getFarthestPath` method is only available in JBuilder 9 and 10.

```
public String getFullPath(TreePath path);
```

```
public String getFullPath(TreePath path, boolean  
    allPackage);
```

Return the full name for the given path, or an empty string if the path is `null`.

`path` is the path to retrieve.

`allPackage` is `true` to append “`.*`” if a package is selected, or `false` (the default) to omit it.

```
public int getMode();
```

Retrieve the current mode of the packages within the tree. It should be one of the `MODE_*` constants from `JBProject`.

```
public PackageBrowserFilter getPackageBrowserFilter();
```

Return the current filter applied to this tree, or `null` if none has been specified.

```
public Node getPathNode(TreePath path);
```

Retrieve the node at the end of a path.

`path` is the path to follow to find the node.

```
public Node[] getPathNodes(TreePath[] paths);
```

Return the list of nodes corresponding to the given paths.

`paths` is the list of paths to convert into nodes.

```
public Project getProject();
```

Find the project associated with this browse tree.



VERSION

The `getProject` method is not available in JBuilder 7.

```
public Node getSelectedNode();
```

Get the currently selected node from the tree, or `null` if no node has been chosen.

```
public Node[] getSelectedNodes();
```

Obtain a list of all the selected nodes from the tree, or an empty array if none were picked.

```

public String getSelectedPath();
public String getSelectedPath(boolean allPackage);
    Retrieve the full name of the currently selected package or class, or null if
    no path was chosen.
    allPackage is true to append “.*” if a package is selected, or false (the
    default) to omit it.
public String getSelectedPaths(boolean allPackage);
    Retrieve the full names of all the currently selected packages or classes, or an
    empty array if none have been chosen.
    allPackage is true to append “.*” if a package is selected, or false (the
    default) to omit it.
public boolean isClassNameInTree(String className);
    Determine whether a particular class is present in the tree, returning true if it
    is, or false if it is not.
    className is the full name of the class to find, like “java.util.
    Vector”.
public boolean isPackageName(String packageName);
    Discover whether a certain package is in the tree, returning true if it is, or
    false if it is not.
    packageName is the full name of the package to check, such as “java.
    util”.
public void setMode(int mode);
    Alter the mode of the packages in the tree.
    mode is the new mode. It should be one of the MODE_* constants from
    JBProject (see Chapter 6)
public void setPackageBrowserFilter(PackageBrowserFilter
    filter);
    Modify the filter applied to items in the tree.
    filter is the new filter to apply to the tree, or null to show all entries.
public void setProject(Project project);
    Set the project to scan for packages.
    project is the project used to locate packages.
public void setSelectedPath(String path);
    Establish the current path.
    path is the full path to the required package or class.

```

PathSet Class

Managing a collection of paths, such as for a library definition within JBuilder, is the job of the `com.borland.jbuilder.paths.PathSet` class. It keeps track of the paths for the source, class, and documentation files, and any other libraries that it depends on. Although the class has constructor and update methods, you usually just retrieve an existing instance and query it for information:

```

PathSet library = (PathSet)PathSetManager.getLibraries().get(0);
Url[] classPath = library.getClassPath();

```

The methods of this class are:

```
public PathSet(String name);
```

Create a new set of paths for a library.

`name` is the name for this set – the library name. It must not be `null`.

```
public synchronized int addEntries(Url url);
```

Scan the supplied location recursively (a directory or a JAR file) for candidates as source, class, or documentation (Javadoc) paths. Source paths contain `.java` files, class paths contain `.class` or `.jar` files, and document paths contain `index-all.html`. These are added to the appropriate internal lists. The return value is the number of directories to back up to continue searching. If this is zero you resume your search in the current directory. If it is one you should move to the parent of the current directory before continuing, etc.

`url` is the starting location for the scan.

```
public static void addUniquePath(List list, Url url);
```

```
public static Url[] addUniquePath(Url[] oldPath, Url url);
```

Add a given path (`Url`) to a list or array of paths, but only if it is not already there. The list is updated, or an amended array of paths is returned.

`list` is the list of paths to examine and add to.

`oldPath` is the array of paths to use as the basis for adding the new path.

`url` is the new path to add, when appropriate.



VERSION

The `addUniquePath` methods are protected and non-static in JBuilder 7.

```
public static void addUniquePaths(List list, PathSet[]
    pathSets);
```

```
public static void addUniquePaths(List list, Url[] urls);
```

```
public static void addUniquePathsIfEnabled(List list, Url[]
    urls);
```

Update a list of paths to include an array of `Urls` or `PathSets`, but only if they are not already present. The `addUniquePathsIfEnabled` method imposes a further restriction – that the new paths not be disabled by the current edition of JBuilder (Enterprise/SE/etc.).

`list` is the list of paths to examine and add to.

`pathSets` is the array of path sets to include, when appropriate.

`url` is the array of new paths to add, when appropriate.



VERSION

The `addUniquePaths*` methods are protected and non-static in JBuilder 7.

```
public synchronized void delete();
```

Delete this library's file from permanent storage and also from its collection.

```
public synchronized Url[] getClassPath();
```

Return the list of locations that make up the class path for this library.

```
public synchronized PathSetCollection getCollection();
```

Obtain a reference to the collection that this library is a member of.

NOTE

The `PathSetCollection` class is not covered here, but it encapsulates the groupings within the Configure Libraries and Configure JDKs dialogs, such as Project, User Home, and JBuilder. From these objects you can find out which libraries and JDKs it manages, and update these lists.

```
public PathSet getCopy(PathSetCollection collection);
    Copy this set of paths and return a reference to it.
    collection is the collection to add the new set of paths to.
public synchronized Url[] getDocPath();
    Return the list of locations that make up the documentation path for this
    library, such as file:///C%|/JBuilder/OpenTools/Javadoc/doc.
public synchronized String getEmptyDescription();
    Get the description for an empty library.
```

VERSION

The `getEmptyDescription` method is not available in JBuilder 7.

```
public synchronized Url[] getFullClassPath();
public synchronized Url[] getFullDocPath();
public synchronized Url[] getFullSourcePath();
    Combine the appropriate set of paths from this library with the corresponding
    ones from all of its required libraries and return the entire collection. For
    example, the full class path may include:
    file:///C%|/JBuilder/OpenTools/Javadoc/class
    zip:///C%|/JBuilderX/lib/jbuilder.jar
public String getFullName();
    Retrieve the name of this library. It appears to be identical to getName.
public Icon getIcon();
    Obtain an icon for this library.
public synchronized String getIncompleteDescription()
    Get the description for an incomplete library, typically "Library is
    incomplete".
public long getLastModificationSaved();
    Find out when this path set was last modified, returning a standard timestamp
    value (which may be zero).
```

VERSION

The `getLastModificationSaved` method is only available in JBuilder 9 and 10.

```
public long getLastModified();
    Contrary to what the documentation says, this value is not a timestamp, but is
    a sequential number starting from zero that gets incremented each time
    significant changes are made to the path set. Modifications to the full paths
    returned by this library may occur through objects other than this one.
public synchronized LibKit getLibKit();
    Obtain a reference to the library kit for this path set. The returned class is not
    covered in this book.
```

VERSION

The `getLibKit` method is not available in JBuilder 7.

```
public synchronized String getName();
```

Retrieve the name of this library.

```
public Class getPathSetReferenceClass();
```

Get a reference to the PathSet class.

```
public TreeMap getProperties();
```

```
public String getProperty(String category, String name);
```

```
public String getProperty(String category, String name,
    String defaultValue);
```

Find one or more property values for this path set. These are initialized via the `setProperty` method.

`category` is the name of the property category.

`name` is the name of the individual property.

`defaultValue` is the value to use if no other one can be found.



VERSION

The `getProperties` and `getProperty` methods are only available in JBuilder 9 and 10.

```
public synchronized String getReferenceName(PathSet
    pathSet);
```

Supplies the name to use in property values to identify this path set.

Typically this is the same as `getName` returns.



VERSION

The `getReferenceName` method is only available in JBuilder 9 and 10.

```
public synchronized PathSet[] getRequired();
```

```
public synchronized String[] getRequiredNames();
```

Obtain a list of the other libraries required by this one, as either path set objects or just as names. If there are none, an empty array is returned.

```
public synchronized PathSetResolver getResolver();
```

Find the resolver used to look up libraries. This class is not covered in this book.



VERSION

The `getResolver` method is protected prior to JBuilder 10.

```
public synchronized Url[] getSourcePath();
```

Return the list of locations that make up the source path for this library, like `file:///C%|/JBuilder/OpenTools/Javadoc/src`.

```
public Url getUrl();
```

Discover the location of the file where this library definition is stored, such as `file:///C%|/Documents and Settings/keith/.jbuilderX/Project.library`.

```
public synchronized boolean isEmpty();
```

Find out if this library contains any path information. It returns true if no paths have been set or false if any have.

```
public synchronized boolean isEnabled();
```

Determine whether this library is enabled, returning true if all the entries on the class path are enabled (based on the restrictions imposed by JBuilder editions, such as the JBCL classes not being available in the Personal version), or false otherwise.

```
public synchronized boolean isIncomplete();
```

Discover whether this library is incomplete in some way. The default value is false, but this may be overridden in subclasses. Incomplete libraries are grayed out in the UI.

```
public synchronized boolean isReadOnly();
```

Returns true if this library is read-only, or false if it may be updated.

```
public static Url[] pathFromString(String path);
```

```
public static Url[] pathFromString(String path, boolean asArchive);
```

```
public static Url[] pathFromArray(String[] paths);
```

```
public static Url[] pathFromArray(String[] paths, boolean asArchive);
```

```
public static Url[] pathFromArray(String parentPath, String[] paths);
```

```
public static Url[] pathFromArray(String parentPath, String[] paths, boolean asArchive);
```

Convert from string representations of paths to `Url`s.

`path` is a classpath list to separate out and convert.

`parentPath` is the base path to use with the list of (relative) paths.

`paths` is an array of paths to convert. The values are relative paths in the case of the last two versions of this method.

`asArchive` is true (the default) to format the `Url` as a JAR reference when appropriate, or false to leave it as a simple file name.

**VERSION**

The last two versions of the `pathFromArray` method are not available in JBuilder 7.

```
public static String pathToString(Url[] path);
```

```
public static String pathToString(Url[] path, boolean resolvePaths);
```

```
public static String[] pathToArray(Url[] path);
```

```
public static String[] pathToArray(Url[] path, boolean resolvePaths);
```

Convert an array of path `Url`s into their string versions – either as a single classpath, or as an array of file names.

`path` is the array of `Url`s to convert.

`resolvePaths` is true to resolve paths to their platform-specific representation, or false to use the generic format. For example, when true on a Windows platform, the path “/JBuilder/OpenTools” becomes “C:\JBuilder\OpenTools”.

**VERSION**

The method versions above that take the `resolvePaths` parameter are not available in JBuilder 7.

```
public void resetFullPaths();
```

Reset the internal timestamp to force a full update.

```
public synchronized void save();
```

Save the definitions for this library to permanent storage.

```
public synchronized void setClassPath(Url[] classPath);
```

Alter the list of paths that make up the class path for this library.

`classPath` is the new set of locations.

```
public synchronized void setCollection(PathSetCollection
collection);
```

Change the collection to which this library belongs.

`collection` is the new collection.

```
public synchronized void setDocPath(Url[] docPath);
```

Modify the list of paths that make up the documentation path for this library.

`docPath` is the new set of locations.

```
public void setFromCopy(PathSet pathSet);
```

Copy another `PathSet`'s details to this one.

`pathSet` is the set of paths to copy.

```
public synchronized void setLibKit(LibKit libKit);
```

Establish the library kit to use.

`libKit` is the new library kit. This class is not covered in this book.



VERSION

The `setLibKit` method is not available in JBuilder 7.

```
public synchronized void setName(String name);
```

Update the library's name.

`name` is the new name. It must not be null.

```
public void setProperty(String category, String name,
String value);
```

Set a property value for this path set.

`category` is the name of the property category.

`name` is the name of the individual property.

`value` is the value to use.



VERSION

The `setProperty` method is only available in JBuilder 9 and 10.

```
public synchronized void setRequired(String required);
public synchronized void setRequired(String[] required);
```

Change the list of other libraries required by this one.

`required` is one or more names of other libraries.

```
public synchronized void setSourcePath(Url[] sourcePath);
```

Modify the list of paths that make up the source path for this library.

`sourcePath` is the new set of locations.

```
public synchronized void setUrl(Url storagePath);
```

Establish the location where this library definition is saved.

`storagePath` is that location.

A single public field is defined in the class:

```
public static final PathSet EMPTY_ARRAY[];
```

This is the return value for appropriate methods when there are no paths of the requested type.

The `JDKPathSet` and `ProjectPathSet` classes extend this class to provide additional support for defining JDKs and projects.

Platform Class

When it becomes necessary to identify which platform your tool is running on, you can use the abilities of the `com.borland.primetime.util.Platform` class to help you out.



UNDOCUMENTED

This class has not yet been documented.

It provides two methods:

```
public static boolean isMacLAF();
```

Returns true if the Mac look-and-feel is in use, or false if it is not.

```
public static boolean isSpecialDown(MouseEvent mouseEvent);
```

Determine whether a mouse event is a special case on this platform (it occurs while a particular key is pressed), returning true if it is, or false if it is not.

`mouseEvent` is the mouse event to examine.

The following constants are also available:

```
public static final boolean IBM_LINUX;
```

True if running IBM's version of Linux, false otherwise.

```
public static final boolean IBM_VENDOR;
```

True if running under IBM's JVM, false otherwise.

```
public static final boolean LINUX;
```

True if running under Linux, false otherwise.

```
public static final boolean MAC;
```

True if running on a Macintosh, false otherwise.

```
public static final String OS_NAME;
```

The name of the operating system under which the tool is running.

```
public static final boolean SOLARIS;
```

True if running under Solaris, false otherwise.

```
public static final boolean STANDALONE_DDEDITOR;
```

True if running the stand-alone deployment descriptor editor, false otherwise.

```
public static final boolean UNIX;
```

True if running under some form of Unix, false otherwise.

```
public static final boolean WIN32;
```

True if running under Windows, false otherwise.

Several of these fields are used within the `UISampler` class described at the end of this chapter.

ProjectPathSet Class

The `com.borland.jbuilder.paths.ProjectPathSet` class provides more detailed information about the paths associated with a project, compared to the `PathSet` class it derives from. Although constructor and update methods are available, normally you would just retrieve an instance from a `JBuilder` project and query it for information:

```
ProjectPathSet paths = jbproject.getPaths();
Url backup = paths.getBakPath();
```

**UNDOCUMENTED**

The `ProjectPathSet` class was not documented until JBuilder 9.

Its additional methods are:

```
public ProjectPathSet(JBProject project);
```

Create a new path set for a project. Normally you just retrieve the one supplied by each JBuilder project.

`project` is the JBuilder project to report on.

```
public synchronized void addProjectLibrary(PathSet library);
```

Add a project-specific library to the path set.

`library` identifies the project to add.

**VERSION**

The `addProjectLibrary` method is only available in JBuilder 9 and 10.

```
public synchronized Url[] getAuxPath(String pathId);
```

```
public synchronized Map getAuxPaths();
```

Returns one or all of the auxiliary path entries for this path set.

`pathId` identifies the set of entries to retrieve.

**VERSION**

The `getAuxPath` and `getAuxPaths` methods are only available in JBuilder 10.

```
public synchronized Url getBakPath();
```

Find the directory used for backups of files from a project, like `file:///C%|/JBuilder/OpenTools/Javadoc/bak`.

```
public synchronized Url getDefaultSourcePath();
```

Returns the location established by `setDefaultSourcePath`. If this has not been set then it returns first entry in the source path, or the default ("src") if there are none.

```
public synchronized PathSetCollection[] getFullLibPath();
```

Retrieve the complete set of library collections through this method. This would include the top-level nodes from the Configure Libraries dialog: Project, User Home, and JBuilder.

```
public boolean getIncludeTestPath();
```

Discover whether or not the test path is included in the project's paths.

```
public synchronized JDKPathSet getJDK(String name);
```

Locate a particular JDK reference, or the default one (first in the list) if the given name cannot be found. The returned class derives from `PathSet` but is not otherwise covered in this book.

`name` is the name of the JDK to find.

```
public synchronized JDKPathSet getJDKPathSet();
```

Get the default (first in the list) JDK reference.

```
public synchronized ArrayList getJDKs();
```

Retrieve the JDK references known to this project. Each item in the returned list is a `JDKPathSet` object.

```
public synchronized LibKit[] getLibKits(Class classRef);
```

Return the list of library kits associated with a particular class. The returned class is not covered in this book.

classRef is the class to find the library kits for.

**VERSION**

The `getLibKits` method is not available in JBuilder 7.

```
public synchronized Url getLibPath();
```

Find the location of the base for this project, such as `file:///C%|/JBuilder/OpenTools/Javadoc`.

```
public synchronized ArrayList getLibraries();
```

Discover all the available libraries. Each item in the returned list is a `PathSet` object.

```
public synchronized PathSet getLibrary(String name);
```

Find a reference to a specific library.

name is the name of the library to locate.

```
public String getName();
```

Obtain the name for this path set (based on its `Url`).

**VERSION**

The `getName` method is not available in JBuilder 7.

```
public synchronized Url getOutPath();
```

Get the output directory for the compiled class files, like `file:///C%|/JBuilder/OpenTools/Javadoc/classes`.

```
public static long getPathTime();
```

Retrieves the last modification time for the project's paths.

**WARNING**

The `getPathTime` method has been deprecated as the `PathSetManager` now takes care of this functionality.

```
public synchronized PathSetCollection
getProjectLibraries();
```

Returns the collection of path sets that reflect the libraries for this project.

**VERSION**

The `getProjectLibraries` method is not available in JBuilder 7.

```
public synchronized Url[] getResourcePath();
```

Provides the set of source paths for the project plus any auxiliary paths.

**VERSION**

The `getResourcePath` method is only available in JBuilder 10.

```
public synchronized Url getTestPath();
```

Get the location of the test path for this project, like `file:///C%|/JBuilder/OpenTools/Javadoc/test`.

```
public synchronized Url getWorkingDirectory();
```

Retrieve the working directory for this project, like `file:///C%|/JBuilder/OpenTools/Javadoc`.

```
public boolean putClassOnFullPath(String className);
public boolean putClassOnFullPath(String className, String
    libraryName);
```

Add the path for a particular class to the required list for this path set, if it is not already there. The method returns true if the path was actually added, or false if it was already present.

`className` is the full name of the class to add.

`libraryName` is the name of the library to make the first one in the path. The path order is not changed if this is null (the default).



VERSION

The second `putClassOnFullPath` method is not available in JBuilder 7.

```
public synchronized void reloadLibraries();
```

Discard any existing library definitions and re-read them from permanent storage.

```
public synchronized void setAuxPath(String pathId, Url[]
    auxPath);
```

Update the set of auxiliary paths for this project.

`pathId` is the identifier for this path.

`auxPath` is the set of `Urls` for this auxiliary path.



VERSION

The `setAuxPath` method is only available in JBuilder 10.

```
public synchronized void setBakPath(Url bakPath);
```

Alter the directory used for storing backups of files changed in the project.

`bakPath` is the new directory.

```
public synchronized void setDefaultSourcePath(Url url);
```

Establish the location of the default source path.

`url` is the location to use.

```
public synchronized void setFullLibPath(
    PathSetCollection[] collections);
```

Update the complete set of library collections, as is found in the top-level nodes from the Configure Libraries dialog: Project, User Home, and JBuilder.

`collections` is the list of collections to add.

```
public void setIncludeTestPath(boolean include);
```

Update whether or not the test path is included in the project's path.

`include` is true (the default) to include the test path, or false to exclude them.

```
public synchronized void setJDKPathSet(JDKPathSet jdkPath);
```

Alter the JDK associated with this project.

`jdkPath` is the new JDK reference.

```
public synchronized void setJDKs(ArrayList jdks);
```

Change the locations of the JDKs for this project.

`jdks` is a list of `PathSets` for the JDKs.

```
public synchronized void setLibPath(Url libPath);
```

Modify the base path for the project.

`libPath` is the new path.

```
public synchronized void setLibraries(ArrayList libraries);
```

Update the list of libraries for this project.

libraries is a list of PathSets for the new libraries.

```
public synchronized void setOutPath(Url outPath);
```

Change the directory to which compiled classes are written.

outPath is the new directory.

```
public synchronized void setTestPath(Url url);
```

Establish the location of the test path for this project.

url is the test path location.

```
public synchronized void setWorkingDirectory(Url workDir);
```

Alter the working directory for the project.

workDir is the new directory.

RegularExpression Class

The `com.borland.primetime.util.RegularExpression` class provides simple regular expression handling. It only knows about three special characters: “*” matches with the shortest possible sequence of characters, including none, “?” matches with any single character, and “\” escapes the following character, allowing these three to be matched exactly.

The following example shows how a `RegularExpression` object is used to filter a list of file.

```
File dir = new File("c:/JBuilderX/jdk1.4/bin");
File[] files = dir.listFiles();
RegularExpression regexp = new RegularExpression("java*.exe");
System.out.println("Java files in " + dir.getAbsolutePath());
for (int index = 0; index < files.length; index++) {
    if (regexp.exactMatch(files[index].getName())) {
        System.out.println(" " + files[index].getName());
    }
}
```

The class’ abilities are described below:

```
public RegularExpression(String pattern);
```

```
public RegularExpression(String pattern, boolean
    caseSensitive);
```

Create a new expression to test against. The break on new line and pattern match attributes are both set to true. See their `set` methods below for their meanings.

pattern is the string of characters to be matched, including any special characters described above.

caseSensitive is true (the default) if the pattern is case-sensitive, or false if it ignores case when matching.

```
public boolean exactMatch(String test);
```

```
public boolean exactMatch(char[] test);
```

```
public boolean exactMatch(char[] test, int startIndex, int
    endIndex);
```

Determine whether the pattern matches the given value in its entirety, returning true if matched and false if not.

test is the value to check against the pattern, either as a string or an array of characters.

`startIndex` is the position in the array to start matching at.

`endIndex` is the position after the position in the array to stop at.

```
public MatchResult findSubstringMatch(String test);
public MatchResult findSubstringMatch(char[] test, int
    startIndex, int endIndex);
```

Locate the position in the given value that matches with this pattern. It returns a `MatchResult` object that describes the match if found, or `RegularExpression.NO_MATCH` if not found.

`test` is the value to check against the pattern, either as a string or an array of characters.

`startIndex` is the position in the array to start matching at.

`endIndex` is the position after the position in the array to stop at.

```
public int getLength();
```

Get the length of the expression.



VERSION

The `getLength` method is only available in JBuilder 10.

```
public boolean isBreakOnNewline();
```

Find out whether the pattern allows newline characters to match with the “*” pattern. It returns true if newline characters do match, or false if they do not.

```
public boolean isCaseSensitive();
```

Return whether or not this pattern is case-sensitive.

```
public boolean isPatternMatch();
```

Discover whether any special characters are in fact treated specially. It returns true if special characters retain their pattern meanings, or false if all characters are treated as literal values.

```
public boolean isRegExpMatch();
```

Find out how wildcard characters and escape sequences are treated. When true, they are evaluated as regular expression values according to the JDK 1.4 regexp engine, but when false (the default) they are treated as literal characters.



VERSION

The `isRegExpMatch` method is not available in JBuilder 7.

```
public static boolean isSpecialChar(char character);
```

Determine whether a particular character has a special meaning within a pattern and so needs to be escaped to match it literally. It returns true if the character is special and false otherwise.

`character` is the value to test for a special meaning.

```
public boolean prefixMatch(String test);
```

```
public boolean prefixMatch(char[] test);
```

```
public boolean prefixMatch(char[] test, int startIndex, int
    endIndex);
```

Find out whether the pattern matches a given value at its beginning, returning true if matched and false if not.

`test` is the value to check against the pattern, either as a string or an array of characters.

`startIndex` is the position in the array to start matching at.

`endIndex` is the position after the position in the array to stop at.

```
public void setBreakOnNewline(boolean breakOnNewline);
```

Establish whether the pattern allows newline characters to match with the “*” pattern.

`breakOnNewline` is true (the default) if newline characters do match, or false if they do not.

```
public void setPatternMatch(boolean patternMatch);
```

Set whether any special characters are treated specially.

`patternMatch` is true (the default) to interpret special characters with their pattern meanings, or false to treat all characters as literal values.

```
public void setRegExpMatch(boolean regExpMatch);
```

Control how wildcard characters and escape sequences are treated.

`regExpMatch` is true to evaluate them as regular expression values according to the JDK 1.4 regexp engine, or false to treat them as literal characters.

**VERSION**

The `setRegExpMatch` method is not available in JBuilder 7.

```
public int substringMatch(String test);
public int substringMatch(char[] test);
public int substringMatch(char[] test, int startIndex, int
    endIndex);
```

Check whether the pattern matches any portion of the given value, returning the index where the match was found, or -1 if no part matched.

`test` is the value to check against the pattern, either as a string or an array of characters.

`startIndex` is the position in the array to start matching at.

`endIndex` is the position after the position in the array to stop at.

In addition, the class defines several constants for your use:

```
public static final MatchResult NO_MATCH;
```

The result of a `findSubstringMatch` call that fails to find any match.

```
public static final char CHAR_ANY;
```

The “?” character that matches with any single character.

```
public static final char CHAR_ESCAPE;
```

The “\” character that allows other characters to be interpreted literally.

```
public static final char CHAR_WILDCARD;
```

The “*” character that matches with any number of characters, including none.

RegularExpression.MatchResult Class

The result of a `findSubstringMatch` call on a `RegularExpression` object is an instance of the `com.borland.primetime.util.RegularExpression.MatchResult` class. It encapsulates the location of the matched text within a string or character array. If no match is found the `RegularExpression.NO_MATCH` instance is returned instead.

Use the following methods with this class:

```
public int getLength();
```

Retrieve the length of the matched text.

```
public int getStartIndex();
```

Find the starting position of the matched text.

```
public void setStartIndex(int newIndex);
```

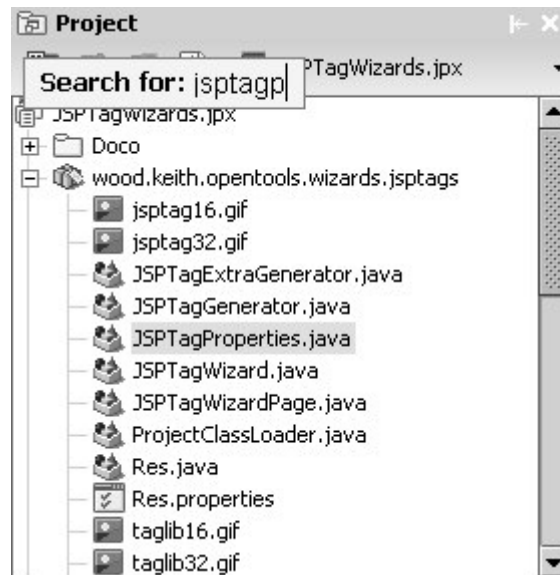
Set the starting position.

`newIndex` is the new starting position.

SearchTree

You may have noticed that the tree components that appear within JBuilder's IDE have the ability to search through their contents. Simply start typing the name of the item you are trying to find when the tree has focus and JBuilder starts scanning. A popup window appears to show what you have typed, while the currently matching item is highlighted within the tree. You can step through all the items that match the current value by pressing the up and down arrow keys. Figure 9-12 shows the searching abilities in action.

Figure 9-12. A *SearchTree*.



These abilities are captured in the `com.borland.primetime.ui.SearchTree` class that extends the Swing `JTree` component. The functionality is based on the assumption that the text displayed in the tree for each node is the same as that returned by the node's `toString` method and will not work correctly if this is not so.



VERSION

In JBuilder 10 *SearchTree* also implements `java.awt.dnd.AutoScroll`.

To add this functionality to your own tools, all you need to do is to use the *SearchTree* class wherever you were using *JTree*.

Additional features of the new class include:

```
public void enableDragDrop(boolean enable);
```

Enable or disable drag-and-drop support for nodes that are instances of `DraggableNode` (not otherwise covered in this book).
`enable` is `true` to enable drag-and-drop support, or `false` to disable it.

**VERSION**

The `enableDragDrop` method is only available in JBuilder 10.

```
public ArrayList getExpansionState();
```

Retrieve a list of all the expanded `TreePaths` in the tree.

```
public TreePath getFarthestPath(String path);
```

```
public TreePath getNearestPath(String path);
```

```
public TreePath getNearestPath(String[] stringPath);
```

Convert a partial path into an entry in the tree. The first method returns the path of the item equal to or immediately after the entered string value, while the second returns that path's parent if not an exact match, or the same path if the string value is a full path already. Case is ignored in the matching process.

For example, in the package/class browser, a path parameter of "java.util.co" returns "java.util.Collection" for the farthest and "java.util" for the nearest paths. Passing "java.util" to these methods returns "java.util" from both.

`path` is the path to start from, with periods (.) separating levels in the tree.

`stringPath` is the path to start from, broken up into separate array elements.

**VERSION**

The `getNearestPath` method that takes an array of strings is only available in JBuilder 10.

```
public int getPreserveMode();
```

Retrieve the current preserve mode setting for the tree. See `setPreserveMode` for more details.

```
public ArrayList getSelectionState();
```

Return the current selections within the tree as an array. Each item in the array is another array, this time of strings, representing the individual levels in the path for each selection.

```
public String[] getStringPath(TreePath path);
```

Separate out the element names on a path into an array.

`path` is the path to separate out.

**VERSION**

The `getStringPath` method is only available in JBuilder 10.

```
public void selectNearestPath(String path);
```

Find the entry in the tree nearest to the given path and select it.

`path` is the path to start from, with periods (.) separating levels in the tree.

**VERSION**

The `selectNearestPath` method is not available in JBuilder 7.

```
public void setDefaultTooltipEnabled(boolean enabled);
```

Indicate whether or not to show the standard `JTree` tooltips – for those nodes whose text is truncated.

`enabled` is `true` to show the tool tips, or `false` (the default) to suppress them.

**VERSION**

The `setDefaultTooltipEnabled` method is only available in JBuilder 10.

```
public void setExpansionState(ArrayList expandList);
```

Alter which nodes in the tree are expanded.

`expandList` is a list of all the `TreePaths` to be expanded in the tree.

```
public void setPreserveMode(int mode);
```

Update the preserve mode for the tree. This value determines how the tree should try to preserve the tree expansion and selection settings as the structure of the tree changes.

`mode` is the new setting and must be one of the constants defined in this class: `PRESERVE_MODE_NONE` to not attempt to preserve the original settings, `PRESERVE_MODE_IDENTITY` to base the preservation on the identities of the nodes involved, or `PRESERVE_MODE_VALUE` to use the string value of the nodes to maintain the settings.

```
public void setSelectionState(ArrayList selectList);
```

Establish the selection(s) within the tree via this method. The array passed in contains more arrays as elements. The sub-arrays are lists of strings that correspond to the different levels within the tree.

`selectList` is the array of string arrays with the new selection.

Instances of the `SearchTree` class can be found throughout JBuilder itself. Look at the Project Pane and Structure Pane for starters. In JBuilder 10 there is also a `SearchList` class.

Streams Class

Use the `com.borland.primetime.util.Streams` class to help out with streams. Its (all static) methods are:

**UNDOCUMENTED**

The `Streams` class is undocumented and so may change in future versions of JBuilder.

```
public static synchronized void copy(InputStream in,
    OutputStream out) throws IOException;
```

Copy the entire contents of an input stream into an output stream.

`in` is the input stream to read.

`out` is the output stream to write to.

```
public static synchronized byte[] read(InputStream in)
    throws IOException;
```

```
public static synchronized byte[] read(InputStream in, long
    length) throws IOException;
```

Load the contents of an input stream into a byte array.

`in` is the input stream to read from.

length is the maximum number of bytes to read. If not specified, the entire stream is loaded.

```
public static char[] readChars(InputStream in, String
encoding) throws UnsupportedOperationException,
IOException;
```

Load the contents of an input stream into a byte array.

in is the input stream to read.

encoding is the encoding scheme to use while reading, or null for the default scheme.

Strings Class

Dealing with string values is the specialty of the `com.borland.primetime.util.Strings` class. All of its methods are static, making them immediately available. They involve conversions between different platforms, encoding and decoding characters, and formatting strings as shown below:

```
public static boolean canConvertToInteger(String value);
```

Returns true if the value can be converted into an integer, or false if it cannot.

value is the text to test.



VERSION

The `canConvertToInteger` method is only available in JBuilder 10.

```
public static String capitalize(String value);
```

Capitalize the first character of the given value, e.g. “java rocks” becomes “Java rocks”.

value is the text to process.



VERSION

The `capitalize` method is only available in JBuilder 10.

```
public static String convertLineEndings(String input,
String lineSeparator);
```

Alter Unix- or Windows-style line endings to a given value.

input is the text to convert.

lineSeparator is the text to replace the line endings with.



VERSION

The `convertLineEndings` method is not available in JBuilder 7.

```
public static int convertToInteger(String value) throws
NumberFormatException;
```

Converts the given string into an integer value, or throws an exception if it cannot.

value is the text to convert.



VERSION

The `convertToInteger` method is only available in JBuilder 10.

```
public static String convertToPlatformLineEndings(String
input);
```

Convert line-ending characters within a string to conform to the setting for the current platform.

input is the text to convert.

```
public static String convertToUnixLineEndings(String
input);
```

Change line endings to the Unix-style “\n” setting.

input is the text to convert.



VERSION

The `convertToUnixLineEndings` method is not available in JBuilder 7.

```
public static String decapitalize(String value);
```

Decapitalize the first character of the given value, e.g. “JAVA ROCKS” becomes “jAVA ROCKS”.

value is the text to process.



VERSION

The `decapitalize` method is only available in JBuilder 10.

```
public static String decode(String encoded);
```

Restore a string to its original form by undoing an encode call.

encoded is the string to decode.

```
public static String[] decodeArray(String encoded);
```

Perform the opposite of the `encodeArray` method, transforming a delimited list of values into an array, decoding them as they are extracted. It returns null if the string passed in is null.

encoded is the semi-colon (;) delimited list of values to break apart and decode.

```
public static String encode(String original);
```

Convert a string to escape any characters that could cause problems, based on the standard encoding (see `STANDARD_ENCODING` field below). It returns null if passed a null string.

original is the string to encode.

```
public static String encodeArray(String[] original);
```

Encode a list of strings and concatenate them to produce a semi-colon (;) delimited string value. The standard encoding is used, along with “\.” for any semi-colons in the original strings. A null is returned if the array is passed in as null.

original is the array of strings to encode. None of these may be null.

```
public static String format(String template, Object[]
params);
```

```
public static String format(String template, Object
param0);
```

```
public static String format(String template, Object param0,
Object param1);
```

```
public static String format(String template, Object param0,
Object param1, Object param2);
```

```
public static String format(String template, Object param0,
    Object param1, Object param2, Object param3);
```

These methods are wrappers for the standard `MessageFormat` class. They let you substitute values into a string at locations identified by the text “{n}” where n is a number from 0 up.

For example, calling

```
Strings.format("Error in {0} at {1},{2}", fileName,
    new Integer(line), new Integer(column));
```

could produce the following

```
Error in Xyz.java at 10,23
```

`template` is the message to generate with embedded positional markers.

`params` is an array of objects to insert into the message.

`param0` through `param3` are individual objects to insert into the message, substituting for “{0}” through “{3}”.

```
public static String[] getSubStrings(String value, int
    maxWidth, FontMetrics fm);
```

Break text up into lines designed to fit in a given width. Breaks occur between words as well as at normal line breaks.

`value` is the text to break up.

`maxWidth` is the maximum width in pixels of a line.

`fm` defines the font characteristics for the calculation.

VERSION

The `getSubStrings` method is only available in JBuilder 10.

```
public static boolean isEmpty(String value);
```

Discover whether or not a string is null or of zero length, returning true if so, or false if not.

`value` is the text to examine.

VERSION

The `isEmpty` method is only available in JBuilder 10.

```
public static String ltrim(String value);
```

```
public static String rtrim(String value);
```

Remove whitespace from the left- or right-hand side of a string.

`value` is the text to trim.

```
public static boolean startsWithIgnoreCase(String value,
    String possiblePrefix);
```

Returns true if a string begins with a given prefix, ignoring case differences, or false if it does not.

`value` is the text to examine.

`possiblePrefix` is the text to look for.

VERSION

The `startsWithIgnoreCase` method is only available in JBuilder 10.

The class also includes three constants:

```
public static final String EMPTY_ARRAY[];
```

An empty array of strings for your use.

```
public static final StringEncoding STANDARD_ENCODING;
```

A string encoding object that uses the standard encoding below.

```
public static final String STANDARD_ENCODING_DESCRIPTION;
```

The text value (“\r\n\b\t\f”) that feeds into the standard encoding object to escape the usual Java whitespace characters.

Strings.StringEncoding Class

Escaping and un-escaping characters within strings is the task appointed to the `com.borland.primetime.util.Strings.StringEncoding` class. It accepts a list of encodings when created and then applies these to strings it is presented with. Each encoding consists of a string of paired characters. The first in each pair is the character that needs to be escaped. This is done by replacing it with the escape character followed by the second character in the pair. The escape character itself is doubled whenever it is encountered during encoding.

For example, given an encoding string of “+p-m*t/d” and an escape character of “\”, the string “1 + 2 * 3 - 4” would become “1 \p 2 \t 3 \m 4”.

```
Strings.StringEncoding encoding =
    new Strings.StringEncoding("+p-m*t/d");
System.out.println(encoding.encode("1 + 2 * 3 - 4"));
```

The class’ methods are listed below:

```
public Strings.StringEncoding(String encoding);
public Strings.StringEncoding(String encoding, char
    escapeChar);
public Strings.StringEncoding(String encoding, char
    escapeChar, boolean useUnicodeEscape);
```

Create a new encoding for the pairs of characters provided.

`encoding` is the string of pairs of characters to encode.

`escapeChar` is the escape character to use – defaulting to “\”.

`useUnicodeEscape` is true (the default) to use Unicode escape sequences for non-printing and non-ASCII characters, or false otherwise.

```
public String decode(String encoded);
```

Restore a string to its original form by undoing an encode call.

`encoded` is the string to decode.

```
public String encode(String original);
```

Convert a string to escape any characters that could cause problems, based on this encoding. It returns null if passed a null string.

`original` is the string to encode.

The `Strings.StringEncoding` class is demonstrated in the `UISampler` class described at the end of this chapter.

TableSorter Class

Keeping your table data sorted by column contents is simple with the `com.borland.primetime.ui.table.TableSorter` class. It operates as a wrapper around an existing `TableModel` passing many calls on to that model through the abilities inherited from `TableMap` (also in the same package). The data in the original table model is not reordered. Instead an ordered mapping of

the rows is held in this class, with requests directed at the model being translated as necessary.



UNDOCUMENTED

This class remains undocumented.

The methods of this class are as follows:

```
public TableSorter();
public TableSorter(TableModel model);
```

Construct a new table sorter and encapsulate the data.
`model` is the original table model to wrap in this sorter. If not set here you must call `setModel` to establish the link before using this class.

```
public void addMouseListenerToHeaderInTable(JTable table);
```

Call this method to inform this object of events in the table header so that it can re-sort when the user clicks a column header.
`table` is the table displaying the data.

```
public void checkModel();
```

If the number of rows in the model differs from that in the mapping, the latter is recalculated.

```
public int compare(int rowIndex1, int rowIndex2);
```

Compare two rows and return `-1` if the first sorts before the second, `0` if they are equal, or `+1` if the first sorts after the second. Comparisons between the rows are based on the columns nominated for sorting.
`rowIndex1` and `rowIndex2` are the indexes of the two rows to compare.

```
public int compareRowsByColumn(int rowIndex1, int
rowIndex2, int columnIndex);
```

Compare the same column in two rows and return values as above.
`rowIndex1` and `rowIndex2` are the indexes of the two rows to compare.
`columnIndex` is the index of the column in both rows to compare.

```
public int getSortedRowIndex(int rowIndex);
```

Given a row index in the original model, find its new index in the sorted order.
`rowIndex` is the index of the row in the underlying model.

```
public int getUnsortedRowIndex(int rowIndex);
```

Given a row index after sorting, obtain its index in the original model.
`rowIndex` is the index of the sorted row.

```
public Object getValueAt(int rowIndex, int columnIndex);
```

Return the value held in the specified cell, after translating through the sorting map.
`rowIndex` and `columnIndex` denote the position of the cell.

```
public int lastColumnSorted();
```

Retrieve the index of the column used for the last sort of the data.

```
public boolean lastSortAscending();
```

Determine whether or not the last sort was in ascending order, returning `true` if it was, or `false` if it was not.



VERSION

The `lastSortAscending` method is only available in JBuilder 10.

```

public void n2sort();
    Perform an  $O(N^2)$  sort on the contents of the table.
public void reallocateIndexes();
    Ensure that the sorting map matches with the rows in the underlying model.
public boolean redoLastColumnSort();
    Sort again by the last column selected. It returns true if there was a previous
    column sort, or false if there was not.
public void setModel(TableModel model);
    Link to the table model that contains the actual data for the table.
    model is the table model to wrap and sort.
public void setSelectTopAfterSort(boolean selectTop);
    Change how the table selection moves after sorting.
    selectTop is true to select the new row in the position of the original
    selection, or false (the default) to leave the selection as it is.
public void setValueAt(Object value, int rowIndex, int
    columnIndex);
    Update the value held in the specified cell, after translating through the
    sorting map.
    value is the new value for this cell.
    rowIndex and columnIndex denote the position of the cell.
public void shufflesort(int[] unsorted, int[] sorted, int
    startIndex, int endIndex);
    Perform a shuttle sort on the contents of an array.
    unsorted is the array of integer values to be sorted.
    sorted is the resulting sorted array.
    startIndex and endIndex define the extent of the original array to sort.
public void sort(Object object);
    Sort the model data by the previously selected column.
    object is ignored.
public void sortByColumn(int index);
public void sortByColumn(int index, boolean ascending);
    Sort the data in the model by the specified column.
    index is the index of the column to sort by.
    ascending is true (the default) to sort in ascending order, or false to sort in
    descending order.
public void swap(int index1, int index2);
    Swap the positions of two rows.
    index1 and index2 are the two row indexes to swap.
public void tableChanged(TableModelEvent event);
    When the number of rows in the underlying model change, reset the sorting
    map.
    event holds details about the table changes.

```

This class and its functionality are packaged into the `com.borland.pruntime.ui.table.JSortedTable` class. Just use this class wherever you would use a normal `JTable`, and the sorting functionality comes built-in. You have access to the `TableSorter` it uses through its `getTableSorter` method.

See the `UISampler` class described at the end of this chapter for an example of the `TableSorter`.

Text Class

Manipulate text with the `com.borland.primetime.util.Text` class.



UNDOCUMENTED

The `Text` class has not yet been officially documented.

Its (all) static methods are listed below:

```
public static String getHeadingSpace(boolean useTabs, int
    tabSize, int width);
```

Supply a string that represents spacing for an indentation level.

`useTabs` is true to use tabs where possible, or false to always use spaces.

`tabSize` is the number of spaces per tab stop. This value is ignored if `useTabs` is false.

`width` is the size (in characters) of the spacing required.



VERSION

The `getHeadingSpace` method is only available in JBuilder 9 and 10.

```
public static int getIndentColumn(String text, int
    tabSize);
```

Provide the column number where non-blank text starts in a string, taking into account any tabs. The size of the string is returned if it consists solely of space and tab characters.

`text` is the string to process.

`tabSize` is the number of spaces per tab stop.



VERSION

The `getIndentColumn` method is only available in JBuilder 9 and 10.

```
public static String[] makeIntoArray(String text);
```

Break up continuous text into an array of strings delimited by linefeeds and carriage returns.

`text` is the string to process.



VERSION

The `makeIntoArray` method is only available in JBuilder 9 and 10.

```
public static String removeAllWhitespaceFrom(String text);
```

Delete all the whitespace characters from the given text.

`text` is the string to process.

```
public static String[] removeTrailingBlankLines(String[]
    text);
```

As the name says, this method deletes any blank lines at the end of the array of strings and returns the result.

`text` is the string to process.



VERSION

The `removeTrailingBlankLines` method is only available in JBuilder 9 and 10.

```
public static char[] replaceTabs(char[] text, int tabSize);
public static String replaceTabs(String text, int tabSize);
```

Replace any tab characters in the given text with the appropriate number of spaces.

`text` is the data to process, presented as either a character array or a string.

`tabSize` is the number of spaces per tab stop.

TextFile Class

Make working with text files easier with the `com.borland.primetime.util.TextFile` class.



UNDOCUMENTED

The `TextFile` class is currently undocumented.

It extends the basic `java.io.File`, adding the methods below:

```
public TextFile(File file);
public TextFile(File file, String encoding);
public TextFile(String fileName);
public TextFile(String fileName, String encoding);
public TextFile(Url url);
public TextFile(Url url, String encoding);
```

Create a new text file reference with one of these constructors.

`file` is a reference to the file that contains the text.

`filename` is the full path and file name of the file.

`url` is the location of the file to open.

`encoding` is the character encoding for the file. It defaults to the system property “`file.encoding`”.

```
public String getContents();
```

Retrieve the contents of the file as a single string.

```
public void setContents(String text);
```

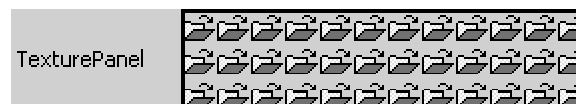
Update the contents of the file through this method.

`text` is the new file contents. These are written straight out to the underlying file.

TexturePanel Class

Fill a panel with repetitions of a background image by using the `com.borland.primetime.ui.TexturePanel` class (see Figure 9-13).

Figure 9–13. A TexturePanel.



UNDOCUMENTED

This class is currently undocumented.

It extends the basic `JPanel`, adding the methods below:

```
public TexturePanel();
public TexturePanel(Image image);
public TexturePanel(LayoutManager layoutManager);
public TexturePanel(Image image, LayoutManager
    layoutManager);
```

Create a new texture panel with one of these constructors.

`image` is the image to replicate across the panel's surface.

`layoutmanager` is the layout manager to use for the panel.

```
public Image getTexture();
```

Retrieve the image copied across the panel's background.

```
public void setTexture(Image image);
```

Establish the image repeated across the panel's surface.

`image` is the picture to use.

See the `UISampler` class described at the end of this chapter for an example of the `TexturePanel`.

VetoException Class

The `com.borland.primetime.util.VetoException` class defines an exception that is thrown in several situations to stop an action in JBuilder from progressing. It adds nothing to the basic `Exception`, serving merely as a separate and identifiable exception type.

`VetoExceptions` are used in wizards to confirm that each page contains valid data. Wizards also use it to determine whether they can finish their processing or need additional information to complete. For example, Listing 9-1 shows the validation method for a wizard page.

Listing 9-1. Page validation in a wizard.

```
public void checkPage() throws VetoException {
    // File name must be specified
    if (getClassName().length() == 0) {
        classNameField.requestFocus();
        JOptionPane.showMessageDialog(wizardHost.getDialogParent(),
            "A class name must be entered",
            TITLE, JOptionPane.ERROR_MESSAGE, null);
        throw new VetoException();
    }
}
```

Its constructors are:

```
public VetoException();
```

```
public VetoException(String message);
```

Create a new exception to veto an activity.

`message` is the description of the problem.

ZipIndex Class

Examining Zip files is made easier with the `com.borland.primetime.util.ZipIndex` class. Once used to open a Zip file, it provides access to all the entries within it, either as a single list, or by stepping through the directory hierarchy. For example, the code in Listing 9-2 examines an `OpenTools` JAR file and lists out its contents. In addition, when it finds the manifest file (`MANIFEST.MF`), it

prints out its contents. A partial listing of the resulting output is shown in Listing 9–3.

Listing 9–2. Display the contents of a Zip file.

```
String fileName = "c:/JBuilder/JBShared/ext/GIFEditor.jar";
ZipIndex zip = ZipIndex.getZipIndex(new File(fileName));
System.out.println("Zip file: " + fileName);
ZipIndexEntry[] entries = zip.getZipIndexEntries();
for (int index = 0; index < entries.length; index++) {
    System.out.println("  " + index + " " +
        entries[index].getDirectory() +
        (entries[index].getDirectory().length() > 0 ? File.separator : "") +
        entries[index].getName() + " " +
        new Date(entries[index].getLastModified()));
    if (entries[index].getName().equals("MANIFEST.MF")) {
        try {
            System.out.println("Manifest:\n" +
                new String(zip.read(entries[index])));
        }
        catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

Listing 9–3. Zip contents displayed.

```
Zip file: c:/JBuilder/JBShared/ext/GIFEditor.jar
0 Acme Fri Nov 30 00:00:00 GMT+10:00 1979
1 META-INF Fri Nov 30 00:00:00 GMT+10:00 1979
2 wood Fri Nov 30 00:00:00 GMT+10:00 1979
3 Acme\IntHashtable.class Sat Jan 31 22:40:52 GMT+10:00 2004
4 Acme\IntHashtableEntry.class Sat Jan 31 22:40:52 GMT+10:00 2004
5 Acme\IntHashtableEnumerator.class Sat Jan 31 22:40:52 GMT+10:00 2004
6 Acme\JPM Fri Nov 30 00:00:00 GMT+10:00 1979
7 Acme\JPM\Encoders Fri Nov 30 00:00:00 GMT+10:00 1979
8 Acme\JPM\Encoders\GifEncoder.class Sat Jan 31 22:40:52 GMT+10:00 2004
9 Acme\JPM\Encoders\GifEncoderHashitem.class Sat Jan 31 22:40:52
GMT+10:00 2004
10 Acme\JPM\Encoders\ImageEncoder.class Sat Jan 31 22:40:52 GMT+10:00
2004
11 META-INF\MANIFEST.MF Sat Jan 31 22:40:52 GMT+10:00 2004
Manifest:
Manifest-Version: 1.0
OpenTools-UI: wood.keith.opentools.gifeditor.GIFEditorViewerFactory
12 wood\keith Fri Nov 30 00:00:00 GMT+10:00 1979
13 wood\keith\opentools Fri Nov 30 00:00:00 GMT+10:00 1979
14 wood\keith\opentools\gifeditor Fri Nov 30 00:00:00 GMT+10:00 1979
15 wood\keith\opentools\gifeditor\ActionConstants.class Sat Jan 31
22:40:52 GMT+10:00 2004
16 wood\keith\opentools\gifeditor\Color.gif Sat Jan 31 22:40:52
GMT+10:00 2004
17 wood\keith\opentools\gifeditor\ColorAction$1.class Sat Jan 31
22:40:52 GMT+10:00 2004
:
```



UNDOCUMENTED

The ZipIndex class is not documented by Borland.

The methods of this class are described below. Note that all filenames are case-sensitive.

```
public synchronized void close();
```

Close the associated Zip file.

```
public boolean contains(String entryName);
```

Return true if the Zip file contains this entry, or false if it does not. If the file is hidden (see the `hide` method) then it returns false.

`entryName` is the full name of the entry to look for.

```
public synchronized ZipIndexEntry[] getAllChildren(String
    rootName);
```

Extract all the children beneath a particular entry in the Zip file.

`rootName` is the starting point to find children from. Use an empty string for all children at any level. Do not pass in `null`.

```
public ZipIndexEntry[] getChildren(String entryName);
```

Retrieve a list of the children of a particular entry in the Zip file. If it has no children, an empty array is returned. If the file is hidden (see the `hide` method) then it also returns an empty array.

`entryName` is the full name of the entry to examine.

```
public static synchronized ZipIndex getExistingZipIndex(
    File file);
```

Locate and return a reference to the entry for a particular file, or `null` if it cannot be found.

`file` is the file to find.

**VERSION**

The `getExistingZipIndex` method is not available in JBuilder 7.

```
public String[] get_filenames(String entryName);
```

Get an array of path entries for a given entry in the Zip file. An empty array is returned if the entry is not found. If the file is hidden (see the `hide` method) then it returns an empty array.

`entryName` is the name of the entry to look at.

```
public long getLastModified(String entryName) throws
    IOException;
```

```
public long getLastModified(ZipIndexEntry entry);
```

Retrieve the timestamp of the specified Zip file entry.

`entryName` is the full name of the entry to get the timestamp for.

`entry` is the Zip entry to get the timestamp for.

```
public static synchronized ZipIndex[] getOpenZipIndexes();
```

Obtain a list of all the Zip files that are currently open.

```
public long getRawLastModified(String entryName) throws
    IOException;
```

```
public long getRawLastModified(ZipIndexEntry entry);
```

Retrieve the raw timestamp of the specified Zip file entry. This value has the date and time encoded in sets of bits within it: bits 0 to 5 are the number of seconds divided by two, bits 6 to 11 are the minutes, bits 12 to 16 are the hours, bits 17 to 21 are the day, bits 22 to 25 are the month, and bits 26 to 31 are the year after subtracting 1980.

`entryName` is the full name of the entry to get the timestamp for.

`entry` is the Zip entry to get the timestamp for.

```
public static synchronized ZipIndex getZipIndex(File file);
```

```
public static synchronized ZipIndex getZipIndex(Url url);
```

Create a new `ZipIndex` object for the given Zip file.

file or url is the reference to the Zipped file to be processed.



VERSION

The `getZipIndex` method that takes a `Url` argument is only available in JBuilder 10.

```
public ZipIndexEntry[] getZipIndexEntries();
```

Return a list of all the entries in the Zip file. If the file is hidden (see the `hide` method) then it returns an empty array.

```
public ZipIndexEntry getZipIndexEntry(String entryName);
```

Locate a given entry and return a reference to it, or `null` if it cannot be found. If the file is hidden (see the `hide` method) then it also returns `null`.

`entryName` is the name of the entry to find.

```
public synchronized void hide();
```

Make the file temporarily unavailable. Use the `show` method to return it to full function.

```
public static synchronized void hideAll();
```

Call `hide` for all the entries returned by `getOpenZipIndexes`.



VERSION

The `hideAll` method is not available in JBuilder 7.

```
public boolean isDirectory(String entryName);
```

Return `true` if the given path is a directory, or `false` otherwise. If the file is hidden (see the `hide` method) then it returns `false`.

`entryName` is the name of the entry to check. Do not include any trailing file path separator character.

```
public boolean isHidden();
```

Determine whether or not the file is currently hidden (see the `hide` method).

```
public boolean isOpen();
```

Determine whether the Zip file has been opened. Return `true` if it has, or `false` if it has not.

```
public int length(String entryName) throws IOException;
```

Find the length (in bytes) of a given entry. It throws a `FileNotFoundException` if the given entry is unknown. You must call `open` before using this method, or you will get `NullPointerExceptions` instead.

`entryName` is the name of the entry to find.



VERSION

The `length` method is only available in JBuilder 10.

```
public synchronized void open();
```

Open the associated Zip file and load its index.

```
public byte[] read(String entryName) throws IOException;
```

```
public synchronized byte[] read(ZipIndexEntry entry) throws
    IOException;
```

Load the contents of the specified entry in the Zip file into a byte array, and return it.

`entryName` is the full name of the entry to read from the Zip file.

`entry` is the Zip entry to read.

```
public int read(String entryName, byte[] bytes) throws
    IOException;
public synchronized int read(ZipIndexEntry entry, byte[]
    bytes) throws IOException;
    Load the contents of the specified entry in the Zip file into the byte array, and
    return the number of bytes read.
    entryName is the full name of the entry to read from the Zip file.
    entry is the Zip entry to read.
    bytes is the byte array to fill with the entry's contents.
public synchronized void show();
    Make the file available for full use again following a call to the hide
    method.
public static synchronized void showAll();
    Call show for all the entries returned by getOpenZipIndexes.
```

**VERSION**

The `showAll` method is not available in JBuilder 7.

One of the tabs in the `UISampler` class described later illustrates how the `ZipIndex` and `ZipIndexEntry` classes can be used to peruse a Zip file.

ZipIndexEntry Class

Individual entries within a Zip file are encapsulated by instances of the `com.borland.prime.time.util.ZipIndexEntry` class. You retrieve these entry objects from the `ZipIndex` object that is tied to their Zip file. Their abilities are shown below:

**UNDOCUMENTED**

This class has not been documented yet.

```
public ZipIndexEntry(String fullPathName);
public ZipIndexEntry(String pathName, String fileName);
    Create a new Zip file entry.
    fullPathName is the full path and file name within the Zip file for this
    entry.
    pathName is the path name within the Zip file.
    fileName is the file name for this entry.
public String getDirectory();
    Return the name of the directory that contains this entry.
public long getLastModified();
    Retrieve the timestamp for this entry indicating when it was last altered.
public String getName();
    Obtain the name of this entry. This is just the file name without any path
    information.
public int getRawLastModified();
    Retrieve the raw timestamp of this entry. The value has the date and time
    encoded in sets of bits within it: bits 0 to 5 are the number of seconds divided
    by two, bits 6 to 11 are the minutes, bits 12 to 16 are the hours, bits 17 to 21
```

are the day, bits 22 to 25 are the month, and bits 26 to 31 are the year after subtracting 1980.

```
public boolean isDirectory();
```

Return true if this entry is a directory, or false otherwise.

UISampler Example

The `UISampler` class included as an example on the accompanying Web site demonstrates several of these classes in action. It is installed as an `OpenTool` and is accessed via the UI Sampler entry on the Tools menu.

The sampler's first tab, Info, (see Figure 9–14) shows off the `Platform` and `JBuilderInfo` classes. Details from these are displayed on the form.

Figure 9–14. Demonstrating *Platform* and *JBuilderInfo*.

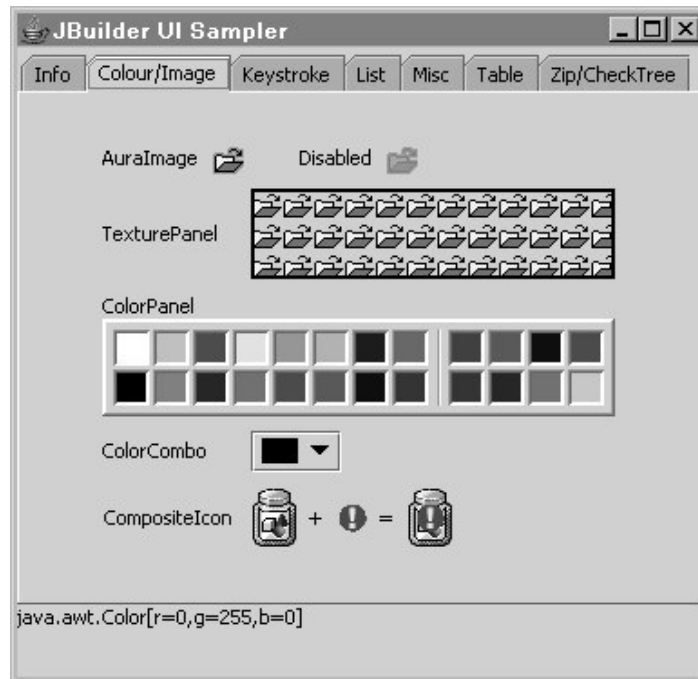
The screenshot shows a Java Swing window titled "JBuilder UI Sampler". It has a tabbed interface with the following tabs: Info, Colour/Image, Keystroke, List, Misc, Table, and Zip/CheckTree. The "Info" tab is currently selected. The form contains the following fields and controls:

- OS Name:** A text field containing "Windows 2000".
- Platform Selection:** A group of four checkboxes:
 - ☒ Windows
 - ☐ Solaris
 - ☐ Unix
 - ☐ Linux
- User:** A text field containing "Keith Wood".
- Company:** A text field containing "N/A".
- Description:** A text field containing "JBuilder X Foundation".
- Build Number:** A text field containing "10.0.176.0".
- SKU:** A text field containing "Foundation".
- Licensing:** A group of four checkboxes:
 - ☒ Licensed
 - ☐ Professional
 - ☐ Trial
 - ☐ Enterprise

At the bottom of the dialog, there is a `MessageLabel` displaying the text: `java.awt.Color[r=0,g=255,b=0]`.

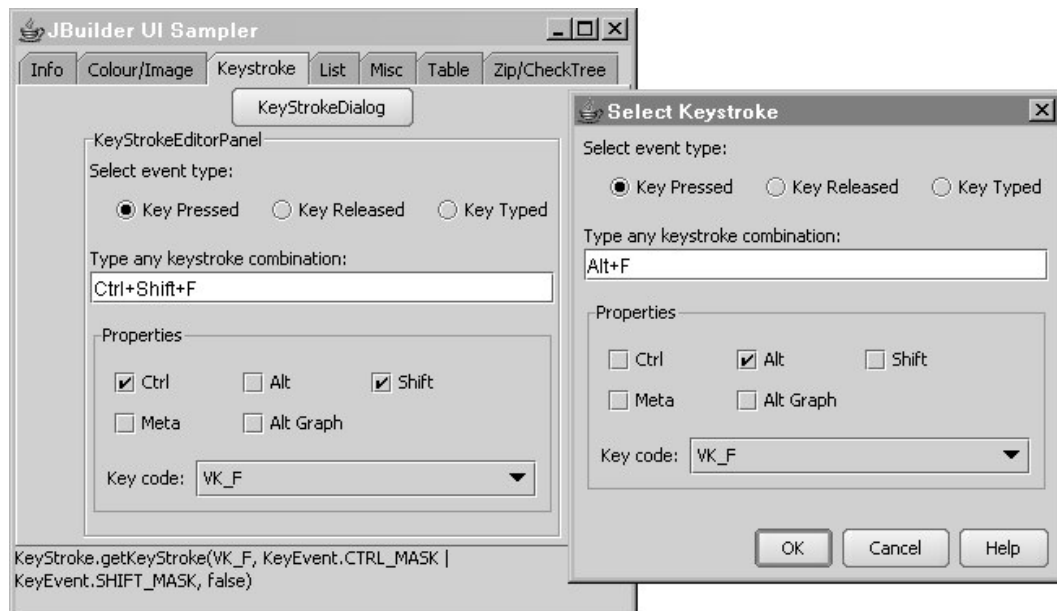
The second tab (see Figure 9–15) shows off the `AuraImage`, `TexturePanel`, `ColorPanel`, `ColorCombo`, and `CompositeIcon` classes. Behind the scenes, the `Icons` and `Images` classes are also used. Selecting a new color updates the aura image. The color details are also displayed in the `MessageLabel` at the bottom of the form.

Figure 9–15. Demonstrating *AuraImage*, *TexturePanel*, *ColorPanel*, *ColorCombo*, and *CompositeIcon*.



The next tab (see Figure 9–16) illustrates the *KeyStrokeDialog*, *KeyStrokeEditorTextField*, and *KeyStrokeEditorPanel* classes. Selecting a keystroke in the popup dialog displays its description in the message label at the bottom of the form, while selecting a keystroke on this page displays its Java initialization string.

Figure 9–16. Demonstrating *KeyStrokeDialog*, *KeyStrokeEditorTextField*, and *KeyStrokeEditorPanel*.



The List tab (see Figure 9–17) illustrates the `ListPanel` class through a customized descendent (`ListPanelSample`). Entries are simple text values.

Figure 9–17. Demonstrating `ListPanel`.

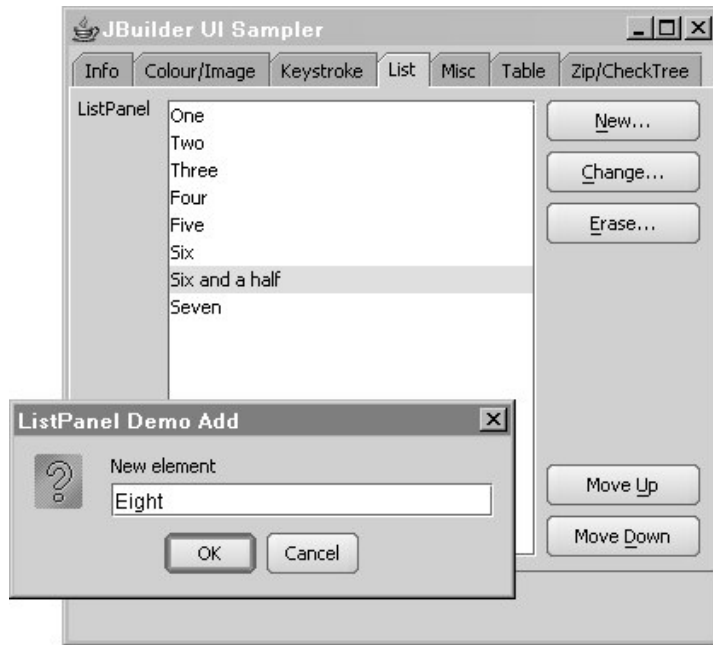
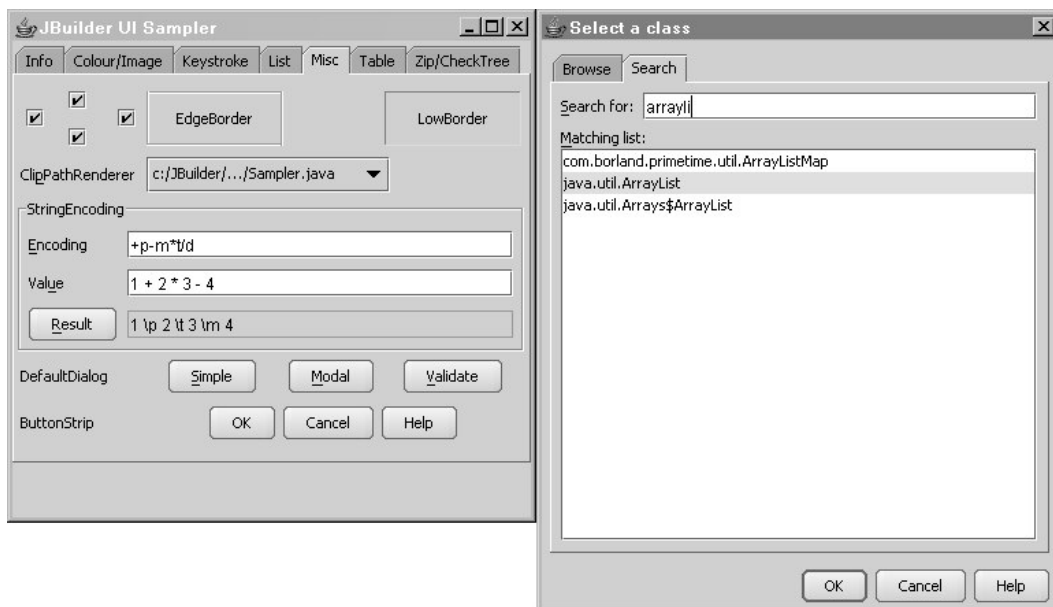


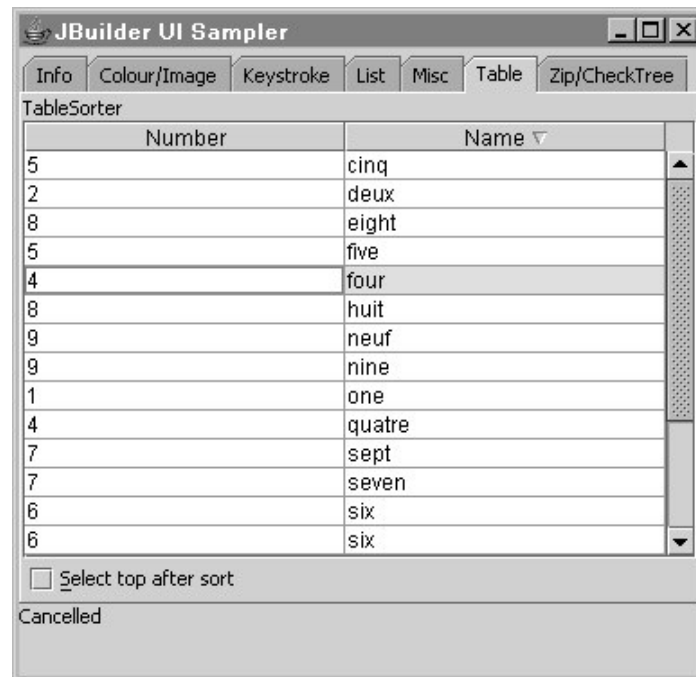
Figure 9-18, the Misc tab, demonstrates several miscellaneous classes, including `EdgeBorder` and `LowBorder`, `ClipPathRenderer`, `Strings.StringEncoding`, `DefaultDialog`, and `ButtonStrip`. Resize the form horizontally to see the effects of the clip renderer in the combobox. The `DefaultDialog` buttons popup appropriate dialog boxes. For the `Validate` button, you are required to validate the contents of the dialog before it can be closed. The `OK` button on the button strip calls up class browser from the `PackageBrowserTree` class, with your selection being displayed in the message label.

Figure 9–18. Demonstrating other miscellaneous classes.



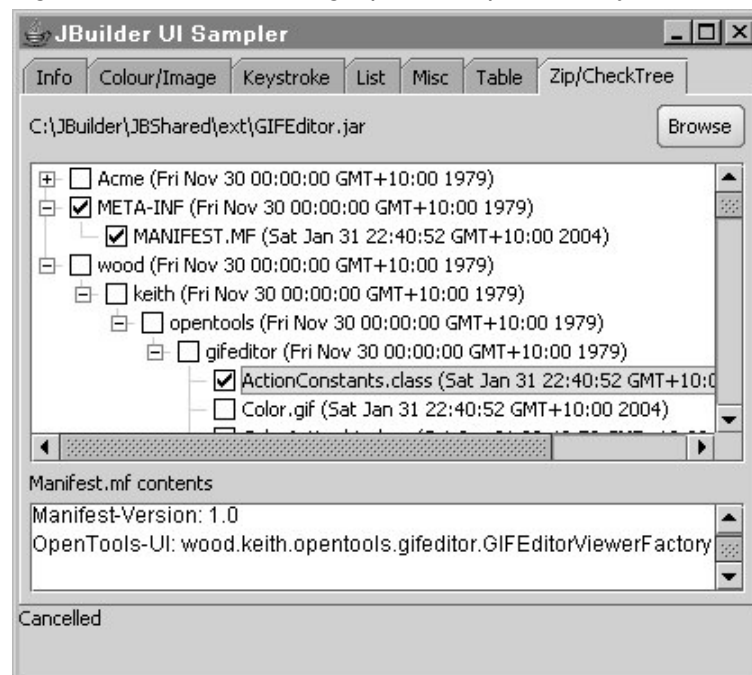
The Table tab (see Figure 9–19) shows the use of the `TableSorter` class to back up a `JTable`.

Figure 9–19. Demonstrating `TableSorter`.



And finally, Figure 9–20 demonstrates the `ZipIndex` and `ZipIndexEntry` class. When you browse for and select a Zipped file, its directory entries are read and loaded into a `CheckTree`. If a manifest file is found in the Zip file, its contents are copied into the text area at the bottom of the form.

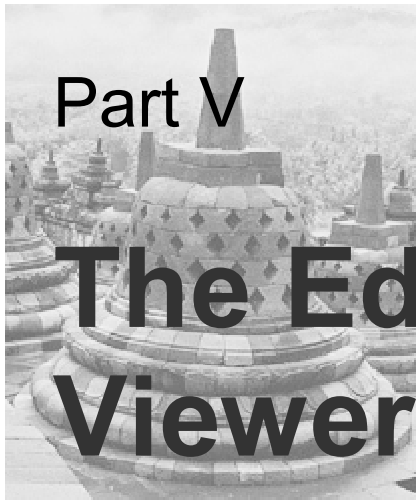
Figure 9–20. Demonstrating `ZipIndex`, `ZipIndexEntry`, and `CheckTree`.



Summary

The classes presented in this chapter cover a wide range of functionality. Included are utility classes that provide information about the JBuilder environment, and let you work with classpaths, icons, images, streams, strings, and Zip files. User interface components include images and icons, keystroke controls, list cell renderers, an enhanced tree, and a textured panel.

The `UISampler` class that accompanies this chapter demonstrates many of these classes, showing how they can be used.



Part V

The Editors and Viewers

The Content Pane is the main focus of the JBuilder IDE. It is here that you see the contents of the files that make up your projects. Each file may have a number of different views of it, corresponding to the tabs along the bottom of the pane. Viewers may pertain to just one node type, or be applicable across many assorted types.

Normal text-based files have a viewer associated with them that allows you to update their contents. This is the editor, which can provide syntax highlighting for various styles of code, and responds to numerous keystrokes to invoke actions on its contents.

Chapter 15 looks at the editor itself and focuses on its configuration settings, and how it highlights certain lines for you, either through syntax highlighting or via custom styles and marks. Chapter 16 examines the role of editor kits and scanners, tools that extract the syntax of a text-based file for highlighting. Linking keystrokes to actions is the subject of Chapter 17, and the management of the keymaps that provide this mapping.

Providing alternate views of a node is discussed in Chapter 18, allowing you to extend your interactions with a particular node type. Related to this, Chapter 19 describes the Structure Pane and how you can use this to further enhance your node viewer.

The Component Modeling Tool (CMT) is covered in Chapter 20. This tool lets you interact with the components encoded in a Java class, retrieving or setting their properties, and examining their relationships. CMT abilities are put to use in Chapter 21 which looks at designers that extend the Structure Pane for the Design tab, while in Chapter 22 you use them to apply graphical changes to a layout back into the code.



Chapter 20

Component Modeling Tool

Working with the designers in JBuilder requires interacting with the UI components that make up a class. The Component Modeling Tool (CMT) is a collection of interfaces that allows you to delve into the components defined within a class. You can read property settings from them, and alter these values in response to user activities.

CMT in turn relies on the Java Object Toolkit (JOT) to parse the Java code and extract subcomponents from it (see Chapter 24). JOT also allows CMT to discover what properties and events those components expose, following the standard JavaBeans patterns. In fact, if the component has an associated `BeanInfo` class, that is also scanned for further definitions. Settings for the properties and event handlers made in the UI initialization method (usually `jbInit`) for the parent component are retrieved and stored against each subcomponent.

To fully expose the attributes of a class, its ancestral chain must be traversed. At each stage a `CmtComponent` object is created to model the superclass. Performing all this searching and instantiation is an expensive operation during the design process, so JBuilder caches information about each class after it has first been loaded. JBuilder retains the summarized results between sessions by storing them in special files in the user's home directory. Within the appropriate folder for your version of JBuilder (such as `.jbuilder9`) appears the `pme` (properties, methods, or events) directory that contains files with `.pme` extensions. Each one is named for the class that it describes, like `javax.swing.JButton.pme`. Then JBuilder can just read this file rather than walking through the class hierarchy.

NOTE

The `.pme` files are simple text files, similar to Windows initialization files. They contain three sections with headers marked by square brackets (`[]`), which then contain sets of values, one per line. The `BeanDescriptor` section contains overall information about the component, including its class name and any icons used to represent it. Next comes the `PropertyDescriptor` section that lists the component's properties, with their display names, tooltip, type, and access method names. And finally there is the `EventSetDescriptors` section, which holds the set of listener interfaces implemented on the component, again with their generic name, the interface name, and their add and remove methods.

Within a `CmtComponent` object, the attributes of that class are represented by `CmtEvent`, `CmtMethod`, and `CmtProperty` objects. If you have access to the source for the component, then each of these also implements a source form – `CmtComponentSource`, `CmtEventSource`, `CmtMethodSource`, or `CmtPropertySource` – to allow you to update its details. Together these objects define the available attributes for any component of this type.

When a component of a certain type is created within a class, it is embodied in a `CmtSubcomponent` instance. You can then interact with its attribute values through the `CmtEventState`, `CmtMethodCall`, and `CmtPropertyState` interfaces. Again, if you are able to update the code containing these attributes, you do so through another interface: `CmtPropertySetting`.

NOTE

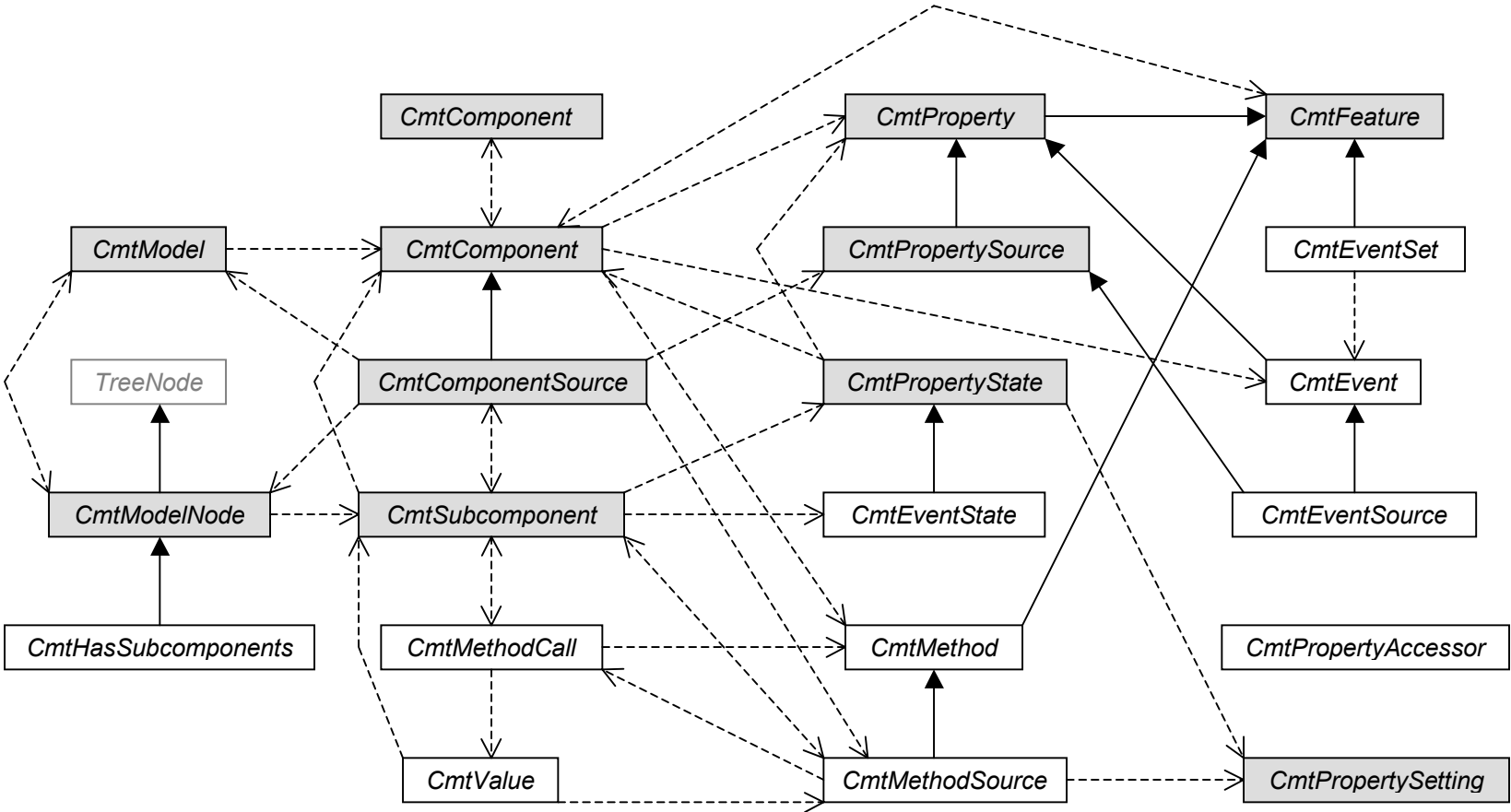
The CMT interfaces and classes belong to the `com.borland.jbuilder` branch of the package structure since they are directly related to Java code. This follows the split between the PrimeTime packages (the generic IDE framework) and JBuilder (the Java-specific IDE).

The Inspector in the designer uses all this information to provide the lists of properties and events for each component, updating its display whenever a different component is selected. Property editors allow each property or event to be presented and updated in an appropriate manner. Default editors apply to the basic property types. You can also provide custom editors for specific properties – linking the two through a `BeanInfo` class.

Within the designer (Design tab for Java source files), nodes (`CmtModelNode`) in a subtree (`CmtModel`) in the Structure Pane represent the subcomponents in a class. These provide access to the underlying subcomponent and its attributes.

Figure 20–1 shows how the interfaces in this package relate to each other. Solid lines and arrows between interfaces indicate inheritance. Dashed lines and open arrows denote navigational routes, for example, from a `CmtSubcomponent` object you can get to a `CmtPropertyState` and thence to the `CmtProperty` object. Interfaces in shaded boxes are discussed in detail in this chapter. The `TreeNode` interface comes from the standard `javax.swing.tree` package.

Figure 20–1. The CMT interfaces.



CmtComponents Interface

The `com.borland.jbuilder.cmt.CmtComponents` interface gives you access to the components that make up a file in a project. You receive a reference to this interface when working with the `Designer` interface through its `annotate` method (see Chapter 21), or directly from the `DesignerViewer` class (the one behind the **Design** tab) as follows:

```
DesignerViewer viewer = (DesignerViewer)browser.getViewerOfType(
    browser.getActiveNode(), DesignerViewer.class);
if (viewer != null) {
    CmtComponents comps = viewer.getComponents();
}
```

Its methods are shown below:

```
public CmtComponent getComponent(JotClass clazz);
public CmtComponent getComponent(JotClass clazz, boolean
    logErrors);
public CmtComponent getComponent(JotFile file);
public CmtComponent getComponent(JotFile file, boolean
    logErrors);
public CmtComponent getComponent(Url url);
```

Gain access to the main component within a file or class through these methods. They return `null` if no appropriate component is found.

`clazz` is the `JotClass` representing the component.

`file` is the `JotFile` that contains the component.

`url` is the location of the file to open.

`logErrors` is `true` (the default) to report any errors to a tab in the Message Pane (labeled **Designer**), or `false` to suppress them.

```
public JotPackages getPackages();
```

Retrieve the JOT package manager for this set of components. See Chapter 24 for more details.

```
public JBProject getProject();
```

Obtain a reference to the project for this set of components.

```
public void release(CmtComponent component);
```

Free up any resources held by a component and its associated information.

`component` is the component that is no longer needed.

```
public void shutdown();
```

Finish with this component manager.

The `com.borland.jbuilder.cmt.CmtComponentManager` class implements `CmtComponents`.



UNDOCUMENTED

The `CmtComponentManager` class has not yet been documented.

This class adds the following methods:

```
public CmtComponent createComponent(CmtComponents
    cmtComponents, JotClass jotClass);
public CmtComponent createComponent(CmtComponents
    cmtComponents, String pmeDir, int pmeReadFlag, int
    pmeWriteFlag, JotFile jotFile, boolean logErrors);
```

Generate a reference to a new component and return it for further use.

`cmtComponents` is the manager for the new component.

`jotClass` is the JOT reference to the class of the component to create.

`pmeDir` is the full path to the location of the `.pme` files.

`pmeReadFlag` is the option set for when to read `.pme` files.

`pmeWriteFlag` is the option set for when to write `.pme` files.

`jotFile` is the JOT reference to the containing file for the new component.

`logErrors` is true (the default) to report any errors to a tab in the Message Pane (labeled Designer), or false to suppress them.

**VERSION**

The `createComponent` methods are only available in JBuilder 9 and 10.

```
public static PropertyEditor findEditor(Class clazz);
```

Find a property editor for a particular type of property. A generic editor is returned if no specific one is available.

`clazz` is the class of the property type for which to locate an editor.

```
public static void registerComponentFactory(
    CmtComponentFactory factory);
```

Register a factory to use to create new component references.

`factory` is the component generator.

**VERSION**

The `registerComponentFactory` method is only available in JBuilder 9 and 10.

CmtComponent Interface

Information about each component within a Java file is held in an object that implements the `com.borland.jbuilder.cmt.CmtComponent` interface. It gives you access to the properties and events of that component.

Its methods are listed below:

```
public String getContainerDelegate();
```

For Swing containers that have an intermediary between the container class and the object that handles the component's layout and sub-components, you use this method to retrieve that reference. For example, `JFrame` lets its content pane do the UI work, so this method returns "`getContentPane()`" for it.

```
public Class getCustomizerClass();
```

If the `BeanInfo` class for this component defined a customizer class, then return it. Otherwise, return `null`. The returned class provides its own user interface to allow you to update the component's properties in one go.

```
public int getDefaultEventIndex();
public int getDefaultPropertyIndex();
```

Find the index into the `getEvents` or `getProperties` arrays for the default event or property for this component, or `-1` if there is no default.

```
public CmtEvent getEvent(String name);
public CmtEvent[] getEvents();
```

Return one or all the events for this component.

`name` is the name of the event to locate, such as `"keyPressed"`. If no matching event is found, a `null` is returned.

```
public RuntimeException getException();
```

Receive back an exception object if this component is in an invalid state, or `null` if everything is valid.

```
public JotFile getFile();
```

Access the JOT file that contains this component.

```
public Icon getIcon();
```

If a `BeanInfo` class is associated with this component, and it defines a small icon to represent it, then return that icon here. Otherwise, return `null`.

```
public Class getLiveClazz();
```

Retrieve a reference to the actual class for this component.

```
public JotClass getLiveType();
```

Obtain a reference to the current class information for this component. If a proxy class is being used in the designer because the real class is abstract, that proxy is returned here.

```
public CmtComponents getManager();
```

Locate the CMT manager for this component through this method.

```
public CmtMethod getMethod(String name);
```

```
public CmtMethod getMethod(String name, JotClass[] types);
```

Find the first method with a particular name and return a reference to it, or `null` if none match.

`name` is the name of the method to locate.

`types` is an array of the types of arguments that the method accepts. Any set of parameters matches if not specified.

```
public CmtMethod[] getMethods();
```

Retrieve a list of all of the exposed methods for this component.

```
public CmtProperty[] getProperties();
```

```
public CmtProperty getProperty(String name);
```

Get all or just one of the properties of this component.

`name` is the name of the single property to retrieve, such as `"background"`.

If such a property is not found, a `null` is returned.

```
public CmtFeature getPropertyFromSetter(String methodName);
```

Find a property or event's details given the name of its associated method. A `CmtProperty` or `CmtEventSet` object is returned, or `null` if the name cannot be matched.

`methodName` is the name of the setter or adder method for the property required, such as `"setBackground"` or `"addMouseListener"`.

```
public JotClass getType();
```

Retrieve JOT information about the class for this component. This appears to always be identical to `getLiveType`.

```
public boolean isBean();
```

Determine whether this component is a `JavaBean`, returning `true` if it is, or `false` if it is not.

```
public boolean isContainer();
```

Returns `true` if this component is a container (it descends from `java.awt.Container`), or `false` if it is not. This determines whether or not it may have sub-components.

```
public boolean isHiddenState();
```

Discover whether this component has been marked as “hidden-state” in the associated `BeanInfo` class.

```
public boolean isReadOnly();
```

If this component resides in a read-only file, this method returns `true`. Otherwise, it returns `false`.

```
public void release();
```

When the component is about to fall out of scope, this method is invoked to allow it to release any resources it may be using.

CmtComponentSource Interface

Building on the abilities of `CmtComponent` above, the `com.borland.jbuilder.cmt.CmtComponentSource` interface adds methods to update the information about the component within a file. It is only implemented when the source for the file is available and can be altered. Within the `Designer` interface (see Chapter 21) several methods receive references to this type of object, or you can access it directly from the `DesignerViewer` class (the one behind the Design tab) as follows:

```
DesignerViewer viewer = (DesignerViewer)browser.getViewOfType(
    browser.getActiveNode(), DesignerViewer.class);
if (viewer != null) {
    CmtComponentSource compSource = viewer.getComponentSource();
}
```

Once you have a `CmtComponentSource`, you can then interact with its subcomponents and the designer models that represent them. You can also register for events that occur within this component.

The new methods are listed below:

```
public void addComponentSourceListener(
    CmtComponentListener listener);
```

Include an object in the list of those informed about events within this component.

`listener` is the object to notify of the events.

```
public CmtMethodSource addMethod(String returnType, String
    name);
```

Add an empty method to this component and return a reference to it.

`returnType` is the full name of the return type for the method.

`name` is the name of the new method.

```
public void addModel(CmtModel model, CmtModel aheadOf);
```

Include a new model in the component tree. Usually designers would call this from their `annotate` methods (see Chapter 21).

`model` is the new model to add.

`aheadOf` indicates where the new model should be placed. Use `null` here to add it at the end.

```
public CmtPropertySource addProperty(String type, String
name);
```

Add a property to this class and return a reference to it.

`type` is the full name of the property's type.

`name` is the field name for the new property.

```
public CmtSubcomponent addSubcomponent(String type, String
name, int scope);
```

Include a new variable in this component and return a reference to it.

`type` is the full name of the variable's type.

`name` is the field name for the new variable.

`scope` is one of `CLASS_SCOPE` or `METHOD_SCOPE` from the `CmtSubcomponent` interface. The former places the entry in the class as a whole, while the latter places the entry within the UI initialization method.

```
public boolean checkReread();
```

Returns true if the buffer is out of date, or false if it has not changed.

```
public void commit(boolean hard);
```

Write any changes to the component back into the shared file buffer.

`hard` is true to write this out to disk as well, or false to just update the buffer.

Its value is actually ignored.

```
public void fireComponentChanged();
```

```
public void fireSubcomponentChanged(CmtSubcomponent
subcomponent);
```

Notify registered listeners that something has changed within this component.

`subcomponent` is the subcomponent that changed, if applicable.

```
public CmtMethodSource getInitMethod(String modelType);
```

Retrieve a reference to the source for the UI initialization method.

`modelType` is the component model type.

```
public CmtModelNode getLastDesignedNode();
```

Obtain a reference to the node last selected for designing.

```
public CmtModel getModel(String name, CmtSubcomponent
subcomponent);
```

Find a model and return a reference to it, or `null` if it cannot be found.

`name` is the name of the model to locate.

`subcomponent` is the subcomponent to look for within that model.

```
public CmtModel[] getModels();
```

```
public CmtModel[] getModels(CmtSubcomponent subcomponent);
```

```
public CmtModel[] getModels(String name);
```

Obtain a list of all the models for this component. This list may be filtered by the supplied parameters, with an empty array resulting when no matches are found. If `release` has been called on this component, then this method returns `null`.

`subcomponent` is the subcomponent to find within the returned models.

`name` is the name of the model(s) to locate.

```
public DefaultTreeModel getModelTree();
```

Get the model tree for this component.

```
public String getName();
```

Returns the name of this component.

```
public JotSourceFile getSourceFile();
```

Retrieve a reference to the source file that contains this component.

```
public CmtSubcomponent getSubcomponent(String name);
```

```
public CmtSubcomponent[] getSubcomponents();
```

Find one or all of the subcomponents for this component. These methods return null or an empty array if no subcomponents are found.

name is the name of a particular subcomponent to locate.

```
public void removeComponentSourceListener(
    CmtComponentListener listener);
```

Remove an object being notified of events in this component.

listener is the object to delete.

```
public void removeMethod(CmtMethod method);
```

```
public void removeModel(CmtModel model);
```

```
public void removeProperty(CmtProperty property);
```

```
public void removeSubcomponent(CmtSubcomponent
    subcomponent);
```

Delete an attribute from this component (or component tree for a model).

method, model, property, or subcomponent identify the attribute to remove.

```
public void renameSubcomponent(CmtSubcomponent
    subcomponent, String newName) throws
    IllegalArgumentException;
```

Replace all references within the file to a subcomponent with a new name. An exception is thrown if the file is readonly, or if the new name is empty, contains illegal characters, or is already in use.

subcomponent identifies the subcomponent to locate.

newName is the new name for this subcomponent.

```
public void setLastDesignedNode(CmtModelNode node);
```

Keep track of the last node selected so that it can be the default when the designer is re-opened.

node is the node selected.

These fields also appear in this interface:

```
public static final String INIT_METHOD_NAME;
```

The name of the method JBuilder uses for UI initialization (“jbInit”) is found here.

```
public static final JotClass INIT_METHOD_PARAMS[];
```

This field contains the list of parameter types for the method above.

```
public static final String JBOWNER_METHOD_NAME;
```

This field contains the name of the method to call to set the owner module.

```
public static final String VA_INIT_METHOD_NAME;
```

The name of the Visual Age UI initialization method is found here, being “initialize”.

CmtComponentListener Interface

Events occurring within a component are reported to interested parties that implement the `com.borland.jbuilder.cmt.CmtComponentListener` interface. Create an instance for this interface and register it with the `CmtComponentSource`:

```
cmtComponentSource.addComponentSourceListener(
    new MyCmtComponentListener());
```

Its methods appear below:

```
public void componentChanged(CmtComponentEvent event);
```

This method is called when the component's contents change.

`event` holds the details about the change.

```
public void eventChanged(CmtComponentEvent event);
```

```
public void methodChanged(CmtComponentEvent event);
```

```
public void propertyChanged(CmtComponentEvent event);
```

When an event, method, or property of the component changes these events fire.

`event` holds the details about the change.

```
public void subcomponentChanged(CmtComponentEvent event);
```

Changes to a subcomponent within a component trigger this method.

`event` holds the details about the change.

CmtSubcomponent Interface

Representing an instance of a `CmtComponent` within a class is the `com.borland.jbuilder.cmt.CmtSubcomponent` interface. It associates a variable declaration with the component type and lets you get at the attributes of that instance. Note that the implicit reference to the current class, `this`, is also included as a subcomponent.

You can access the subcomponents for a file through the `CmtComponentSource` interface, or directly from the `DesignerViewer` class (the one behind the Design tab) as follows:

```
DesignerViewer viewer = (DesignerViewer)browser.getViewerOfType(
    browser.getActiveNode(), DesignerViewer.class);
if (viewer != null) {
    CmtSubcomponent[] subcomps = viewer.getSubcomponents();
}
```

Through the `CmtSubcomponents` returned above you can interact with the code that underlies them, reviewing or altering their types, initialization, and properties.

Its methods are shown here:

```
public void addPropertyChangeListener(
    PropertyChangeListener listener);
```

Include an object interested in changes to property values within this subcomponent.

`listener` is the object to notify.

```

public void addPropertyState(CmtPropertyState
    propertyState);
    Add a property state for this subcomponent.
    propertyState is the new setting to include.
public CmtSubcomponent copy(CmtComponentSource toFile,
    HashMap subcomponentList, boolean toClipboard);
    Copy this subcomponent into another file and return a reference to it.
    toFile is the source file to copy the subcomponent to.
    subcomponentList is the list of subcomponents to copy.
    toClipboard is true to use the clipboard during the copy, or false to not use
    it.
public void firePropertyChange(String propertyName, Object
    oldValue, Object newValue);
    Notify registered listeners that a property has changed its value.
    propertyName is the name of the property that is changing.
    oldValue and newValue are the before and after values for the property.
public Container getAsContainer();
    Retrieve this subcomponent as a live container object. Normally this is the
    same as getLiveInstance, except when containership is delegated to
    another object, in which case that object is returned.
public JotAssignment getAssignment();
    Find the JOT assignment expression for this subcomponent.
public CmtComponent getComponent();
    Returns the CMT component for this subcomponent.
public JotClass getComponentType();
    Obtain a reference to the JOT class type for this subcomponent.
public Dialog getCustomizerDialog();
    Locate the customizer dialog for this subcomponent or null if there is none.
    If this customizer updates any settings it should call setNeedsSerialize
    to indicate that these should be saved.
public JotClass getDeclaredClass();
    Retrieve a reference to the JOT class for this subcomponent. This is the same
    as getComponent.getType unless it is an array, in which case you get a
    reference to the array type. For the self-referencing subcomponent this, it is
    the ancestor class.
public CmtEventState getDefaultEventState();
public CmtPropertyState getDefaultPropertyState();
    Get the state of the default event or property for this subcomponent.
public CmtEventState getEventState(String name);
public CmtEventState[] getEventStates();
    Find one or all of the events for this subcomponent. A null or an empty
    array results if the requested events are not found.
    name is the name of an event to locate.
public JotExpression getInitializer();
    Returns the JOT initialization expression for this subcomponent, usually
    something like "new JButton()".

```

```
public CmtMethodSource getInitMethod();
```

Locate the method from this subcomponent's owner where the subcomponent is initialized.



WARNING

This should always be the standard UI initialization method (`jbInit`), but if the subcomponent has properties set elsewhere, that method may be found instead.

```
public JotClass getLiveClass();
```

Obtain a reference to the actual JOT class for this subcomponent.

```
public Object getLiveInstance();
```

```
public Object getLiveInstance(boolean create);
```

Get a reference to the actual object represented by this subcomponent, or null if it does not yet exist.

`create` is true (the default) to construct an appropriate object if it does not already exist, or false to not create a new one.

```
public CmtMethodCall[] getMethodCalls();
```

Retrieve all the method calls, including property settings, from the UI initialization method that have this subcomponent as the owner. For example, this list may include:

```
deleteButton.setText("Delete");
deleteButton.setMnemonic('D');
```

```
public String getName();
```

Returns the name of the variable for this subcomponent, such as “delete-Button”.

```
public CmtComponentSource getOuterComponent();
```

Find the owner component for this subcomponent. This is usually the main component in the file, even for `this`.

```
public CmtPropertyState getPropertyState(String name);
```

```
public CmtPropertyState[] getPropertyStates();
```

Obtain the states of one or all of the properties of this subcomponent. This list may include those with default values. A null or an empty array is returned if the desired properties are not found.

`name` is the name of a property to find.

```
public int getScope();
```

Discover the scope of the variable for this subcomponent – one of the constants defined in this interface.

```
public String getSourceName();
```

The “name” of this variable if it is a method call instead of just a variable, such as “`dm1.getDataSet().getQuery()`”. For the main class within a file this has the value “`this`”.

```
public boolean isNeedsSerialize();
```

Returns true if this subcomponent should be serialized with its new settings, or false if no significant changes have been made.

```
public void release();
```

Notification that the subcomponent is about to fall out of scope. Release any resources that were held from here.

```

public void releaseLiveInstance();
    Free up the actual object represented by this subcomponent.
public void removePropertyChangeListener(
    PropertyChangeListener listener);
    Delete a listener for changes to property settings.
    listener is the previously registered object being notified.
public void removePropertyState(CmtPropertyState
    propertyState);
    Remove a property setting from this subcomponent.
    propertyState is the setting to delete.
public void serialize() throws NotSerializableException,
    IOException;
public void serialize(OutputStream outStream) throws
    NotSerializableException, IOException;
    Serialize the subcomponent.
    outStream is the stream to write to. If not specified, a file with a .ser
    extension is written to in the same location as the original source file.
public void setAssignment(JotAssignment assign);
public void setAssignment(String valueText);
    Establish the assignment expression for this subcomponent.
    assign is the expression as a JOT object.
    valueText is the expression as text.
public void setCustomizerDialog(Dialog dialog);
    Set the dialog to use when customizing this subcomponent.
    dialog is the customizer to display.
public void setInitializer(String init);
    Modify the initialization expression for this subcomponent.
    init is the new expression.
public void setLiveClass(String className);
    Establish the class for this subcomponent.
    className is the full name of the class.
public void setLiveInstance(Object instance);
    Update the actual object that this subcomponent represents.
    instance is the live object.
public void setNeedsSerialize(boolean yesNo);
    Alter the serialization requirements of this subcomponent.
    yesNo is true to indicate that the subcomponent has changed and should be
    re-serialized, or false to denote no changes.
public void setScope(int scope);
    Set the scope of this subcomponent's variable.
    scope is one of the constants below from this interface.

```

These fields are also available in this interface:

```

public static final int CLASS_SCOPE;
public static final int METHOD_SCOPE;

```

The scope of a variable: the whole class or just the UI initialization method.

CmtFeature Interface

The `com.borland.jbuilder.cmt.CmtFeature` interface brings together several abilities that apply to the attributes of a subcomponent: property, event, or method.

These common abilities are:

```
public CmtComponent getComponent();
```

Retrieve the component that owns this attribute.

```
public String getDisplayName();
```

Obtain the display name for this component.

```
public String getName();
```

Get the name of the instance of this component.

```
public String getShortDescription();
```

Returns a short description for the component.

```
public boolean isExpert();
```

```
public boolean isHidden();
```

These methods indicate whether or not the attribute is shown when in expert or hidden modes in the Inspector.

CmtProperty Interface

The `com.borland.jbuilder.cmt.CmtProperty` interface defines a property of a component in a generic sense. It derives from `CmtFeature`. If you are after the value of a property for a particular component, you need to use the `CmtPropertyState` interface instead.

Find the properties for a component through the `CmtComponent` interface:

```
CmtProperty prop = component.getProperty("maximumSize");
```

or

```
CmtProperty[] props = component.getProperties();
```

NOTE

The names of some properties have special meanings to JBuilder, identifying pseudo-properties of a subcomponent. For instance, the name of a subcomponent (its variable's name) is not actually a property of the component, but is needed by JBuilder for other purposes. Similarly, the constraints used to add a component to a layout do not belong to that component, but are associated with it. The names for these values are surrounded by angle brackets (`< >`) with the first character following the opening bracket indicating a sort order amongst names of this type. For example, the component's variable name is held in a property called "`<Aname>`", while its constraints may be called "`<Bconstraints>`". They are displayed at the top of the Inspector with their names bolded.

The methods of this interface are:

```
public PropertyEditor getEditor();
```

Retrieve the property editor for this property, or `null` if there is none specified. Any editor is defined as part of the `BeanInfo` for the component.

```
public JotMethod getReadMethod();
```

Locate the method within the component that reads the property's value.

```
public JotClass getType();
```

Find out the type (in JOT terminology) of this property.

```
public JotMethod getWriteMethod();
```

Get the component method that updates this property's value.

```
public boolean isBound();
```

```
public boolean isConstrained();
```

Determine whether this property is bound or constrained with these methods.

```
public boolean isReadable();
```

```
public boolean isWritable();
```

Discover whether or not this property is readable or writable through these methods.

The `CmtEvent` interface derives directly from `CmtProperty` and adds nothing new. It simply serves as a marker for events as opposed to properties.

CmtPropertySource Interface

Extending `CmtProperty` is the `com.borland.jbuilder.cmt.CmtPropertySource` interface, which allows you to alter some of the attributes of a property. Usually you would just cast a `CmtProperty` instance to this one after checking that it is available.



UNDOCUMENTED

This class is undocumented, even though there are links to it from other related classes within the documentation.

Its new methods are:

```
public void setName(String name);
```

Change the name for this property.

`name` is the new name.

```
public void setReadable(boolean readable);
```

Update whether this property is write-only.

`readable` is true if the property value can be read, or false if it cannot.

```
public void setType(JotClass jotClass);
```

Alter the type of the property value.

`jotClass` is the new type (in JOT format).

```
public void setWritable(boolean writable);
```

Make the property readonly or available for alteration through this method.

`writable` is true if the property value can be updated, or false if it cannot.

The `CmtEventSource` interface descends directly from `CmtPropertySource` and again adds nothing new, acting as a marker for events.

CmtPropertyState Interface

Each property of a subcomponent is embodied in a `com.borland.jbuilder.cmt.CmtPropertyState` interface object. There may not be a corresponding entry in the UI initialization method, as occurs if this property contains its default value. The presence of an associated `CmtPropertySetting` object denotes a matching line of code.

Find the property states for a subcomponent through the `CmtSubcomponent` interface:

```
CmtPropertyState propState = subcomponent.getPropertyState("maximumSize");
```

or

```
CmtPropertyState[] propStates = subcomponent.getPropertyStates();
```

The methods of this interface are:

```
public CmtProperty getProperty();
```

Get the property to which this value applies.

```
public CmtPropertySetting getPropertySetting();
```

Obtain a reference to the property setting, or null if none is defined.

```
public CmtSubcomponent getSubcomponent();
```

Retrieve the subcomponent to which this property value applies.

```
public Object getValue();
```

Read the live value for this property. For a dimension property this would be a `Dimension` object.

```
public String getValueSource();
```

Get the property value as source code. For a dimension property this might be “new `Dimension(51, 21)`”.

```
public String getValueText();
```

Obtain the property’s value as display text, using the associated property editor. For a dimension property this might be “51, 21” and would be shown in the Inspector.

```
public boolean isDefault();
```

Returns true if this value is the default for the property, or false if it is not. Default values do not have code written out for them.

```
public boolean isPseudoPropertyState();
```

For properties that do not map directly onto the underlying component, this method returns true. Otherwise it returns false. The “<Aname>” property that `JBuilder` adds for the variable name would return true here. See the note under `CmtProperty` for more details.

```
public boolean isReadOnly();
```

Returns true if this property is readonly, or false if it can be altered.

```
public void reset();
```

Change the property value back to its default.

```
public void setDefaultValue(Object value);
```

Update the default value for the property. When set to this value, no code is written for the property. Sometimes it is necessary to change this to force a value to be written.

value is the new default value.

```
public void setValue(Object value) throws  
PropertyVetoException;
```

Establish the new value for this property as an object.

value is the new value object.

```
public void setValueSource(String value);
```

Set the property value as source code.

value is the new value source code.

```
public void setValueText(String value);
```

Alter the property's value as a text expression, via the associated property editor.

value is the new value as text.

```
public void triggerPropertyChange();
```

Notify interested parties that the property value has changed.

The `CmtEventState` interface derives directly from `CmtPropertyState` and adds a single method. It mostly serves to identify event details rather than property ones.

```
public String getDefaultHandlerText();
```

Get the text for the default event handler, such as "deleteButton_actionPerformed".

CmtPropertySetting Interface

Properties with values other than their default have a corresponding line of code in the UI initialization method to establish that setting. This is encapsulated by the `com.borland.jbuilder.cmt.CmtPropertySetting` interface. You retrieve an instance of this interface from the `CmtPropertyState` object for the property:

```
CmtPropertySetting setting = propertyState.getPropertySetting();
```



UNDOCUMENTED

This class is undocumented, even though there are links to it from other related classes within the documentation.

```
public CmtMethodCall getMethodCall();
```

Returns a reference to the method call that sets this property's value. For a dimension property this may encapsulate the code: "deleteButton.setMaximumSize(new Dimension(51, 21))".

```
public CmtProperty getProperty();
```

Obtain a reference to the property associated with this setting.

```
public CmtSubcomponent getSubcomponent();
```

Get back to the subcomponent whose property this setting applies to.

```
public Object getValue();
```

Retrieve the value of this property as an object. For a dimension property this would be a `Dimension` object.

```
public String getValueSource();
```

Returns the source code for this property value. For a dimension property this might be "new Dimension(51, 21)".

```
public void setValueSource(String value);
```

Update the source code for this property value.

value is the new setting code.

CmtModel Interface

The model of the UI component hierarchy is accessible through the `com.borland.jbuilder.cmt.CmtModel` interface, which maps onto the tree

structure displayed in the Structure Pane.

If you define your own designer enhancement, you usually implement this interface to control which components it manages. Within the `annotate` method of the `Designer` interface you would construct your model by scanning through the components in the current class. Then you add the result to the `CmtComponentSource` object provided. See Chapter 21 for more information on interacting with the JBuilder designers.

The methods for this interface are shown below:

```
public CmtModelNode add(CmtModelNode parent, String
    className, CmtModelNode aheadOf, CmtSubcomponent
    subcomponent);
```

Add a new subcomponent to this model, and return a reference to its node. In this method you may need to add text for the new component to the source file for the main class.

`parent` is the node beneath which the new node is added.

`className` is the full name for the class of the component being added.

`aheadOf` is an existing node to mark the position of the new node. If `null`, the new node appears at the end of the list of children.

`subcomponent` is the CMT wrapper for the component if it has already been claimed by another model. It is `null` if this model is the primary one.

```
public void close();
```

Close down the model and release any resources it may have acquired.

```
public CmtModelNode[] getChildren(CmtModelNode parent);
```

Get a list of the child nodes for a given node. If there are no children, an empty array should be returned.

`parent` is the node whose children are retrieved.

```
public CmtComponent getComponent();
```

Obtain a reference to the component represented by this model.

```
public DefaultTreeModel getGraph();
```

Gain access to the model underlying the tree displayed in the Structure Pane through this method.

```
public String getName();
```

Return the name of this model.

```
public CmtModelNode getRoot();
```

Retrieve the root of this model.

```
public boolean isMultiInstance();
```

Determine whether this model is multi-instance, returning `true` if it is, or `false` if it is not.

```
public boolean isSubcomponentOwned(CmtSubcomponent
    subcomponent);
```

Does this model claim ownership of a subcomponent? If this method returns `true` then the subcomponent only appears within this model, and is not available to any other models.

`subcomponent` is the component being tested.

```
public CmtModelNode move(CmtModelNode node, CmtModelNode
    newParent, CmtModelNode aheadOf);
```

Move an existing node in the tree from one branch to another, returning a reference to that node.

`node` is the node to move, as well as the return value.

`newParent` is the node that becomes the new parent for the moved node.

`aheadOf` is an existing node beneath that parent to position the new node before. If `null`, the node is moved to the end of the new parent's list of children.

```
public void remove(CmtModelNode node);
```

Delete a node from the model.

`node` is the node to delete.

CmtModelNode Interface

Whereas `CmtModel` defines the model as a whole, each node within that model is an instance of the `com.borland.jbuilder.cmt.CmtModelNode` interface. You implement this interface on the nodes that make up your model (see above).

Each node may want to listen in on property changes to the associated subcomponent so that you can update the tree display when a component's details are altered. To do this, implement the `PropertyChangeListener` interface on this class as well, and use code like the following in its constructor:

```
if (subcomponent != null) {
    subcomponent.addPropertyChangeListener(this);
}
```

This interface extends the standard `TreeNode` one, adding the methods below:

```
public CmtModel getCmtModel();
```

Retrieve the model that owns this node via this method.

```
public CmtSubcomponent getSubcomponent();
```

Access the subcomponent that this node represents within the tree through this method.

```
public String getTag();
```

Return the name of an icon to use for this node in case its component does not supply one through its `BeanInfo`, or `null` to have no backup icon. This name was previously registered with the Component Tree by calling `addImage` in `com.borland.jbuilder.designer.tree.ComponentViewTreeCellRenderer`.

```
public boolean isDesignable();
```

Enable or disable the **Activate Designer** menu entry on the designer context menu for this node. Return `true` to enable it, or `false` to disable it.

CMT Example

To demonstrate some of the information available through the CMT, the example presented here just dumps details about subcomponents in the current node to the output stream (see Listing 20–1). It creates two menu items on the Tools menu to display a short or full version of the information. These menu items are only enabled when the Design tab is open for a node, since the designer viewer is used to retrieve the initial CMT references. The `update` method of the underlying `BrowserAction` class lets you disable the menu entries when necessary.

Listing 20–1. Dumping CMT information.

```

package wood.keith.opentools.cmtdump;

import com.borland.jbuilder.JBuilderMenu;
import com.borland.jbuilder.cmt.CmtEventState;
import com.borland.jbuilder.cmt.CmtPropertyState;
import com.borland.jbuilder.cmt.CmtSubcomponent;
import com.borland.jbuilder.designer.DesignerViewer;
import com.borland.primetime.PrimeTime;
import com.borland.primetime.ide.Browser;
import com.borland.primetime.ide.BrowserAction;
/**
 * Demonstrate navigation through the CMT hierarchy in JBuilder.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 23 April 2002
 */
public class CMTDump extends BrowserAction {

    /**
     * Add the action to the menu.
     *
     * @param majorVersion the major version of the current OpenTools API
     * @param minorVersion the minor version of the current OpenTools API
     */
    public static void initOpenTool(byte majorVersion, byte minorVersion) {
        if (majorVersion != PrimeTime.CURRENT_MAJOR_VERSION) {
            return;
        }
        JBuilderMenu.GROUP_Tools.add(new CMTDump(false));
        JBuilderMenu.GROUP_Tools.add(new CMTDump(true));
    }

    private boolean full = false;
    /**
     * Initialise the action.
     *
     * @param full true to dump all attributes,
     *             false to just get the type and name
     */
    public CMTDump(boolean full) {
        super("CMT Dump - " + (full ? "Full" : "Short"), 'D',
            "Dump the " + (full ? "full" : "short") + " CMT hierarchy");
        this.full = full;
    }

    /**
     * Start dumping the CMT hierarchy for the current class.
     *
     * @param browser is the active browser
     */
    public void actionPerformed(Browser browser) {
        DesignerViewer viewer =
            (DesignerViewer)browser.getActiveViewer(browser.getActiveNode());
        CmtSubcomponent[] subcomps = viewer.getSubcomponents();
        for (int index = 0; index < subcomps.length; index++) {
            dumpComponent(subcomps[index]);
        }
    }

    /**
     * Dump the CMT details for the current class.
     *
     * @param component is the current component
     */
    private void dumpComponent(CmtSubcomponent component) {
        if (full) {
            System.out.println("-----");
            System.out.println(

```

```

    component.getComponent().getLiveClazz().getName());
// List properties and their settings
System.out.println("Properties:");
CmtPropertyState[] properties = component.getPropertyStates();
for (int index = 0; index < properties.length; index++) {
    System.out.println(
        " " + properties[index].getProperty().getName() +
        " = " + properties[index].getValueText() +
        " | " + properties[index].getValue() +
        " | " + properties[index].getValueSource() +
        " | " + properties[index].isDefault() +
        " | " + properties[index].isPseudoPropertyState());
    if (properties[index].getPropertySetting() != null) {
        System.out.println("    Setting: " +
            properties[index].getPropertySetting().getValue() + " | " +
            properties[index].getPropertySetting().getValueSource() +
            " | " + properties[index].getPropertySetting().
                getMethodCall().getJotMethodCall().getText());
    }
}
// List events and their settings
System.out.println("Events:");
CmtEventState[] events = component.getEventStates();
for (int index = 0; index < events.length; index++) {
    System.out.println(
        " " + events[index].getProperty().getName() +
        " = " + events[index].getValueText() +
        " | " + events[index].getValue() +
        " | " + events[index].getValueSource() +
        " | " + events[index].getDefaultHandlerText() +
        " | " + events[index].isDefault() +
        " | " + events[index].isPseudoPropertyState());
    if (events[index].getPropertySetting() != null) {
        System.out.println("    Setting: " +
            events[index].getPropertySetting().getValue() + " | " +
            events[index].getPropertySetting().getValueSource() +
            " | " + events[index].getPropertySetting().
                getMethodCall().getJotMethodCall().getText());
    }
}
// List other miscellaneous settings
System.out.println("Other");
System.out.println("  Assignment: " +
    (component.getAssignment() == null ? "null" :
    component.getAssignment().getText()));
System.out.println("  Declared class: " +
    (component.getDeclaredClass() == null ? "null" :
    component.getDeclaredClass().getName()));
System.out.println("  Default event state: " +
    (component.getDefaultEventState() == null ? "null" :
    component.getDefaultEventState().getProperty().getName() +
    " = " + component.getDefaultEventState().getValueText()));
System.out.println("  Default property state: " +
    (component.getDefaultPropertyState() == null ? "null" :
    component.getDefaultPropertyState().getProperty().getName() +
    " = " + component.getDefaultPropertyState().getValueText()));
System.out.println("  Initializer: " +
    (component.getInitializer() == null ? "null" :
    component.getInitializer().getText()));
System.out.println("  InitMethod: " +
    (component.getInitMethod() == null ? "null" :
    component.getInitMethod().getJotMethodSource().
        getCodeBlock().getText()));
System.out.println("  LiveClass: " +
    component.getLiveClass().getName());
System.out.println("  LiveInstance: " +
    component.getLiveInstance(true));
CmtMethodCall[] calls = component.getMethodCalls();
for (int index = 0; index < calls.length; index++) {
    System.out.println("  MethodCall " + index + ": " +

```

```

        calls[index].getJotMethodCall().getText());
    }
    System.out.println("  Name: " + component.getName());
    System.out.println("  OuterComponent: " +
        (component.getOuterComponent() == null ? "null" :
        component.getOuterComponent().getName()));
    System.out.println("  Scope: " + component.getScope());
    System.out.println("  SourceName: " + component.getSourceName());
}
else {
    // For short form, just list type and name
    System.out.println(
        component.getComponent().getLiveClazz().getName() + " " +
        component.getName());
}
}

/**
 * Alter the enabled/disabled setting for this action.
 *
 * @param browser is the active browser
 */
public void update(Browser browser) {
    setEnabled(browser.getActiveViewer(browser.getActiveNode())
        instanceof DesignerViewer);
}
}

```

The short form of the subcomponent dump simply lists the type and variable name for each subcomponent found in the open class. The full version starts by listing the subcomponent's type, followed by all of its properties, showing both name and value. Then comes the list of its events, again showing name and current value. Finally there is a collection of other miscellaneous details, including the initialization string for the variable, its scope, and the list of methods from the UI initialization routine that establish its attribute values. The full details output for one subcomponent is shown in Listing 20–2.

Listing 20–2. Subcomponent details.

```

javax.swing.JButton
Properties:
  <Aname> = deleteButton | deleteButton | deleteButton | false | true
  background = 204, 204, 204 | javax.swing.plaf.ColorUIResource[r=204,
g=204,b=204] | new Color(204, 204, 204) | true | false
  componentOrientation = | java.awt.ComponentOrientation@45105c | | true
  | false
  cursor = | java.awt.Cursor[Default Cursor] | | true | false
  dropTarget = | null | null | true | false
  enabled = True | true | propTable.getSelectedRow() > -1 | false | false
  Setting: null | propTable.getSelectedRow() > -1 | deleteButton.
  setEnabled(propTable.getSelectedRow() > -1)
  font = "Dialog", 1, 12 | javax.swing.plaf.FontUIResource[family=
dialog.bold,name=Dialog,style=bold,size=12] | new java.awt.Font(
"Dialog", 1, 12) | true | false
  foreground = Black | javax.swing.plaf.ColorUIResource[r=0,g=0,b=0] |
Color.black | true | false
  locale = <default> | en_AU | java.util.Locale.getDefault() | true |
false
  name = null | null | null | true | false
  visible = True | true | true | true | false
  alignmentX = 0.0 | 0.0 | (float) 0.0 | true | false
  alignmentY = 0.5 | 0.5 | (float) 0.5 | true | false
  autoscrolls = False | false | false | true | false
  border = Compound Border | javax.swing.plaf.BorderUIResource$
CompoundBorderUIResource@7d32cf | | true | false
  debugGraphicsOptions = <default> | 0 | 0 | true | false
  doubleBuffered = False | false | false | true | false
  maximumSize = 51, 21 | java.awt.Dimension[width=51,height=21] | new

```

```

Dimension(51, 21) | false | false
    Setting: java.awt.Dimension[width=51,height=21] | new Dimension(51,
21) | deleteButton.setMaximumSize(new Dimension(51, 21))
    minimumSize = 51, 21 | java.awt.Dimension[width=51,height=21] | new
Dimension(51, 21) | false | false
    Setting: java.awt.Dimension[width=51,height=21] | new Dimension(51,
21) | deleteButton.setMinimumSize(new Dimension(51, 21))
    nextFocusableComponent = | null | null | true | false
    opaque = True | true | true | true | false
    preferredSize = 51, 21 | java.awt.Dimension[width=51,height=21] | new
Dimension(51, 21) | false | false
    Setting: java.awt.Dimension[width=51,height=21] | new Dimension(51,
21) | deleteButton.setPreferredSize(new Dimension(51, 21))
    requestFocusEnabled = True | true | true | true | false
    toolTipText = Delete this property | Delete this property | "Delete this
property" | false | false
    Setting: Delete this property | "Delete this property" |
deleteButton.setToolTipText("Delete this property")
    actionCommand = Delete | Delete | "Delete" | true | false
    borderPainted = True | true | true | true | false
    contentAreaFilled = True | true | true | true | false
    disabledIcon = | null | null | true | false
    disabledSelectedIcon = | null | null | true | false
    focusPainted = True | true | true | true | false
    horizontalAlignment = CENTER | 0 | SwingConstants.CENTER | true | false
    horizontalTextPosition = TRAILING | 11 | SwingConstants.TRAILING | true
| false
    icon = | null | null | true | false
    margin = 2, 4, 2, 4 | java.awt.Insets[top=2,left=4,bottom=2,right=4] |
new Insets(2, 4, 2, 4) | false | false
    Setting: java.awt.Insets[top=2,left=4,bottom=2,right=4] | new
Insets(2, 4, 2, 4) | deleteButton.setMargin(new Insets(2, 4, 2, 4))
    mnemonic = D | 68 | 'D' | false | false
    Setting: 68 | 'D' | deleteButton.setMnemonic('D')
    model = | javax.swing.DefaultButtonModel@5c8f6d | | true | false
    pressedIcon = | null | null | true | false
    rolloverEnabled = False | false | false | true | false
    rolloverIcon = | null | null | true | false
    rolloverSelectedIcon = | null | null | true | false
    selected = False | false | false | true | false
    selectedIcon = | null | null | true | false
    text = Delete | Delete | "Delete" | false | false
    Setting: Delete | "Delete" | deleteButton.setText("Delete")
    verticalAlignment = CENTER | 0 | SwingConstants.CENTER | true | false
    verticalTextPosition = CENTER | 0 | SwingConstants.CENTER | true | false
    <Bconstraints> = [3, 7, 1, 1, 0.0, 0.0, 11, 2, [0, 0, 2, 0], 0, 0] |
java.awt.GridBagConstraints@4c72e3 | new GridBagConstraints(3, 7, 1, 1,
0.0, 0.0,GridBagConstraints.NORTH, GridBagConstraints.HORIZONTAL, new
Insets(0, 0, 2, 0), 0, 0) | false | true
    <CbuttonGroup> = <none> | null | null | false | true
Events:
    componentResized = null | null | null | deleteButton_componentResized |
true | false
    componentMoved = null | null | null | deleteButton_componentMoved | true
| false
    componentShown = null | null | null | deleteButton_componentShown | true
| false
    componentHidden = null | null | null | deleteButton_componentHidden |
true | false
    focusGained = null | null | null | deleteButton_focusGained | true |
false
    focusLost = null | null | null | deleteButton_focusLost | true | false
    hierarchyChanged = null | null | null | deleteButton_hierarchyChanged |
true | false
    ancestorMoved = null | null | null | deleteButton_ancestorMoved | true |
false
    ancestorResized = null | null | null | deleteButton_ancestorResized |
true | false
    keyTyped = null | null | null | deleteButton_keyTyped | true | false
    keyPressed = null | null | null | deleteButton_keyPressed | true | false

```

```

keyReleased = null | null | null | deleteButton_keyReleased | true |
false
mouseClicked = null | null | null | deleteButton_mouseClicked | true |
false
mousePressed = null | null | null | deleteButton_mousePressed | true |
false
mouseReleased = null | null | null | deleteButton_mouseReleased | true |
false
mouseEntered = null | null | null | deleteButton mouseEntered | true |
false
mouseExited = null | null | null | deleteButton_mouseExited | true |
false
mouseDragged = null | null | null | deleteButton mouseDragged | true |
false
mouseMoved = null | null | null | deleteButton_mouseMoved | true | false
inputMethodTextChanged = null | null | null |
deleteButton_inputMethodTextChanged | true | false
caretPositionChanged = null | null | null |
deleteButton_caretPositionChanged | true | false
propertyChange = null | null | null | deleteButton_propertyChange | true
| false
componentAdded = null | null | null | deleteButton_componentAdded | true
| false
componentRemoved = null | null | null | deleteButton_componentRemoved |
true | false
vetoableChange = null | null | null | deleteButton_vetoableChange | true
| false
ancestorAdded = null | null | null | deleteButton_ancestorAdded | true |
false
ancestorRemoved = null | null | null | deleteButton_ancestorRemoved |
true | false
ancestorMoved = null | null | null | deleteButton_ancestorMoved | true |
false
stateChanged = null | null | null | deleteButton stateChanged | true |
false
actionPerformed = null | null | null | deleteButton_actionPerformed |
true | false
itemStateChanged = null | null | null | deleteButton itemStateChanged |
true | false
Other
Assignment: null
Declared class: javax.swing.JButton
Default event state: actionPerformed = null
Default property state: null
Initializer: new JButton()
InitMethod: {
    this.setPageTitle("JSP Tag Class");
    try {
        this.setLargeIcon(new ImageIcon(
            JSPTagWizardPage.class.getResource(
                "/com/borland/jbuilder/wizard/jsp/JspWizardLarge.gif"));
    }
    catch (Exception ex) {
        // Ignore
    }
    this.setInstructions(
        "Enter a name for your class, what sort of functionality " +
        "it supports, and any properties that it accepts.");
    this.setLayout(borderLayout);

    jLabel1.setDisplayedMnemonic('K');
    jLabel1.setLabelFor(packageCombo);
    jLabel1.setText("Package");
    : // Other UI initialization code removed
    deleteButton.setToolTipText("Delete this property");
    deleteButton.setMargin(new Insets(2, 4, 2, 4));
    deleteButton.setMnemonic('D');
    deleteButton.setText("Delete");
    deleteButton.setPreferredSize(new Dimension(51, 21));
    deleteButton.setMinimumSize(new Dimension(51, 21));

```

```

deleteButton.setMaximumSize(new Dimension(51, 21));
deleteButton.addActionListener(
    new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            deleteButton_actionPerformed(e);
        }
    });
: // Other UI initialization code removed
this.add(bodyPanel, BorderLayout.CENTER);
bodyPanel.setLayout(gridBagLayout);

bodyPanel.add(jLabel1, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
    ,GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL,
    new Insets(0, 0, 2, 0), 0, 0));
: // Other UI initialization code removed
bodyPanel.add(deleteButton,
    new GridBagConstraints(3, 7, 1, 1, 0.0, 0.0
    ,GridBagConstraints.NORTH, GridBagConstraints.HORIZONTAL,
    new Insets(0, 0, 2, 0), 0, 0));
: // Other UI initialization code removed
}
LiveClass: javax.swing.JButton
LiveInstance:
javax.swing.JButton[,189,151,51x21,layout=javax.swing.OverlayLayout,alignm
entX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundB
orderUIResource@7d32cf,flags=1696,maximumSize=java.awt.Dimension[width=51,
height=21],minimumSize=java.awt.Dimension[width=51,height=21],preferredSiz
e=java.awt.Dimension[width=51,height=21],defaultIcon=,disabledIcon=,disabl
edSelectedIcon=,margin=java.awt.Insets[top=2,left=4,bottom=2,right=4],pain
tBorder=true,paintFocus=true,pressedIcon=,rolloverEnabled=false,rolloverIc
on=,rolloverSelectedIcon=,selectedIcon=,text=Delete,defaultCapable=true]
MethodCall 0: deleteButton.setToolTipText("Delete this property")
MethodCall 1: deleteButton.setMargin(new Insets(2, 4, 2, 4))
MethodCall 2: deleteButton.setMnemonic('D')
MethodCall 3: deleteButton.setText("Delete")
MethodCall 4: deleteButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        deleteButton_actionPerformed(e);
    }
})
MethodCall 5: deleteButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        deleteButton_actionPerformed(e);
    }
})
MethodCall 6: deleteButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        deleteButton_actionPerformed(e);
    }
})
MethodCall 7: deleteButton.setPreferredSize(new Dimension(51, 21))
MethodCall 8: deleteButton.setMinimumSize(new Dimension(51, 21))
MethodCall 9: deleteButton.setMaximumSize(new Dimension(51, 21))
MethodCall 10: deleteButton.setEnabled(propTable.getSelectedRow() > -1)
Name: deleteButton
OuterComponent: wood.keith.opentools.wizards.jsptags.JSPTagWizardPage
Scope: 1
SourceName: deleteButton

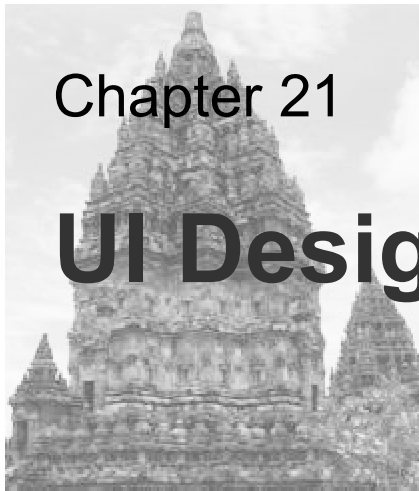
```

Summary

The Component Modeling Tool provides wrapper classes that let you delve into a Java source file and discover what components are declared therein. You can query them for attribute settings, and update these as necessary. CMT builds these abilities on top of the Java parsing supplied by the Java Object Toolkit (JOT).

As an example of its abilities, the demonstration program presented here just dumps the component contents of the current file to standard output. It lists each of the subcomponents within the file, along with their property values and event settings. If in summary mode, only the component's name is displayed.

The CMT is used as part of the following two chapters: in the UI designers that enable you to graphically layout your components, and in the layout assistants that provide feedback when working with layout managers in the designer.



Chapter 21

UI Designers

Within the Content Pane appears a set of tabs, one for each file that is opened, either along the top or the side of the pane. For each file there is a set of sub-tabs along the bottom of the pane that provide access to the various node viewers that apply to that file type (see Chapter 18 for more details on node viewers). Java source files have a Design tab appearing for them, allowing you to visually design the user interface portion of the class.

The Design tab is made up of several interacting parts (see Figure 21–1): the design surface itself, the Component Palette, the Inspector, and the Structure Pane. Components are selected from the Component Palette and then placed onto the design surface (representing the basic container of the main class). Various tabs within the palette let you organize the available components to make them easier to find.

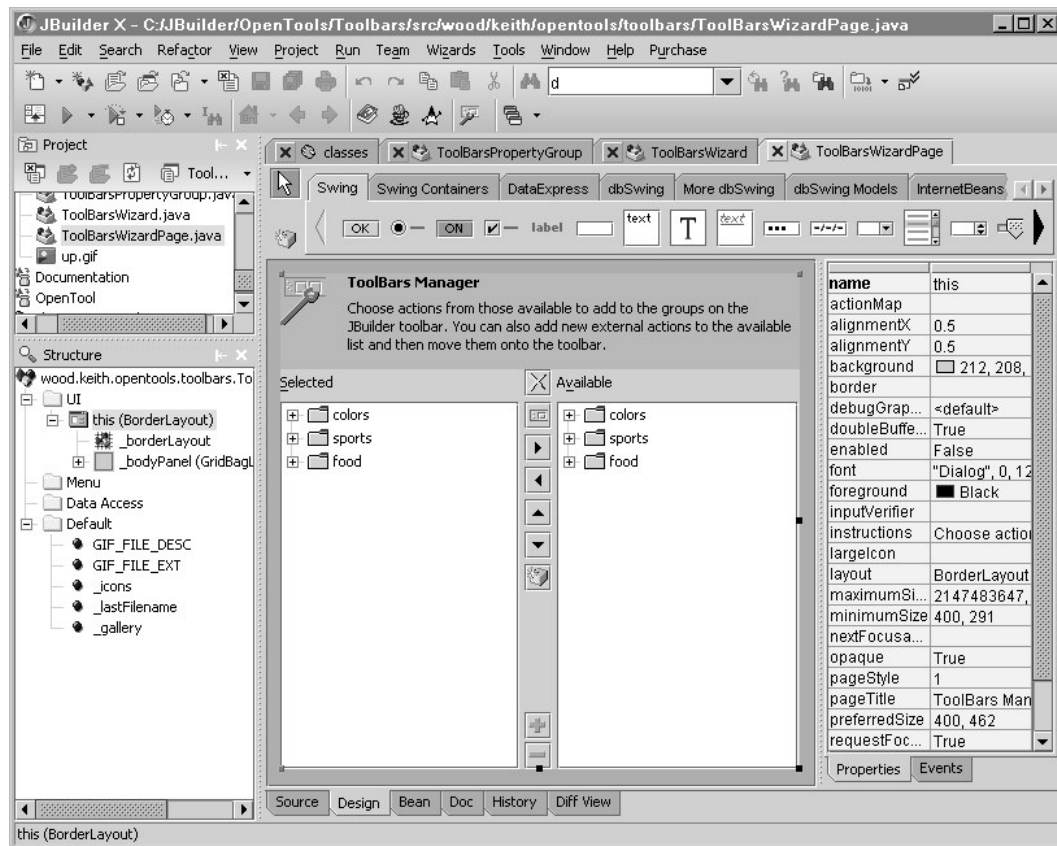
Once a component is selected you can drop it either directly onto the design surface or onto the Component Tree in the Structure Pane. In either case, the new component is assigned to a particular node within the Component Tree and may appear on the design surface as well (if it is a visual component).

Activating a component (by double-clicking it in the Component Tree, or by right-clicking and selecting **Activate Designer**) causes its associated designer to be invoked and to be informed of the selection of that component. The public JavaBean-style properties of the component can then be modified through the Inspector.

A *designer*, in the UI sense, provides support for different types of components within a class, letting you graphically interact with them. There are two parts to this support: identifying the managed components and presenting them in the Component Tree, and supplying a design surface for those components in the Content Pane to let you customize them. Although JBuilder has other tools called designers, this chapter only deals with those attached to the Design node viewer for Java files.

There are three basic designers supplied with JBuilder. The first is the UI designer (attached to objects in the UI category within the Component Tree) for visual components. It displays the component as it would appear at runtime, and handles any processing for layout managers (see Chapter 22) and contained controls.

Figure 21–1. The Design tab for a Java source file.



Secondly, there is the menu designer for objects under the Menu category. It lets you visually design a menu structure for your GUI, adding menu items as necessary and then setting their captions and abilities.

Lastly, there is the data access designer for objects under the Data Access category. It works with components from the DataExpress tab in the Palette, allowing you to define the fields that belong to data sources.

Any objects that are not claimed by one of the above designers are automatically picked up by the Other category. These objects have no designer associated with them; however, selecting them does load the Inspector with their property values and lets you modify them.

You can enhance the design experience by providing your own designers for certain components. Your designer must implement the `Designer` interface, and you must register an instance of it with the `DesignerManager` class.

```
DesignerManager.registerDesigner(new MyDesigner());
```

Typically your designer would add new nodes to the Component Tree within the Structure Pane. A multi-level hierarchy could be constructed if it is appropriate for the components being handled. For more functionality, the activation of one of these nodes can open up a customized designer that lets you easily interact with the selected object.

DesignerManager Class

The `com.borland.jbuilder.designer.DesignerManager` class provides your gateway into the designer processing of JBuilder. It lets you register interest in events that occur during the design process, as well as registering your new designer for inclusion as appropriate. The main methods used during the design phase are shown below. Note that many are static methods, being called directly from the class reference.

```
public static synchronized void addDesignerListener(
    DesignerListener listener);
```

Add a listener for events that occur during the design processing, such as opening and closing designers. See the next section for more details on the listener.

`listener` is the object interested in the designer events.

```
public static synchronized void addDesignerReleaseListener(
    DesignerReleaseListener listener);
```

To be notified when a component within a designer is being released, call this method. See the section below for the listener's events.

`listener` is the object to be informed of the release events.

```
public Designer getDesigner(CmtModel model);
```

Retrieve the designer appropriate to the supplied model.

`model` is the CMT wrapper around the model and its tree.

NOTE

See the previous chapter for more details on the Component Modeling Tool (CMT).

```
public ArrayList getDesignerViewers();
```

Obtain a list of all the current viewers. Each entry in the array is a `DesignerViewer` object.

VERSION

The `getDesignerViewers` method is not available in JBuilder 7.

```
public static DesignerManager getInstance();
```

Return the singleton instance of this class within JBuilder.

```
public void lookupHelp(String className, String
    propertyName);
```

Bring up help for the nominated class and property.

`className` is the full name of the class to locate the help for.

`propertyName` is the name of the particular property within that class to show.

```
public static synchronized void registerDesigner(Designer
    designer);
```

Register a new designer implementation with JBuilder so that it is invoked along with the standard ones when processing the components within the designer tab.

`designer` is an instance of the new designer implementation.

```
public static synchronized void removeDesignerListener(
    DesignerListener listener);
```

Remove a previously added listener for designer events with this method.

`listener` is the object that no longer wants notification of the designer events.

```
public static synchronized void
    removeDesignerReleaseListener(
        DesignerReleaseListener listener);
```

Stop listening to release events for the designers through this method.

`listener` is the object that no longer wants notification of the release events.

The `DesignerManager` has several other methods but these do not apply to the design process and so are not covered here.

DesignerListener and DesignerReleaseListener Interfaces

Various events that occur during the processing of the designer interfaces are available for interested classes to respond to. The `com.borland.jbuilder.designer.DesignerListener` interface lets you react to the opening and closing of designers, as well as other events as shown below. Register your interest through the `addDesignerListener` method of the `DesignerManager` class.

```
public void designerClosed(DesignerEvent event);
```

To respond when a designer is closed, implement this method.

`event` contains the details of the designer that closed.



WARNING

The `designerClosed` and `designerClosing` methods never seem to be called by JBuilder.

```
public void designerClosing(DesignerEvent event);
```

Notification of the closing of a designer, but before it has closed, comes through this method.

`event` contains the details of the designer being closed.

```
public void designerOpened(DesignerEvent event);
```

Find out about designers that have opened completely via this method.

`event` contains the details of the designer that opened. Use its `getComponent` method to retrieve the component being opened, and the `getContext` method for the `DesignerViewer` instance (the one behind the Design tab).

```
public void designerOpening(DesignerEvent event);
```

Implement this method to be informed of designers that are opening to prepare for their instantiation. The method is actually called twice before the designer fully opens.

`event` contains the details of the designer being opened. Find the component that caused the opening via the event's `getComponent` method and the `DesignerViewer` through the `getContext` method.

**NOTE**

The `designerOpening` method may be called more than once per opening.

```
public void designerShow(DesignerEvent event);
```

This method is called whenever the designer is shown or reactivated. It may be called many times.

`event` contains the details of the designer being shown, including the component that is being shown and its `DesignerViewer`.

```
public void designerToolSelected(DesignerEvent event);
```

Whenever a tool is chosen from the Component Palette, this method informs you of it.

`event` contains the details of the selection, in particular the name of the tool from its `getToolName` method.

**WARNING**

The `designerToolSelected` method never seems to be called by JBuilder.

To react to the release of a component from a designer instance, you need to implement the `com.borland.jbuilder.designer.DesignerReleaseListener` interface whose methods are listed below. Use the `addDesignerReleaseListener` method of the `DesignerManager` class to register your object to receive these calls.

```
public void designerReleased(DesignerEvent event);
```

Find out about designers that are finished with a component and have closed it through this method.

`event` contains the details of the component that closed.

```
public void designerReleasing(DesignerEvent event);
```

Before a component being shown by a designer is released, this method lets you prepare for its disappearance.

`event` contains the details of the component being closed.

To send off one of these events, for either interface, you call the `DesignerManager`'s `processDesignerEvent` method and pass along an appropriate instance of `DesignEvent` class.

DesignerEvent Class

This class encapsulates information about events within the designer environment. These events include the opening and closing of designers, showing or hiding the designer window, saving changes made within a designer, or selecting a component from the palette. They appear as parameters of the `DesignerListener` and `DesignerReleaseListener` method calls.

Depending on the type of event that occurs, you can retrieve information about the designer being opened or closed, the component being affected, or the tool being selected from the Component Palette. Use the corresponding methods from the list below to obtain these data.

```

public DesignerEvent(Object source);
public DesignerEvent(Object source, int id);
public DesignerEvent(Object source, boolean show);
public DesignerEvent(Object source, CmtComponentSource
    component);
public DesignerEvent(Object source, CmtComponentSource
    component, Designer designer);
public DesignerEvent(Object source, String toolName);

```

Create a new event instance from the parameters specified with one of these constructors. Typically these events are generated by JBuilder itself and are passed on to you through the appropriate listener methods.

`source` is the `DesignerViewer` instance associated with this event.

`id` is the event type based on one of the static field values from this class. If defaults to `COMMITTED` if no parameters other than `source` are given.

`show` is `true` when a component is being shown and `false` when it is being hidden. When this version of the constructor is used, the event type becomes `SHOW`.

`component` is the CMT wrapper for the component affected. If the `designer` is not specified when `component` is given the event type is set to `OPENING`.

`designer` is the `Designer` instance for this event. The event type is set to `OPENED` when this parameter is specified.

`toolName` is the name of the component selected from the Component Palette. In this case, the event type becomes `TOOL_SELECTED`.

```

public void dispatch(EventListener eventlistener);

```

Send this event on to the listener passed in, calling the appropriate method based on the event's type.

`eventListener` is the listener for these events. Note that it must be an implementation of either the `DesignerListener` or `DesignerReleaseListener` interfaces according to the type of the event.

```

public CmtComponentSource getComponent();

```

Retrieve the CMT wrapper for the component through this method.

```

public DesignerViewer getContext();

```

Find the designer viewer (the one behind the Design tab) associated with this event via this method.

```

public Designer getDesigner();

```

Return the designer implementation from this method. It should have a value for an `OPENED` event, and is `null` otherwise.

```

public boolean getShow();

```

This method returns `true` if the component is being shown and `false` if it is being hidden. It is `false` if the event type is not `SHOW`.

```

public String getToolName();

```

Retrieve the name of the component selected from the palette with this method. It is `null` if the event type is not `TOOL_SELECTED`.

```

public boolean isReleaseEvent();

```

Returns `true` if this event is either a releasing or a released event, or `false` for all other event types.

The constants defined in this class indicate what type of event is occurring:

```
public static final int CLOSED;
public static final int CLOSING;
```

A component has closed or is closing. In the latter case you should save any pending updates.

```
public static final int COMMITTED;
```

Any changes have been written out to the source file.

```
public static final int OPENED;
public static final int OPENING;
```

A component has been opened and a designer selected, or is about to be opened.

```
public static final int RELEASED;
public static final int RELEASING;
```

A component has closed or is closing (save any outstanding changes). These types track the release of subcomponents.

```
public static final int SHOW;
```

The designer window is being shown or hidden.

```
public static final int TOOL_SELECTED;
```

A tool from the palette has been chosen.

Designer Interface

To add new abilities to the designer process, you need to create a class that implements the `com.borland.jbuilder.designer.Designer` interface. This lets you add new items to the Component Tree within the Structure Pane and to bring up your own customized designer interface for the components that you manage.

```
public boolean activate(CmtComponentSource componentSource,
    CmtModelNode root, CmtModelNode node, DesignerViewer
    viewer);
```

Bring the view for the given model node up to date. It is called once when the designer is first opened, and then whenever the user requests it. The method returns true if the viewer can be activated, or false if it cannot.

Retrieve a reference to any existing viewer component through the `DesignerViewer` object. If one does not exist you should create a new instance and set it for this node.

```
InternetBeansViewer designer =
    (InternetBeansViewer)viewer.getDesignView(this);
if (designer == null) {
    // Create and register the UI for the designer
    designer = new InternetBeansViewer();
    viewer.setDesignView(this, designer);
}
```

`componentSource` is the source file reference for the components handled by this designer.

`root` is the CMT wrapper for root of the model sub-tree being opened. This is the first node below the category node within the Component Tree in the Structure Pane.

`node` is the CMT wrapper for the current model node being opened. It may be the same as the root node.

`viewer` is the instance of the viewer for this node.

```
public void annotate(CmtComponentSource componentSource,
    CmtComponents components);
```

This is the main method of the `Designer` interface that you need to implement as it allows you to add new nodes to the Component Tree in the Structure Pane. Within this method you typically create your top-level node and construct an appropriate tree structure beneath it (the *model*), corresponding to the components that this designer deals with. Components within the class are accessible through the following call:

```
CmtSubcomponent[] subcomponents = componentSource.getSubcomponents();
```

You can then walk through the array, picking out any components of interest and add them to your model.

At the end of the sub-tree creation, you call the `componentSource.addModel` method to add the hierarchy to the Component Tree for the file. You may add any number of sub-trees using this method.

The method is called when the designer is first opened for a file, and again whenever its nodes are altered. It should not perform any activities related to the viewer for this designer.

`componentSource` is the source file reference for the components handled by this designer.

`components` is an interface to the JOT representation of the source file.

```
public void close(CmtComponentSource componentsource,
    DesignerViewer viewer);
```

Finishes up the processing for a given source file and lets you release any UI components created. If you had registered for notification of changes to the Structure Pane tree, you should unregister at this time too.

```
viewer.getSelection().removeTreeSelectionListener(this);
```

`componentSource` is the source file reference for the components handled by this designer.

`viewer` is the instance of the viewer for this node.

```
public String getModelName();
```

Return the name for this designer's model. It must match the value returned by the `getName` method of that model.

```
public void open(CmtComponentSource componentsource,
    DesignerViewer viewer);
```

Create the UI for the designer when a file is opened.

```
MyViewer designer = (MyViewer)viewer.getDesignView(this);
if (designer == null) {
    designer = new MyViewer();
    viewer.setDesignView(this, designer);
}
```

You may also want to register to be notified of selection changes to the tree structure within the Structure Pane, as shown below:

```
viewer.getSelection().addTreeSelectionListener(this);
```

`componentSource` is the source file reference for the components handled by this designer.

`viewer` is the instance of the viewer for this node.

InternetBeansDesigner Example

To demonstrate how the UI designer API can be used to add functionality, you can develop support for the InternetBeans components available on the Palette tab of the same name. These controls assist in creating and handling the presentation layer of a Web application. You create a static JSP and embed the controls within it as placeholders for dynamic content, typically coming from a database.



NOTE

Since this example revolves around the InternetBeans components, you may not be able to compile or run it on some editions of JBuilder, such as the Personal edition.

InternetBeans fall into two main types: those that descend from `IxPageProducer` and those that descend from `IxComponent` (both in the `com.borland.internetbeans` package). The page producers read an HTML template file and parse its contents to fill in the dynamic portions. Meanwhile, the components represent HTML controls and generate appropriate tags for them within the page. Components are linked to a page producer and its HTML through their `pageProducer` property, and then to a tag on the page via the `controlName` or `elementId` properties.

By default, all of these controls appear beneath the Other category in the Structure Pane since no other node claims them. The designer developed in this example creates a new top-level node named **InternetBeans** that claims all the InternetBeans dropped onto the main class. Beneath that node it has a two-level structure: the top level consists of page producers and components not associated with a page producer, while the bottom level contains the components attached to each page producer.

When a page producer is selected and activated (by double-clicking or by right-clicking and selecting **Activate Designer**), the Content Pane loads and displays the HTML page attached to it (see Figure 21-2). The name of the component appears in a bar at the top of the design surface, while the HTML source is syntax highlighted below it. If no HTML file is attached to the page producer the design surface displays the text “Please set the HTML file name for this component” instead.

If a control component is selected and activated, its corresponding page producer and attached HTML file are loaded (if not already present). Then the HTML is searched for the identifier for this component and the section located is brought into view and highlighted (see Figure 21-3). In this way, you can easily find the position of a particular component within the page. If the component has not yet been assigned to a page producer, the design surface displays the text “Please set the page producer for this component”.

Figure 21–2. The InternetBeans designer extension.

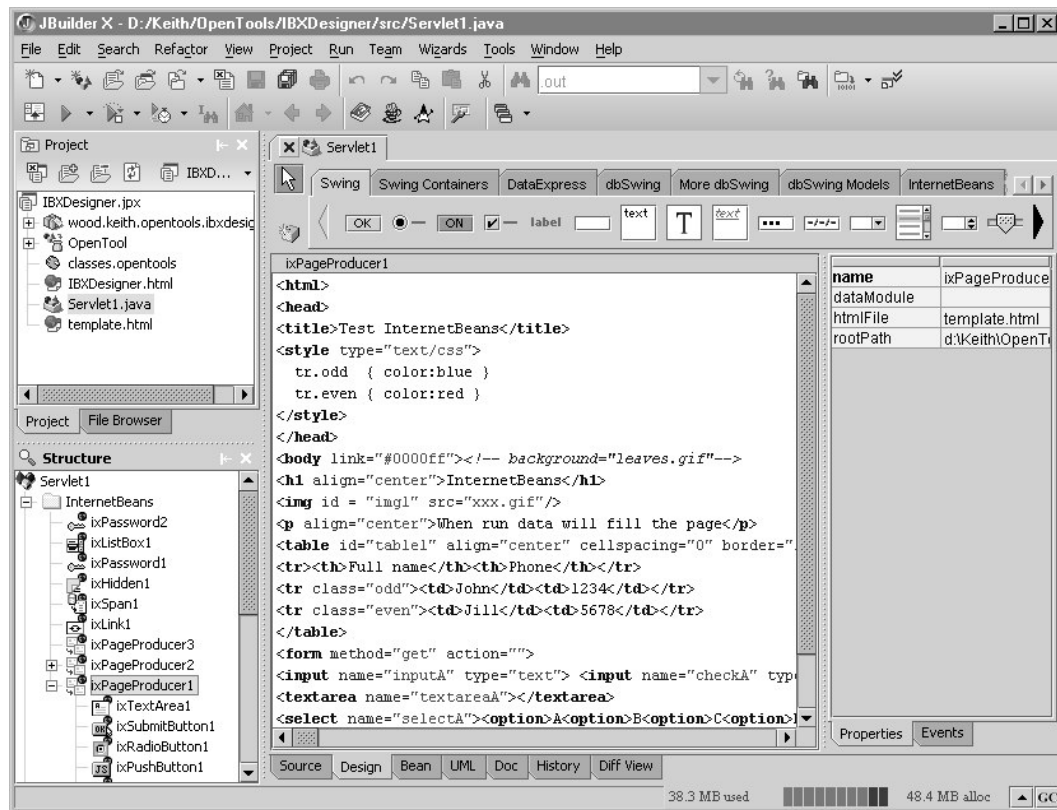
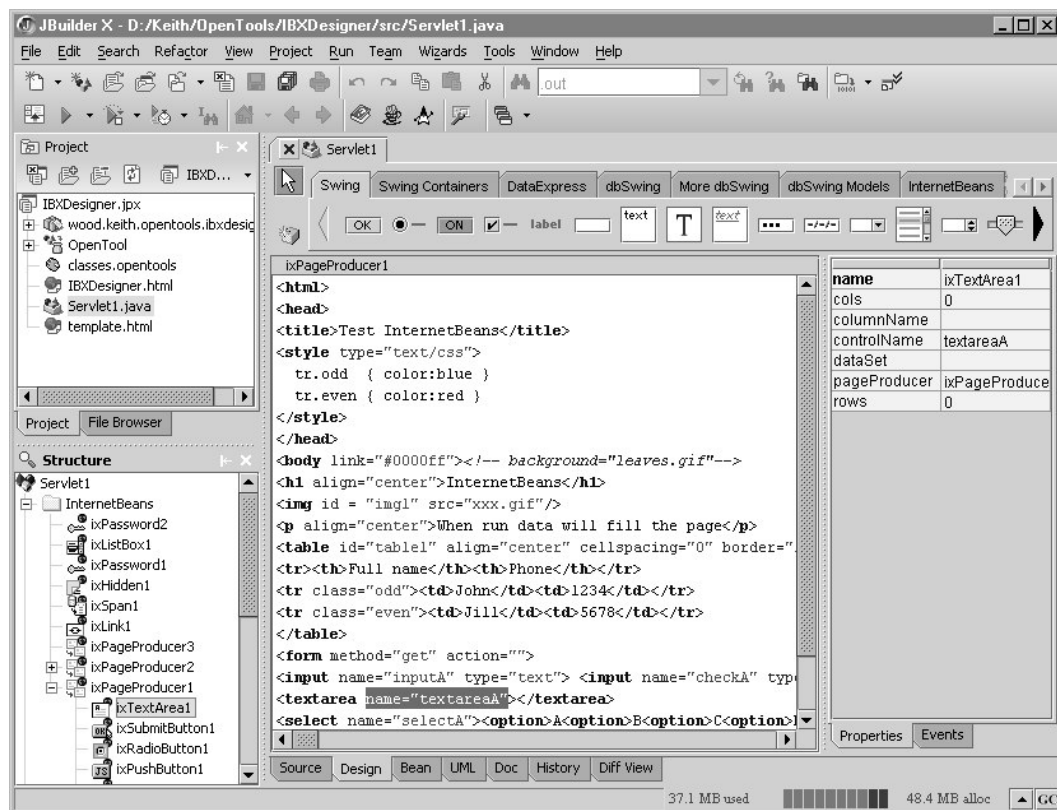


Figure 21–3. Selecting an InternetBeans component.



The starting point for the designer is a class that implements the Designer interface – the `InternetBeansDesigner` class in this case (see Listing 21–1). Create this new class and add the `OpenTools` initialization routine to register an instance of the class with the `DesignerManager` class.

Listing 21–1. The `InternetBeansDesigner` class.

```
package wood.keith.opentools.ibxdesigner;

import java.util.Enumeration;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;

import com.borland.internetbeans.IxComponent;
import com.borland.internetbeans.PageProducer;
import com.borland.jbuilder.cmt.CmtComponents;
import com.borland.jbuilder.cmt.CmtComponentSource;
import com.borland.jbuilder.cmt.CmtModelNode;
import com.borland.jbuilder.cmt.CmtSubcomponent;
import com.borland.jbuilder.designer.Designer;
import com.borland.jbuilder.designer.DesignerManager;
import com.borland.jbuilder.designer.DesignerViewer;
import com.borland.primetime.PrimeTime;

/**
 * A designer that manages InternetBeans components.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 10 September 2001
 */
public class InternetBeansDesigner
    implements Designer, TreeSelectionListener {

    private static final String VERSION = "1.0";

    private InternetBeansModelNode _rootNode = null;
    private DesignerViewer _viewer = null;

    /**
     * Register the new designer for the InternetBeans.
     *
     * @param majorVersion the major version of the current OpenTools API
     * @param minorVersion the minor version of the current OpenTools API
     */
    public static void initOpenTool(byte majorVersion, byte minorVersion) {
        if (majorVersion != PrimeTime.CURRENT_MAJOR_VERSION) {
            return;
        }
        DesignerManager.registerDesigner(new InternetBeansDesigner());
        if (PrimeTime.isVerbose()) {
            System.out.println("Loaded InternetBeans Designer v" + VERSION);
            System.out.println("Written by Keith Wood (kbwood@iprimus.com.au)");
        }
    }

    public InternetBeansDesigner() {
    }

    /**
     * Update the viewer for this designer.
     *
     * @param source the source reference that corresponds
     *               to the components
     * @param root the root of the subtree for this designer
     * @param node the node being opened
     * @param viewer the node viewer component
     * @return true if this node was handled, false otherwise
     */
}
```

```

*/
public boolean activate(CmtComponentSource source, CmtModelNode root,
    CmtModelNode node, DesignerViewer viewer) {
    viewer = viewer;
    InternetBeansViewer designer =
        (InternetBeansViewer)viewer.getDesignView(this);
    if (designer == null) {
        // Create and register the UI for the designer
        designer = new InternetBeansViewer();
        viewer.setDesignView(this, designer);
    }
    // Then set up the UI to display the current node
    Object object = root.getSubcomponent().getLiveInstance();
    designer.setPageProducer(
        (object instanceof PageProducer ? (PageProducer)object : null),
        root.getSubcomponent().getName());
    object = node.getSubcomponent().getLiveInstance();
    designer.setSelection(
        (object instanceof IxComponent ? (IxComponent)object : null));
    return true;
}

/**
 * Build a subtree for the components handled by this designer -
 * PageProducers and IxComponents from the InternetBeans collection.
 *
 * @param source          the source reference that corresponds
 *                        to the components
 * @param componentManager the manager of the components
 */
public void annotate(CmtComponentSource source,
    CmtComponents componentManager) {

    // Obtain an array containing all subcomponents (bean instance
    // variables) of the class being designed. The array will also
    // contain "this".
    CmtSubcomponent[] subcomponents = source.getSubcomponents();

    // Create a root model node object for the subtree.
    _rootNode = new InternetBeansModelNode(
        this, null, null, getModelName());

    // Create a tree structure for holding the structure of our model.
    // The root object of the tree is our root node.
    DefaultTreeModel tree = new DefaultTreeModel(_rootNode);

    // Now create the "model" itself.
    InternetBeansModel model =
        new InternetBeansModel(this, source, tree, getModelName());

    // Create additional nodes, one for each PageProducer, and attach them
    // to the tree, all as first level children of the root node.
    for (int index = 0; index < subcomponents.length; index++) {
        if (subcomponents[index].getLiveInstance()
            instanceof PageProducer) {
            tree.insertNodeInto(new InternetBeansModelNode(
                this, model, subcomponents[index]), _rootNode, 0);
        }
    }

    // Create additional nodes, one for each IxComponent,
    // and attach them into the tree under their page producer
    // (if available).
    for (int index = 0; index < subcomponents.length; index++) {
        if (subcomponents[index].getLiveInstance() instanceof IxComponent) {
            PageProducer producer =
                ((IxComponent)subcomponents[index].getLiveInstance()).
                getPageProducer();
            InternetBeansModelNode parent = _rootNode;
            Enumeration children = _rootNode.children();

```

```

        while (children.hasMoreElements()) {
            InternetBeansModelNode child =
                (InternetBeansModelNode)children.nextElement();
            if (child.getSubcomponent().getLiveInstance() == producer) {
                parent = child;
                break;
            }
        }
        tree.insertNodeInto(new InternetBeansModelNode(
            this, model, subcomponents[index]), parent, 0);
    }

    // Hand the model we made back to the source. This call,
    // source.addModel(), is the primary action that annotate()
    // performs back to the ComponentTree's structure.
    // You can add zero or more models.
    source.addModel(model, null);
}

/**
 * Tidy up after finished with the designer.
 *
 * @param source the source reference that corresponds
 *              to the components
 * @param viewer the node viewer component
 */
public void close(CmtComponentSource source, DesignerViewer viewer) {
    InternetBeansViewer designer =
        (InternetBeansViewer)viewer.getDesignView(this);
    if (designer != null) {
        designer.setPageProducer(null, "");
    }
    viewer.getSelection().removeTreeSelectionListener(this);
}

public String getModelName() { return "InternetBeans"; }

/**
 * Create the UI for the designer if necessary.
 *
 * @param source the source reference that corresponds
 *              to the components
 * @param viewer the node viewer component
 */
public void open(CmtComponentSource source, DesignerViewer viewer) {
    viewer.getSelection().addTreeSelectionListener(this);
    InternetBeansViewer designer =
        (InternetBeansViewer)viewer.getDesignView(this);
    if (designer == null) {
        designer = new InternetBeansViewer();
        viewer.setDesignView(this, designer);
    }
}

/**
 * Update the UI to show a particular PageProducer.
 *
 * @param producer the PageProducer to load
 * @param name      the component name to display
 */
public void setPageProducer(PageProducer producer, String name) {
    InternetBeansViewer designer =
        (InternetBeansViewer)_viewer.getDesignView(this);
    if (designer != null) {
        designer.setPageProducer(producer, name);
    }
}
}

```

```

* Update the UI to select a particular IxComponent.
*
* @param object the IxComponent to select within the HTML file
*/
public void setSelection(Object object) {
    InternetBeansViewer designer =
        (InternetBeansViewer)_viewer.getDesignView(this);
    if (designer != null) {
        designer.setSelection(
            object instanceof IxComponent ? (IxComponent)object : null);
    }
}

/**
* Respond to selections within the structure tree and
* update the UI accordingly.
*
* @param evt the event for the tree selection
*/
public void valueChanged(TreeSelectionEvent evt) {
    if (_viewer == null) {
        return;
    }
    try {
        TreePath[] paths = evt.getPaths();
        if (paths != null && paths.length >= 1) {
            Object object = paths[0].getLastPathComponent();
            if (object instanceof InternetBeansModelNode) {
                object = ((InternetBeansModelNode)object).getSubcomponent();
                object = (object == null ? null :
                    ((CmtSubcomponent)object).getLiveInstance());
                setSelection(object);
            }
        }
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

Next implement the methods of the Designer interface. The `activate` method creates a new UI component for the designer if necessary, and asks that component to prepare itself for display. If the current node is a page producer, its name is displayed and its associated HTML file is loaded and shown. If the current node is some other InternetBeans component, its location within its producer's file is highlighted.

The `annotate` method lets you add your model to the design tree in the Structure Pane. In this case you create a new model sub-tree, with its root being named `InternetBeans`. You then step through all of the sub-components within the source file and add any that are page producers directly beneath the root. A second pass through the sub-components lets you locate all the other InternetBeans components and either attaches them to their page producer, or directly to the root if they have no assigned producer. Finally, the new sub-tree is added back into the main tree with the following:

```
source.addModel(model, null);
```

Closing the designer lets you release any resources that you have acquired, and to remove any listeners that were added. For this example, you just clear out the display and unsubscribe to changes in the Component Tree.

The name of the model is used as the caption for the root node of the new sub-tree added by this designer. Note that this name is passed across to the

model when it is created in the `annotate` method to ensure that the two are identical.

Opening the designer gives you a chance to create whatever UI component is used to work with the selected components. Here you construct a new `InternetBeansViewer` component, as well as registering for changes to the Component Tree selection with the following:

```
viewer.getSelection().addTreeSelectionListener(this);
```

The `setPageProducer` and `setSelection` methods pass these calls onto the designer component, if it exists. This helps to de-couple the design since only the main designer class needs to know about the designer UI component.

Changes to the user's selection in the Component Tree are notified through the `valueChanged` method. The currently selected item is passed onto the viewer to allow it to update its display, but only if that item represents one of the components managed by this designer.

InternetBeansModel and InternetBeansModelNode Examples

A customized model manages the InternetBeans sub-tree within the Structure Pane. The `InternetBeansModel` class implements `CmtModel` and maintains the hierarchy internally via an embedded `DefaultTreeModel` (see Listing 21-2).

Listing 21-2. The InternetBeansModel class.

```
package wood.keith.opentools.ibxdesigner;

import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

import com.borland.internetbeans.IxComponent;
import com.borland.internetbeans.PageProducer;
import com.borland.jbuilder.cmt.CmtComponent;
import com.borland.jbuilder.cmt.CmtComponentSource;
import com.borland.jbuilder.cmt.CmtModel;
import com.borland.jbuilder.cmt.CmtModelNode;
import com.borland.jbuilder.cmt.CmtSubcomponent;

/**
 * The model that contains all the InternetBeans components.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 10 September 2001
 */
public class InternetBeansModel implements CmtModel {

    private InternetBeansDesigner _designer;
    private CmtComponentSource _source;
    private DefaultTreeModel _tree;
    private String _name;

    /**
     * Initialise the model that manages the subtree for this designer.
     *
     * @param designer the designer instance that created this model
     * @param source the source reference that corresponds to this model
     * @param tree the tree of nodes managed by this model
     * @param name the name of the model
     */
    public InternetBeansModel(InternetBeansDesigner designer,
```

```

    CmtComponentSource source, DefaultTreeModel tree, String name) {
        _designer = designer;
        _source = source;
        _tree = tree;
        _name = name;
    }

    public boolean isMultiInstance() { return false; }

    /**
     * Does this sub-tree claim and manage the subcomponent.
     * By claiming a subcomponent, you prevent it from appearing
     * under another sub-tree.
     *
     * @param subcomponent the embedded component to be checked
     * @return true if this model claims this component, false otherwise
     */
    public boolean isSubcomponentOwned(CmtSubcomponent subcomponent) {
        Object object = subcomponent.getLiveInstance();
        return (object instanceof PageProducer ||
            object instanceof IxComponent);
    }

    public DefaultTreeModel getGraph() { return _tree; }

    public CmtComponent getComponent() { return _source; }

    public CmtModelNode[] getChildren(CmtModelNode parent) {
        CmtModelNode[] children = new CmtModelNode[parent.getChildCount()];
        for (int index = 0; index < parent.getChildCount(); index++) {
            children[index] = (CmtModelNode)parent.getChildAt(index);
        }
        return children;
    }

    public CmtModelNode getRoot() { return (CmtModelNode)_tree.getRoot(); }

    public CmtModelNode getParent(CmtModelNode child) {
        return (CmtModelNode)child.getParent();
    }

    /**
     * Add a new node to this subtree.
     *
     * @param parent          the node to add the new node to
     * @param componentClassName the new class to instantiate
     * @param aheadOf          the node to add the new node before
     * @param subcomponent     a previously constructed wrapper for the
     *                          new node, or null if not yet handled
     * @return the new node
     */
    public CmtModelNode add(CmtModelNode parent, String componentClassName,
        CmtModelNode aheadOf, CmtSubcomponent subcomponent) {
        CmtSubcomponent subcomp = _source.addSubcomponent(
            componentClassName, null, CmtSubcomponent.CLASS_SCOPE);
        InternetBeansModelNode node =
            new InternetBeansModelNode(_designer, this, subcomp);
        _tree.insertNodeInto(node, (DefaultMutableTreeNode)parent,
            (aheadOf == null ? 0 : parent.getIndex(aheadOf)));
        _source.commit(false);
        return node;
    }

    /**
     * Move a node to a new location.
     *
     * @param node          the node to relocate
     * @param newParent     the node to add the new node to
     * @param aheadOf       the node to add the new node before
     * @return the moved node
     */

```

```

    */
    public CmtModelNode move(CmtModelNode node, CmtModelNode newParent,
        CmtModelNode aheadOf) {
        _tree.insertNodeInto((DefaultMutableTreeNode)node,
            (DefaultMutableTreeNode)newParent,
            (aheadOf == null ? -1 : newParent.getIndex(aheadOf)));
        return node;
    }

    /**
     * Delete a node from the tree.
     *
     * @param node the node to delete
     */
    public void remove(CmtModelNode node) {
        _source.removeSubcomponent(node.getSubcomponent());
        _tree.removeNodeFromParent((DefaultMutableTreeNode)node);
        _source.commit(false);
    }

    public String getName() { return _name; }

    public void close(){}
}

```

JBuilder uses the `getGraph` method to retrieve the tree model from this class so that it can be integrated into the Component Tree.

The name of the model appears on its root node and is retrieved from the `getName` method. Recall that this must be the same name as returned by the `getModelName` method of the class that implements the `Designer` interface.



WARNING

If the two names are different, then no error appears but the designer UI cannot be activated through the designer implementation.

The `isSubcomponentOwned` method is quite important as this lets you claim particular components embedded in the main class being constructed. If you return `true` from here, the subcomponent does not appear under the default tree (labeled `Other`) in the Structure Pane. If you return `false`, the subcomponent may appear twice: once within your sub-tree and once under `Other`.

Most of the rest of the methods simply manipulate the embedded tree model, allowing for the fact that the nodes in the tree are descended from `CmtModelNode`. When a new component is added to the tree it must also be added to the corresponding source file, hence a reference to that source file is passed into the model's constructor and retained for this use. A new instance of a component node is created and added to the sub-tree. Finally, the changes to the source file are saved through the following call:

```
source.commit(false);
```

Similarly, when a component is removed from the Component Tree, you must also delete it from the source file.

Individual nodes are represented by the `InternetBeansModelNode` class, which extends `javax.swing.tree.DefaultMutableTreeNode` and implements `CmtModelNode` (see Listing 21-3). The former lets it participate as part of the tree structure built up in an `InternetBeansModel` object, while the latter lets it interact with the components from the current source file.

Listing 21–3. The *InternetBeansModelNode* class.

```

package wood.keith.opentools.ibxdesigner;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.Enumeration;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

import com.borland.internetbeans.IxComponent;
import com.borland.internetbeans.PageProducer;
import com.borland.jbuilder.cmt.CmtModel;
import com.borland.jbuilder.cmt.CmtModelNode;
import com.borland.jbuilder.cmt.CmtSubcomponent;

/**
 * The tree node that encapsulates the InternetBeans component.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 10 September 2001
 */
public class InternetBeansModelNode extends DefaultMutableTreeNode
    implements CmtModelNode, PropertyChangeListener {

    private InternetBeansDesigner _designer;
    private CmtModel _model;
    private CmtSubcomponent _subcomponent;
    private String _name;

    /**
     * Initialise the node for this subcomponent.
     *
     * @param designer the designer instance that created this node
     * @param model the model that manages this subtree
     * @param subcomponent the component encapsulated by this node
     * @param name the name of the node
     */
    public InternetBeansModelNode(InternetBeansDesigner designer,
        CmtModel model, CmtSubcomponent subcomponent, String name) {
        _designer = designer;
        _model = model;
        _subcomponent = subcomponent;
        _name = name;
        if (subcomponent != null) {
            subcomponent.addPropertyChangeListener(this);
        }
    }

    /**
     * Initialise the node for this subcomponent.
     *
     * @param designer the designer instance that created this node
     * @param model the model that manages this subtree
     * @param subcomponent the component encapsulated by this node
     */
    public InternetBeansModelNode(InternetBeansDesigner designer,
        CmtModel model, CmtSubcomponent subcomponent) {
        this(designer, model, subcomponent,
            (subcomponent == null ? "null" : subcomponent.getName()));
    }

    public String getName() { return _name; }

    public void setName(String value) { _name = value; }

    public CmtModel getCmtModel() { return _model; }

    public CmtSubcomponent getSubcomponent() { return _subcomponent; }

```

```

public String getTag() { return null; }

public boolean isDesignable() { return true; }

/**
 * Update the node, including its position, based on the component's
 * property values.
 *
 * @param evt the triggering event
 */
public void propertyChange(PropertyChangeEvent evt) {
    if (evt.getPropertyName().equals("<Aname>")) {
        // Change the displayed name
        setName((String)evt.getNewValue());
        getCmtModel().getGraph().nodeChanged(this);
    }
    else if (evt.getPropertyName().equals("pageProducer")) {
        // Reposition the component under its page producer
        DefaultTreeModel tree = getCmtModel().getGraph();
        tree.removeNodeFromParent(this);
        PageProducer producer =
            ((IxComponent)getSubcomponent().getLiveInstance()).
            getPageProducer();
        InternetBeansModelNode parent =
            (InternetBeansModelNode)getCmtModel().getRoot();
        Enumeration children = parent.children();
        while (children.hasMoreElements()) {
            InternetBeansModelNode child =
                (InternetBeansModelNode)children.nextElement();
            if (child.getSubcomponent().getLiveInstance() == producer) {
                parent = child;
                break;
            }
        }
        tree.insertNodeInto(this, parent, 0);
        // And update the designer viewer
        _designer.setPageProducer(producer,
            (producer == null ? getName() : parent.getName()));
        _designer.setSelection(getSubcomponent().getLiveInstance());
    }
    else if (evt.getPropertyName().equals("controlName") ||
        evt.getPropertyName().equals("elementId")) {
        // Highlight selected component
        _designer.setSelection(getSubcomponent().getLiveInstance());
    }
}

public String toString() { return getName(); }
}

```

The `isDesignable` method returns `true` to indicate that there is a designer UI attached to this sub-tree that is invoked through the `activate` method of the designer. Meanwhile the `getName` method provides the name to display for this node within the tree.

To respond to changes in the property values of the underlying component, each model node also implements the `java.beans.PropertyChangeListener` interface and registers itself with the CMT subcomponent. As property values are altered, the `propertyChange` method is called. Within this method, you look for particular properties that affect the appearance of the component within the Component Tree and update it accordingly.

The name of the component is the caption for the node within the tree. Since this is a pseudo-property within JBuilder, it appears with the name "<Aname>". You must also notify the tree that one of its nodes has changed so that it can repaint itself correctly. Changes to the `pageProducer` property affect a

component's parent within the tree. Hence the node is removed from its current position and the tree is searched for the new producer. If found, the node is re-inserted beneath it, otherwise the node is re-inserted at the top level, directly beneath the `InternetBeans` node. Finally, if the `controlName` or `elementId` property is modified, the associated viewer is informed (via the designer) so that it can highlight the new position within the HTML file.

InternetBeansViewer Example

The last piece of the designer puzzle is the component that provides the user interface for the designer. Instances of the `InternetBeansViewer` class (as shown in Listing 21-4) supply that capability for this example and are created by the `activate` or `open` methods of the `InternetBeansDesigner`.

Listing 21-4. The `InternetBeansViewer` class.

```
package wood.keith.opentools.ibxdesigner;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Point;
import java.io.IOException;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

//import com.borland.jbuilder.editor.HTMLEditorKit; // JB9-
import com.borland.internetbeans.AbstractIxControl;
import com.borland.internetbeans.IxComponent;
import com.borland.internetbeans.PageProducer;
import com.borland.primetime.editor.EditorManager;
import com.borland.primetime.editor.EditorPane;
import com.borland.primetime.editor.TextEditorKit;
import com.borland.primetime.ide.Browser;
import com.borland.primetime.node.html.HtmlEditorKit; // JB10+
import com.borland.primetime.vfs.Buffer;
import com.borland.primetime.vfs.BufferListener;
import com.borland.primetime.vfs.BufferUpdater;
import com.borland.primetime.vfs.ReadOnlyException;
import com.borland.primetime.vfs.Url;

/**
 * The UI for the InternetBeans designer.
 * It displays the contents of the HTML file associated with a
 * PageProducer and highlights the position of IxComponents
 * within that file.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 10 September 2001
 */
public class InternetBeansViewer extends JPanel
    implements BufferListener, BufferUpdater, DocumentListener {

    private BorderLayout _borderLayout = new BorderLayout();
    private JLabel _nameLabel = new JLabel();
    private JScrollPane _viewScroll = new JScrollPane();
    private EditorPane _viewPane = new EditorPane();
    /* Specialised editor kit for display */
    // private TextEditorKit _htmlEditorKit = // JB9-
    // EditorManager.getEditorKit(HTMLEditorKit.class);
    private TextEditorKit _htmlEditorKit = // JB10+
    EditorManager.getEditorKit(HtmlEditorKit.class);
```

```

private PageProducer _producer = null;
private Buffer _buffer = null;

public InternetBeansViewer() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**
 * JBuilder UI initialisation.
 *
 * @throws Exception
 */
private void jbInit() throws Exception {
    this.setLayout(_borderLayout);
    _nameLabel.setText("Component");
    _viewPane.setFont(EditorManager.getFont());
    _viewPane.setEditorKit(_htmlEditorKit);
    _viewPane.setText("Internet Beans Viewer");
    _viewPane.getDocument().addDocumentListener(this);
    _viewScroll.setPreferredSize(new Dimension(200, 200));
    this.add(_nameLabel, BorderLayout.NORTH);
    this.add(_viewScroll, BorderLayout.CENTER);
    _viewScroll.getViewPort().add(_viewPane, null);
}

/**
 * Display the HTML content of the PageProducer,
 * or an appropriate message if it is not available.
 * Also tie into the VFS for changes to the HTML file buffer.
 *
 * @param producer the PageProducer to get the content from
 * @param name      the component name to display at the top
 */
public void setPageProducer(PageProducer producer, String name) {
    if (_buffer != null) {
        _buffer.removeBufferListener(this);
        _buffer = null;
    }
    _viewPane.getDocument().removeDocumentListener(this);
    _producer = producer;
    _nameLabel.setText("    " + name);
    _viewPane.setEditable(false);
    if (producer == null) {
        _viewPane.setText(
            "Please set the page producer for this component");
        _viewPane.getDocument().addDocumentListener(this);
        return;
    }
    try {
        String filePath = null;
        try {
            filePath = producer.getHtmlPath();
        }
        catch (NullPointerException npe) {
            filePath = null;
        }
        if (filePath == null || filePath.length() == 0) {
            _viewPane.setText(
                "Please set the HTML file name for this component");
        }
        else {
            _buffer = Browser.getActiveBrowser().getActiveProject().
                getNode(new Url("file", filePath)).getBuffer();
            _buffer.addBufferListener(this);
            _viewPane.setText(new String(_buffer.getContent()));
        }
    }
}

```

```

        _viewPane.setEditable(true);
    }
}
catch (IOException ioe) {
    ioe.printStackTrace();
    _viewPane.setText("Error during HTML load (" +
        ioe.getClass().getName() + ")\n" + ioe.getMessage());
}
_viewPane.getDocument().addDocumentListener(this);
}

/**
 * Mark the position of a component on the page.
 *
 * @param component the component to be highlighted
 */
public void setSelection(IxComponent component) {
    Point loc = new Point(0, 0);
    if (component != null && component.getPageProducer() == _producer) {
        boolean useName = (component instanceof AbstractIxControl);
        String id = (useName ?
            ((AbstractIxControl) component).getControlName() :
            component.getElementId());
        String text = _viewPane.getText();
        String attrName = (useName ? "name" : "id");
        if (!findText(text, attrName + "\\\"" + id + "\"", loc)) {
            if (!findText(text, attrName + "= \"" + id + "\"", loc)) {
                if (!findText(text, attrName + "'\" + id + "'", loc)) {
                    if (!findText(text, attrName + "\" = '" + id + "'", loc)) {
                        findText(text, attrName + "=" + id, loc);
                    }
                }
            }
        }
    }
    _viewPane.setSelectionStart((int) loc.getX());
    _viewPane.setSelectionEnd((int) loc.getY());
    _viewPane.repaint();
}

/**
 * Find the supplied text and return its position.
 *
 * @param text the text being searched
 * @param subtext the text to search for
 * @param point the location of the subtext (updated for output)
 * @return true if the subtext was found, false otherwise
 */
private boolean findText(String text, String subtext, Point point) {
    int start = text.indexOf(subtext);
    if (start == -1) {
        point.setLocation(0, 0);
        return false;
    }
    else {
        point.setLocation(start, start + subtext.length());
        return true;
    }
}

public PageProducer getPageProducer() { return _producer; }

/**
 * If the buffer is changed by someone other than us,
 * then reload the text.
 *
 * @param buffer the buffer being changed
 * @param updater the lazy updater for this buffer
 */
public void bufferChanged(Buffer buffer, BufferUpdater updater) {

```

```

    if (updater != this) {
        _viewPane.getDocument().removeDocumentListener(this);
        _viewPane.setText(new String(buffer.getContent()));
        _viewPane.getDocument().addDocumentListener(this);
    }
}

public void bufferLoaded(Buffer buffer) {
    // Do nothing
}

public void bufferSaving(Buffer buffer) {
    // Do nothing
}

public void bufferStateChanged(Buffer buffer, int oldState,
    int newState) {
    // Do nothing
}

/**
 * Supply any changes to the file buffer as needed.
 *
 * @param buffer the buffer being updated
 * @return the new content
 */
public byte[] getBufferContent(Buffer buffer) {
    return _viewPane.getText().getBytes();
}

public void insertUpdate(DocumentEvent e) {
    doModified();
}

public void removeUpdate(DocumentEvent e) {
    doModified();
}

public void changedUpdate(DocumentEvent e) {
    doModified();
}

/**
 * Notify interested parties of changes to the file buffer.
 */
private void doModified() {
    if (_buffer != null) {
        try {
            _buffer.setContent(this);
        }
        catch (ReadOnlyException ex) {
            // Ignore
        }
    }
}
}

```

The viewer extends `JPanel` and adds to it a label across its top, with an editor pane filling the remainder. The label displays the name of the currently selected page producer component, while the editor shows its HTML file. To have the HTML displayed with syntax highlighting, you must assign the correct editor kit to the editor. JBuilder provides an appropriate kit through its `com.borland.pruntime.node.html.HtmlEditorKit` class.



VERSION

The `HtmlEditorKit` class was in the `com.borland.jbuilder.editor` package before JBuilder 10. Note that it was also named slightly differently.

Since you are displaying the contents of the HTML file for the page producer, you should interact with it through JBuilder's Virtual File System (VFS). Then, if another copy of that file is opened, the two versions will remain synchronized via the internal buffer, and will be stored as part of the project if everything is saved.

When a new page producer is activated, the `setPageProducer` method of the viewer is called. If an existing buffer is being used, the viewer is removed from its notification list. An attempt is then made to open the HTML file, with an appropriate message being added to the editor if no file name is specified. The VFS is requested to open the file as a JBuilder node and return the buffer for it. To this buffer is added the viewer as a listener for changes, while the editor is loaded with the buffer's current contents.

During construction, the viewer object is registered as a document listener for changes to the editor pane, and the editor is made editable. Responding to these change events lets you notify other users of the HTML file buffer of updates. Each of the document listener's methods calls the `doModified` method that checks for an associated buffer, and tells it that new content is available.

Rather than copy the new text for every keystroke, the viewer class also implements the `com.borland.primetime.vfs.BufferUpdater` interface (see Chapter 5), allowing the updating to be deferred until it is actually required, such as switching to another view of the file. The single method of this interface, `getBufferContent`, returns the new content as a byte array on demand. Other interested parties are informed of changes through the `setContent` method of the buffer. But rather than supplying the data directly, the instance of the updater is sent instead. The other classes can then call back when they need to update their own view.

Similarly, the `com.borland.primetime.vfs.BufferListener` interface lets this class react to changes made to the buffer by other objects. In this case, you are only interested in changes and so use the `bufferChanged` method to load the new text via the buffer passed in. You should first check that the updater, also passed as a parameter, is not this class to avoid a cycle of updates.

When a new InternetBeans component is selected from the Component Tree, the current page producer's HTML is searched for the tag that corresponds to that component. The viewer is notified through its `setSelection` method, which checks that the component belongs to the current page producer and then determines which property value to use for the search. Descendents of the `com.borland.internetbeans.AbstractIxControl` class link to the tag through their `controlName` property, while other InternetBeans components link via their `elementId` property.

Several searches then follow based on differing spacing separating the HTML attribute and its value, and the different use of quotes for the attributes. The `findText` method looks for the given text and updates a `Point` parameter wherein the X value indicates its starting position and the Y value is the ending position, as well as returning a boolean value indicating success. If the text is not found the point is returned as (0, 0). Using the values from the point you set the selection within the editor and hence highlight the section corresponding to the original component.

If no page producer is assigned to a component yet, the initial call to the `setPageProducer` method is passed `null`, resulting in the name of that component being shown, along with a message that the page producer needs to be set.

Now all the parts of the new designer are ready. Compile the classes and place them into a JAR file. Add the OpenTools manifest entry:

```
OpenTools-Designer: wood.keith.opentools.ibxdesigner.InternetBeansDesigner
```

Then copy the completed JAR file to the `{JBuilder}/lib/ext` directory and restart JBuilder. Now, whenever you open the Design tab for a Java source file, you see the InternetBeans category within the Component Tree in the Structure Pane. If you add InternetBeans components to your class, they appear under this node. You should set the HTML file name for your page producer, then select the page producer for each other component and enter its tag name. The results are shown in Figures 21-2 and 21-3.

By activating the new designer (through double-clicking on one of the components in the tree, or by right-clicking and selecting **Activate Designer** from the popup menu) you see the designer's UI and can quickly locate controls within their producer's page. Any updates made to the HTML are reflected in other views of that file automatically.

Summary

To enhance the design process within JBuilder, you can add your own designers to those already supplied. The designers manifest themselves when the Design tab is selected for a Java source file and affect the IDE in two ways: they create hierarchies of components within the Structure Pane, and they provide graphical design surfaces for the components that they manage. Standard designers look after visual components, menu construction, and database access.

The example designer discussed here deals with the InternetBeans components, building a tree that shows which controls belong to which page producers. As the latter are activated, their HTML template is displayed, while controls bring up their page producer's content before highlighting the section that they affect. By linking into the VFS, your designer can immediately update and respond to changes in other copies of the HTML template that may be open.

One of the sample projects that come with JBuilder also illustrates the UI designer interfaces.

```
{JBuilder}/samples/OpenToolsAPI/designer/Designer.jpx
```

A basic example that has three different designers included. The first shows how to annotate the Component Tree by adding all the subcomponents under a node called `this`. The second version creates a one-level sub-tree named `DesignerStep2`, but only if the main class is a descendent of a known class. The third version only adds those components derived from a known type, placing them under a node called `DesignerStep3`.

None of the three versions prevents the subcomponents from appearing under the node that they are normally associated with. All three also display debugging-type messages to a popup window for the unused methods in the `Designer` interface to help you follow the flow of the calls.



Chapter 22

Layout Assistants

Another way that you can enhance the design environment is to provide support for working with layout managers in a GUI project. Layout managers enable you to design a user interface in a manner that is portable across platforms and screen capabilities. A manager encapsulates a policy about where components are placed and how they are repositioned and resized as their container is resized. By nesting layout managers you can achieve just about any appearance and behavior required.

Java comes with several layout managers already included, from the simple `FlowLayout`, to `BorderLayouts`, and the more complex `GridBagLayout`. Each has its own uses and idiosyncrasies. And if the existing ones are not enough, you can always develop your own manager to provide your specific requirements by implementing the `LayoutManager` or `LayoutManager2` interfaces (both in the `java.awt` package).

However, these layout managers do not provide enough information or feedback to enable them to be used effectively in a graphical design environment. Borland's solution is to define the `LayoutAssistant` interface. Once implemented and registered against the layout manager class, it acts as an intermediary between `JBuilder` and the manager, supplying visual feedback for common design operations.

Note that a layout does not become available for selection within the drop-down list for the `layout` property until a layout assistant is registered for it. This is because `JBuilder` has no way of interacting with it until that assistant exists.



UNDOCUMENTED

Unfortunately, the entire layout assistant portion of the `JBuilder OpenTools API` is undocumented. There are a couple of mentions of its existence in the `OpenTools` documentation, but only to say that it can be done. And there is one, very basic, example.

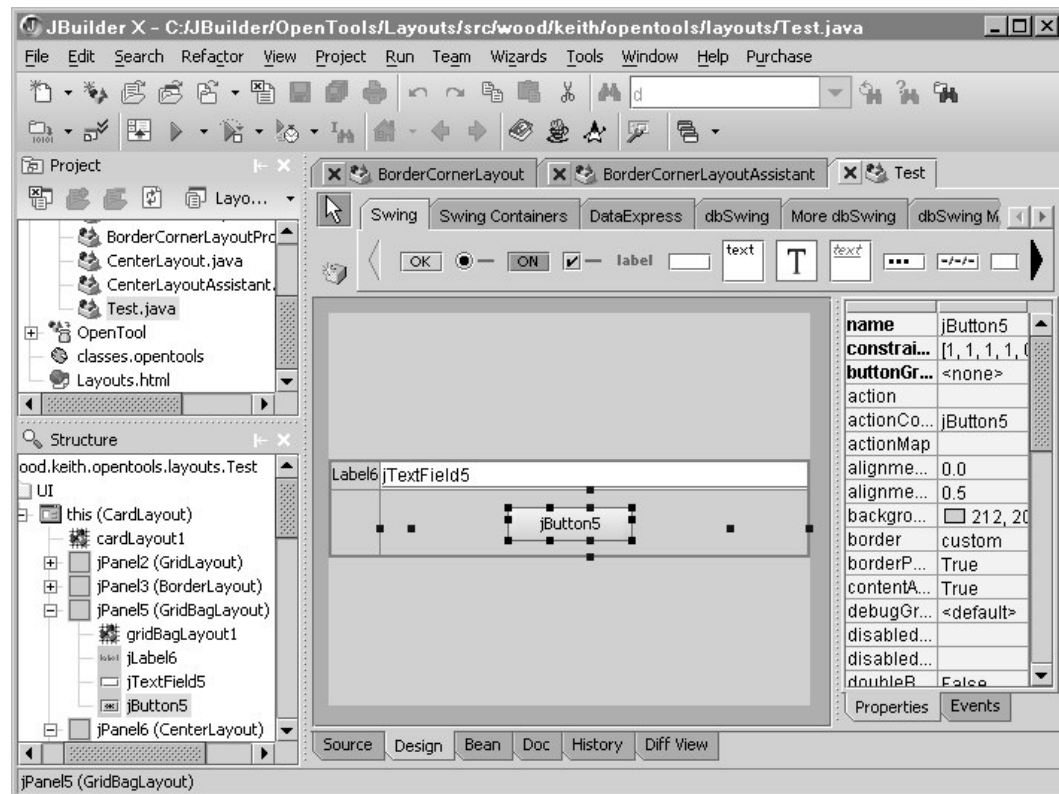
You can see the abilities supplied by the assistant by opening the `Design` tab for a frame. Typically, when the mouse hovers over the container itself, the name of that container and its layout manager are shown in the status bar. For components within the layout, the status bar usually displays the component name and whatever additional information the assistant deems appropriate, such as the constraint name or coordinates.

When you add a new component to the frame, the status bar again provides

details about where it would be placed within the layout were it to be dropped at that point. If you drag an existing component, you typically receive feedback both in the status bar, and within the frame itself via a selection box. For layouts where it makes sense, the assistant can also provide feedback when resizing a component.

Figure 22–1 shows the assistant for the `GridBagLayout` in use – with the button component selected. Note that the assistant outlines the existing cells within the grid, as well as providing handles for resizing the padding and insets of a cell. The status panel displays the container’s name and layout manager.

Figure 22–1. Layout assistant for the `GridBagLayout` in action.



LayoutAssistant Interface

The `com.borland.jbuilder.designer.ui.LayoutAssistant` interface defines numerous methods for interacting with the associated layout manager. These let you add functionality at appropriate times during the design process for your layout. As with the other designers, the layout assistant interacts with the components in the source file through the Component Modeling Tool (CMT), which was described in Chapter 20.

To make your new layout assistant available, you must register it with the `com.borland.jbuilder.designer.ui.UIDesigner` class, which should be performed within the “Designer” OpenTools category. The first parameter is the class of the layout assistant, followed by the full name of the actual layout that it deals with, and a flag that is true to add the layout name to the drop-down list of supported layouts in the Inspector or false to omit it.

```
UIDesigner.registerAssistant(BorderCornerLayoutAssistant.class,
    "wood.keith.opentools.layouts.BorderCornerLayout", true);
```

NOTE

The `UIDesigner` class is one of JBuilder's standard designers (see Chapter 21). Apart from its Designer abilities and the `registerAssistant` method mentioned above, it has two other static methods relating to layouts. Otherwise, this class is not covered in this book.

`getAssistedLayouts` returns an array of the names of registered layouts (those registered with `true` above). `getAssistantClass` takes one of these names and returns the class of the assistant for that layout.

The ultimate purpose of your layout assistant is to update the constraints for the selected components within the designer. Usually this is done with code like the following, where `node` is a `com.borland.jbuilder.designer.ui.ModelNode` object (which implements `CmtModelNode`):

```
node.getConstraints().setValueSource(constraint);
```

A new instance of the layout assistant is created for each layout where it is used. The methods to be implemented are listed below.

```
public void cleanupRemovedComponent(ModelNode contNode);
```

When a child component is deleted from the container, this method gives you a chance to recompute the constraints for the remaining children, or to perform whatever other processing is appropriate.

`contNode` is the CMT wrapper for the container that is using this layout manager.

```
public void constraintEditorSelectionChanging(ModelNode
node);
```

Notification appears via this method that the user has changed the selection of nodes within the Structure Pane while the constraint editor is open. Normally this method just frees up any listeners it had registered and tidies up the previous editor. The `editConstraints` method is then called to let you initialize the editor for the new selection.

`node` is the CMT wrapper for the previously selected component.

```
public void editConstraints(TreePath[] treePaths, ModelNode
node);
```

This method is called when the user changes the nodes selected in the Structure Pane while the constraints editor is open. First you receive a `constraintEditorSelectionChanging` call to allow you to tidy up the previous editor. In this method you then update the editor with the constraints for the new selection.

`treePaths` is an array of path details for the selected components. Find each component with the following:

```
CmtModelNode cmtmodelnode =
    (CmtModelNode)treePaths[i].getLastPathComponent();
```

`node` is the CMT wrapper for the container.

```
public String getConstraintsType();
```

Return the full path and class name of the type of the constraints used by this layout manager, for example `"java.lang.String"`. If no constraints apply, then return `null`. JBuilder calls this method when creating the constraint values.

```
public PropertyEditor getPropertyEditor();
```

Return an instance of a property editor for use with the constraints for this layout manager, or `null` if no property editor is available. A single instance can be created and reused via this function. It is a standard JavaBeans property editor (`java.beans.PropertyEditor`).

```
public void layoutChanged(ModelNode contNode);
```

Use this method to respond to changes in the layout property of the container from the manager handled by this assistant to another one. You can then release whatever resources the assistant is holding. These resources were probably acquired in the `prepareChangeLayout` method call.

`contNode` is the CMT wrapper around the container that is switching layouts.

```
public void prepareActionGroup(ActionGroup actionGroup);
```

Add actions to the popup menu that displays when right-clicking over a container using this layout. The actions appear at the bottom of the menu in their own group.

`actionGroup` is the group to which your custom actions are added.

```
public void prepareAddComponent(ModelNode addNode, Point
point, Dimension dimension);
```

Add the specified component to its container using this layout with the appropriate layout constraint based on the point and preferred dimensions provided. This method is called when a new component is dragged from the Component Palette and dropped onto the container. Feedback while positioning the new component comes from the `prepareAddStatus` method.

Set the constraints for the newly added node within this method with the following:

```
addNode.getConstraints().setValueSource(constraint);
```

`addNode` is the CMT wrapper around the new component being dropped. It is already connected to the model hierarchy, so you can find its container's CMT node with the following:

```
ModelNode contNode = (ModelNode)addNode.getParent();
```

`point` is the location where the new component was dropped.

`dimension` is `null` if the new component was just dropped, or it is the selected size if a dragged size was drawn.

```
public String prepareAddStatus(ModelNode curNode, ModelNode
contNode, Point point, Dimension dimension);
```

Provide feedback when a new component is being added to the container, but before it is actually placed. Usually this returns a description of the location within the layout that would be used if the component were dropped here, and is shown in the status bar. The `prepareAddComponent` method does the final placement.

`curNode` is the CMT wrapper for the component being added.

`contNode` is the CMT wrapper for the container that uses this layout manager and will hold the new component.

`point` is the current location of the mouse within the container.

`dimension` is the size of the component to be added. It has a value if a size is being dragged out, and is `null` otherwise.

```
public void prepareChangeLayout(ModelNode contNode);
```

When the layout manager for a container is changed to the one handled by this assistant, this method informs you of that fact. It is called after a new instance of the layout assistant is created, and lets you perform whatever initialization your assistant requires.

This is an appropriate place to set any import statements needed by your layout. You do this with the following code, which ties into the CMT:

```
contNode.getModel().getComponentSource().getSourceFile().
    addImport("wood.keith.opentools.layouts.*");
```

The `layoutChanged` method lets you tidy up when the assistant is no longer needed.

`contNode` is the CMT wrapper for the container that has had its layout altered.



WARNING

The `prepareChangeLayout` method is only called when the layout for a container is changed to this layout. It is not called if the container's layout is already assigned to this layout when the designer opens.

```
public void prepareCloneComponent(ModelNode node, Point
point);
```

Copying a component and adding it to the container invokes this method.

`node` is the CMT wrapper for the component being added.

`point` is the location within the container at which to add the new component.



NOTE

I have not found the circumstances under which the `prepareCloneComponent` and `prepareCloneStatus` methods are called. The descriptions here are based on the method names and how the other methods function.

```
public String prepareCloneStatus(ModelNode curNode,
ModelNode contNode, Point point);
```

To show where a cloned component will be placed, you respond to this method. The resulting string is displayed in the status bar.

`curNode` is the CMT wrapper for the component being added.

`contNode` is the CMT wrapper for the container that uses this layout manager and will hold the new component.

`point` is the current location of the mouse within the container.

```
public String prepareMouseMoveStatus(ModelNode curNode,
ModelNode contNode, Point point);
```

As the mouse moves around the layout container, return a string to display in the status bar that provides feedback about the component currently under the mouse. Often this is the name of the component and its constraints within this layout, or the name and layout class of the container itself if no child is indicated.

`curNode` is the CMT wrapper for the component currently under the mouse. This parameter is `null` if there is no component within the container at the current mouse location.

`contNode` is the CMT wrapper for the container that uses this layout manager and holds the current component.

`point` is the current location of the mouse within the container.

```
public void prepareMoveComponent(ModelNode node, Point
point1, Point point2);
```

Once a component has been dragged to a new position and dropped, this method lets you update its properties based on the drop-point.

`node` is the CMT wrapper around the selected component.

`point1` is the position of the mouse within the container when dropping the moved component.

`point2` is the position of the mouse within the component when starting the drag operation.

```
public String prepareMoveStatus(ModelNode curNode,
ModelNode contNode, Point location, SelectBoxes
selectBoxes, Point offset);
```

Provide feedback when dragging a component to a new position. The returned value is shown in the status bar.

`curNode` is the CMT wrapper for the component currently under the mouse.

`contNode` is the CMT wrapper for the container that uses this layout manager and holds the current component.

`point` is the current location of the mouse within the container.

`selectBoxes` is a reference for drawing a feedback rectangle. See the next section for more details.

`offset` is the location of the mouse within the component when the drag started.

```
public void prepareResizeComponent(ModelNode node,
SelectNib selectNib);
```

When a resizing operation has completed, this method lets you update the component, based on the new settings from its resizing handle.

`node` is the CMT wrapper around the selected component.

`selectNib` encapsulates the resizing handle that is being dragged. See the following section for more details on this class.

```
public String prepareResizeStatus(ModelNode node, Point
point, Dimension dimension);
```

Prepare feedback for display in the status bar when resizing a component.

`node` is the CMT wrapper around the selected component.

`point` is the current mouse position within the designer.

`dimension` is the new size of the selected component.

```
public void prepareSelectComponent(ModelNode node,
DesignView designView);
```

Whenever a component is selected within the designer, this method is triggered. It also fires when the browser is reactivated after switching to another application. It allows you to prepare for working with the chosen component.

`node` is the CMT wrapper around the selected component. It is never the container itself, and the method does not fire if no child component is chosen.

`designView` is a reference to the designer itself. Through this object you gain access to any resizing handles that apply within your layout. Use the `assureNibs` method to generate the desired number of handles. Then set up each one as necessary. See the section on the `SelectNib` class below.

```
public String resizeAction(Point point, DesignView
    designView, SelectNib selectNib);
```

As a component is resized, this method lets you provide appropriate feedback. Return a string value to display in the status bar, presumably to show the new dimensions of the component.

`point` is the current location of the mouse within the designer.

`designView` is a reference to the designer itself.

`selectNib` encapsulates the resizing handle that is being dragged. See the corresponding section for more details on this class.

DesignView Class

The `com.borland.jbuilder.designer.ui.DesignView` class represents the design surface within the designer. It extends `JRootPane`, listens for changes within the Component Tree, and adds the following methods, several of which apply directly to layout assistants:

```
public static void adjustPositionForNib(Point nibPoint,
    Dimension nibSize, Point point, int nibType);
```

When a nib is dragged, this method calculates its effect on the original sizing rectangle, adjusting the data passed to it.

`nibPoint` is the location of the nib rectangle, usually taken from `selectNib.getRectangleLocation`. It is updated by this method.

`nibSize` is the size of the nib rectangle, usually taken from `selectNib.getRectangleDimension`. It is updated by this method.

`point` is the new location of the mouse.

`nibType` is the type of nib being dragged, as this affects how the rectangle changes. Often it is `selectNib.type`.

```
public synchronized SelectNib[] assureNibs(int count);
```

Ensure that a certain number of resizing nib objects is available for use. See the section below for more on nibs.

`count` is the number of nibs required.

```
public static Point componentAbsLocation(Component
    component);
```

Find the location of a component relative to the designer as a whole.

`component` is the component to locate.

```
public DesignerViewer getDesignerViewer();
```

Retrieve the viewer hosting this design surface.

```
public Model getModel();
```

Obtain a reference to the model managed by this design surface.

```
public SelectBoxes getTempComponent();
```

Get the temporary boxes that you can use for feedback when the user interacts with the layout assistant. See below for more detail.

```
public static boolean isConstraintEditorShowing();
    Determine whether the constraints dialog is visible.
public void setModel(Model model);
    Change the model connected with this design surface.
    model is the new model to use.
```

One field of possible interest within the class is:

```
public static ButtonDialog constraintDialog;
    This is a reference to the dialog for the constraint editor. It may be null.
```

SelectBoxes Interface and SelectNib Class

To enhance the visual feedback provided by the layout assistant and to provide additional functionality, you can use the `com.borland.jbuilder.designer.ui.opt.SelectBoxes` interface and the `com.borland.jbuilder.designer.ui.SelectNib` class.

The `SelectBoxes` interface allows you to show or hide one or more selection rectangles in the designer. Typically it is used when moving a component to a new location within the layout, or when resizing a component. An instance of this interface is passed as a parameter to the `prepareMoveStatus` method of the `LayoutAssistant` interface. You can also retrieve a reference to an instance through the `DesignView` object passed to the `prepareSelectComponent` and `resizeAction` methods using the following:

```
SelectBoxes selectBoxes = designView.getTempComponent();
```

Listed below are the methods of the `SelectBoxes` interface.

```
public boolean hide(int index);
    Hide a particular selection box via this method.
    index is the index of the box to hide.
public void hideAll();
    Or make all the selection boxes invisible with this method.
public boolean isVisible();
    Determine whether or not any of the selection boxes are currently being
    shown with this method.
public void setContainer(Container container);
    Establish the container that these selection boxes apply to.
    container is the component that uses the layout manager accessible
    through the current assistant.
public void show(int index, Point point, Dimension
    dimension, int width);
    Create and display a particular selection box with this method.
    index is the index of the box being created (usually starting at zero).
    point is the origin of the box in the coordinates of the designer.
    dimension is the size of the box.
    width is the width of the line for the box in pixels.
```

Resizing handles for components are managed through the `SelectNib` class. Instances of this class are passed to the assistant's `prepareResizeComponent` and `resizeAction` methods. Its methods are as follows:

```
public LayoutAssistant getLayoutAssistant();
```

Retrieve the reference to the assistant associated with this nib.

```
public Rectangle getNibBounds();
```

Obtain the position and size of the rectangle drawn out by the nib within the coordinate system of its container.

```
public Dimension getRectangleDimension();
```

Retrieve the current dimensions of the rectangle drawn out by the resizing nib, from the origin of the component.

```
public Point getRectangleLocation();
```

Find the origin of the rectangle for the nib, based on the coordinates of the designer as a whole.

```
public boolean isSelectable();
```

Returns true if this nib is functional, or false otherwise.

```
public String resizeAction(Point point, DesignView  
designview);
```

For feedback during a resizing operation, this method returns a basic string for display in the status bar.

point is the current location of the mouse within the designer.

designView is a reference to the designer itself.

```
public void setLayoutAssistant(LayoutAssistant  
layoutassistant);
```

Associate the layout assistant with this nib.

layoutAssistant is the reference, typically set to *this* as the nib is set up within the assistant.

```
public void setRectangleDimension(Dimension dimension);
```

Set the size of the base rectangle for this nib. Usually this is the size of the attached component.

dimension is the size of the resizing rectangle.

```
public void setRectangleLocation(Point point);
```

The base rectangle for this nib starts at the point set through this method. Typically this is the origin of the component associated with the nib.

point is the location of the origin for the resizing rectangle.

```
public void setSelectable(boolean flag);
```

Establish whether or not the handle allows selection and dragging. If it is selectable, the cursor changes when over the handle, according to the type described below.

flag is true to make the handle functional, or false to disable it.

Use the methods inherited from `JComponent` to set the position (in the designer coordinate system) and dimensions of the nib: `setBounds`, `setLocation`, or `setSize`. The default size is a five pixel square.

There are several public fields within the `SelectNib` class that also affect its functionality:

```
public Point parentLocation;
```

Use this field to establish the position of the container within the designer.

```
public Object target;
```

Set this value to something appropriate for the type of handle being used, typically the currently selected component. Once the nib has been selected it

appears in calls to the `LayoutAssistant` methods, allowing you to retrieve the object affected by the handle at that time.










```
public int type;
```

The type of the nib determines the cursor used to indicate its functionality, as well as identifying how the sizing rectangle is adjusted when dragged. Table 22–1 shows the defined types – there do not appear to be any constants defined for these values.

```
public int use;
```

Set this field for whatever internal purpose you want to make use of it. For example, the `GridBagLayout` assistant uses this value to identify the different types of resizing handles it exposes.

Table 22–1. SelectNib types

Type	Cursor	Purpose
–1		Not visible
0		Top left resizing nib
1		Top right resizing nib
2		Bottom left resizing nib
3		Bottom right resizing nib
4		Top vertical resizing nib
5		Left horizontal resizing nib
6		Right horizontal resizing nib
7		Bottom vertical resizing nib
8		Four-way resizing nib

Resizing nibs are set up in the `prepareSelectComponent` method of the assistant. You request a number of nibs from the designer via the `designView` parameter. By default, four nibs are established in the corners of the component, although they are initially disabled. Then for each nib you set its properties and `JBuilder` displays it for you. Interactions with the nibs then flow through the assistant's `prepareResizeComponent` and `resizeAction` methods.

The following code snippet retrieves references to the basic four nibs, then sets the bottom-right one to be active and to operate on the current component.

```
SelectNib[] nibs = designView.assureNibs(4);
nibs[3].setBackground(Color.black);
nibs[3].setLayoutAssistant(this);
nibs[3].setSelectable(true);
nibs[3].target = node;
```

Within the `resizeAction` method you use the `SelectNib` object to retrieve the new position and dimensions of the component. For example, to draw an outline representing the new settings, you can use the following:

```
Point nibPoint = selectNib.getRectangleLocation();
Dimension dimension = selectNib.getRectangleDimension();
// Update rectangle size based on nib type and location
DesignView.adjustPositionForNib(nibPoint, dimension, point,
    selectNib.type);
// Draw the outline for the new size
designView.getTempComponent().show(0, nibPoint, dimension, 2);
```

By using the `SelectBoxes` interface and the `SelectNib` class, you can easily enhance the appearance and functionality of your layout assistant.

BasicLayoutAssistant Class

The `com.borland.jbuilder.designer.ui.BasicLayoutAssistant` class provides a basic implementation of the `LayoutAssistant` interface and can be used as the ancestor of your own customized layout assistant. This class maintains a z-ordering of the components in the container, based upon their x and y coordinates.

Unless otherwise described below, the methods required by the interface are implemented as empty or return `null`. Several additional methods are available for your subclass to use:

```
protected int calcBestZ(ModelNode node, ModelNode contNode,
    Point point, Dimension dimension, Rectangle rectangle);
```

Calculate a z-ordering value for a node and return it.

`node` is the node to calculate the z-ordering for.

`contNode` is the container for the node.

`point` is the top-left position of the node, or 0,0 if set to `null`.

`dimension` is the size of the node.

`rectangle` is updated to reflect the new ordering.

```
protected static boolean changeZ(ModelNode node, ModelNode
    contNode, int newZ);
```

Alter the z-ordering for a node, returning `true` if it was changed, or `false` if it was not.

`node` is the node to set the z-ordering for.

`contNode` is the container for the node.

`newZ` is the new z-ordering.

```
protected static boolean frontBack(boolean toBack,
    ModelNode contNode, ArrayList paths, DesignView
    designView);
```

Modify the ordering of the nodes, returning `true` if any were changed, or `false` if none were.

`toBack` is `true` to move the nodes to the back, or `false` to move them to the front.

`contNode` is the container for the nodes.

`paths` is the list of `TreePath` entries within the container to move.

`designView` is a reference to the designer itself.

```
protected static CmtSubcomponent
    getCustomizableSubcomponent(ArrayList paths);
```

Get the CMT subcomponent for a node that has a customizer attached, or `null` if there is none.

`paths` is the list of paths to find the subcomponent for. However, if there is not exactly one `TreePath` entry in the list it returns `null`.

```
protected static Object getKey(ModelNode node);
```

Returns an object to use as a key for ordering nodes. In fact it is a `Point` that corresponds to the midpoint of the component.

`node` is the node to find the key for.

```
protected static CmtSubcomponent
    getSerializableSubcomponent(ArrayList paths);
    Get the CMT subcomponent for a node that is serializable, or null if there is
    none.
    paths is the list of paths to find the subcomponent for. However, if there is
    not exactly one TreePath entry in the list it returns null.
```

```
protected static boolean lessThan(Object object1, Object
    object2);
    Compare two objects (which must be Points) and return true if the first one
    is before the second one (down the form and then across).
    object1 and object2 are the points to compare.
```

```
public void prepareActionGroup(ActionGroup actionGroup);
    Adds Move to First and Move to Last actions to the popup menu. You need to
    override this if you do not want these menu items to appear.
```

```
public void prepareAddComponent(ModelNode addNode, Point
    point, Dimension dimension);
    Calculates the new z-position for the added component, based on its point
    value.
```

```
public String prepareAddStatus(ModelNode curNode, ModelNode
    contNode, Point point, Dimension dimension);
    Returns the node's name and z-position for display in the status bar.
```

```
public void prepareCloneComponent(ModelNode node, Point
    point);
    Calculates the new z-position for the cloned component, based on its point
    value, for display in the status bar.
```

```
public abstract String prepareCloneStatus(ModelNode
    curNode, ModelNode contNode, Point point);
    Returns the node's name and z-position for display in the status bar.
```

```
public String prepareMouseMoveStatus(ModelNode curNode,
    ModelNode contNode, Point point);
    Returns the name of the current component under the mouse, or the container
    if no component, for display in the status bar.
```

```
public void prepareMoveComponent(ModelNode node, Point
    point1, Point point2);
    Calculates the new z-position for the moved component, based on its
    point1 value.
```

```
public String prepareMoveStatus(ModelNode curNode,
    ModelNode contNode, Point location, SelectBoxes
    selectBoxes, Point offset);
    Returns the node's name and new z-position for display in the status bar.
    Also draws an outline of the component at its new position.
```

```
public void prepareResizeComponent(ModelNode node,
    SelectNib selectNib);
    Calculates the new z-position for the resized component, based on its
    selectNib value.
```

```
public String prepareResizeStatus(ModelNode node, Point
    point, Dimension dimension);
    Returns the node's name and z-position for display in the status bar.
```

```

public void prepareSelectComponent(ModelNode node,
    DesignView designView);
    Draws resizing handles at the four corners of the component.

protected void removeBoundsSetting(ModelNode node);
    Set the bounds property for a node back to its default.
    node is the node to update.

protected static void serialize(CmtSubcomponent
    subcomponent);
    Serialize the given subcomponent.
    subcomponent is the CMT subcomponent to serialize.

protected static void sort(ArrayList objects, ArrayList
    keys);
    Sort a list of objects based on their keys.
    objects is the list of objects to be sorted.
    keys is a corresponding list of the object's keys to use in determining their
    order.

protected ArrayList sortedNodes(ModelNode node);
    Sort the children of a node and return the ordered list.
    node is the node whose children are sorted.

```

The class also defines several actions for operating on the layout and its components.

```

public static UpdateAction ACTION_Customizer;
    Invokes a customizer for the selected component if one is available. This
    action is added to the popup menu in the Structure Pane.

public static UpdateAction ACTION_MoveToFirst;
    Moves the current component to the beginning of the z-order. This action is
    added to the popup menu by default.

public static UpdateAction ACTION_MoveToLast;
    Moves the current component to the end of the z-order. This action is added
    to the popup menu by default.

public static UpdateAction ACTION_Serialize;
    Serializes the selected component to a disk file. This action is added to the
    popup menu in the Structure Pane.

```

BorderCornerLayoutAssistant Example

To demonstrate some of the abilities of a layout assistant, you can develop your own. But first you need a layout manager to work with. In this case an enhanced `BorderLayout` is used. As well as positions along each side of the container, the `BorderCornerLayout` also has locations in each corner. Similar to the `BorderLayout`, these are denoted by string constraints: `NORTHEAST`, `NORTHWEST`, `SOUTHEAST`, and `SOUTHWEST`.

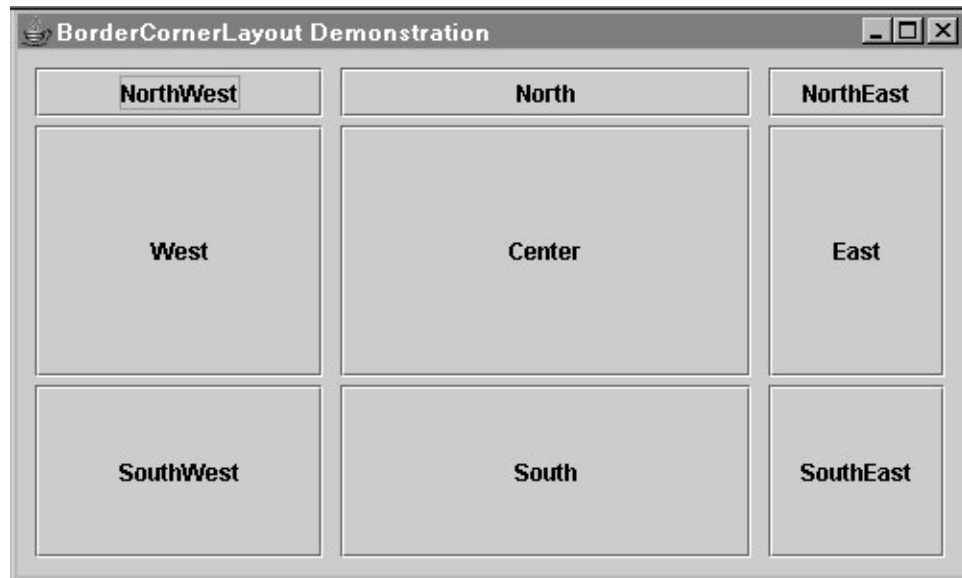
The `setCornersContribute` method takes a boolean value and sets an internal flag with it. When set to true (the default), any components in the corner positions are included in determining the preferred sizing for the borders. When false, only the horizontal and vertical components are considered (unless these

positions are empty). The standard `hgap` and `vgap` properties control spacing between the components.

Components are sized to the maximum of their preferred dimensions around the edges of the layout, with the center taking whatever is left. For example, the maximum width of the northwest, west, and southwest components (assuming `getCornersContribute` returns true) is used as the western width. Similarly, the maximum height of the northwest, north, and northeast components is used as the northern height.

An example of the functionality of the finished layout manager is shown in Figure 22–2.

Figure 22–2. The *BorderCornerLayout* positions.



The next step is to start the layout assistant. Create a new class and derive it from `BasicLayoutAssistant`. Add the `OpenTools` initialization routine and have it register this class with the `UIDesigner` for the `BorderCornerLayout` layout manager. Listing 22–1 shows the complete layout assistant class.

Listing 22–1. A layout assistant for *BorderCornerLayout*.

```
package wood.keith.opentools.layouts;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Point;
import java.beans.PropertyEditor;
import java.util.Enumeration;
import javax.swing.JComponent;

import com.borland.jbuilder.designer.ui.BasicLayoutAssistant;
import com.borland.jbuilder.designer.ui.DesignView;
import com.borland.jbuilder.designer.ui.ModelNode;
import com.borland.jbuilder.designer.ui.SelectNib;
import com.borland.jbuilder.designer.ui.UIDesigner;
import com.borland.jbuilder.designer.ui.opt.SelectBoxes;
import com.borland.primetime.PrimeTime;
import com.borland.primetime.actions.ActionGroup;

/**
 * An assistant to help in the visual construction
```

```

* of a BorderLayout.
*
* @author Keith Wood (kbwood@iprimus.com.au)
* @version 1.0 30 August 2001
*/
public class BorderLayoutAssistant extends BasicLayoutAssistant {

    private static final String VERSION = "1.0";

    /**
     * Register the assistant, which also makes the associated layout
     * available in the drop-down box in the designer.
     *
     * @param majorVersion the major version of the current OpenTools API
     * @param minorVersion the minor version of the current OpenTools API
     */
    public static void initOpenTool(byte majorVersion, byte minorVersion) {
        if (majorVersion != PrimeTime.CURRENT_MAJOR_VERSION) {
            return;
        }
        UIDesigner.registerAssistant(BorderCornerLayoutAssistant.class,
            "wood.keith.opentools.layouts.BorderCornerLayout", true);
        if (PrimeTime.isVerbose()) {
            System.out.println("Loaded BorderLayoutAssistant v" +
                VERSION);
            System.out.println("Written by Keith Wood (kbwood@iprimus.com.au)");
        }
    }

    private BorderLayoutPropertyEditor propertyEditor = null;

    public BorderLayoutAssistant() {
    }

    /**
     * Convert a point into one of the areas within this layout.
     *
     * @param node the node for the container that uses this layout
     * @param point the point to convert into an area
     * @return the constraint name for this area
     */
    private String findArea(ModelNode node, Point point) {
        Dimension size = node.getLiveComponent().getSize();
        if (point == null) {
            // No specific drop point, then try to find a spare spot
            HashMap used = new HashMap(9);
            Enumeration children = node.children();
            while (children.hasMoreElements()) {
                Object child = children.nextElement();
                if (child instanceof ModelNode) {
                    String area = ((ModelNode)child).
                        getConstraints().getValueSource();
                    area = area.substring(area.indexOf('.') + 1);
                    used.put(area, area);
                }
            }
            for (int index = 0; index < BorderLayout.AREAS.length;
                index++) {
                if (used.get(BorderCornerLayout.AREAS[index].toUpperCase()) ==
                    null) {
                    return BorderLayout.AREAS[index];
                }
            }
            // Default to centre
            return BorderLayout.CENTER;
        }
        int limitW = (int)(size.getWidth() / 5);
        int limitH = (int)(size.getHeight() / 5);
        int areaIndex = getSection(point.getX(), size.getWidth(), limitW) +
            getSection(point.getY(), size.getHeight(), limitH) * 3;
    }

```

```

        return BorderLayout.AREAS[areaIndex];
    }

    /**
     * Retrieve the type of constraints used by this layout.
     *
     * @return the constraint type
     */
    public String getConstraintsType() {
        return "java.lang.String";
    }

    /**
     * Retrieve the class name of the layout for display.
     *
     * @param node the node for the container using the layout
     * @return the layout's class name (without any path info)
     */
    private String getLayoutClassName(ModelNode node) {
        String name = node.getSubcomponent().getAsContainer().getLayout().
            getClass().getName();
        int index = name.lastIndexOf('.');
        return (index == -1 ? name : name.substring(index + 1));
    }

    /**
     * Retrieve the property editor for this layout's constraints.
     *
     * @return the appropriate property editor
     */
    public PropertyEditor getPropertyEditor() {
        if (propertyEditor == null) {
            propertyEditor = new BorderLayoutLayoutPropertyEditor();
        }
        return propertyEditor;
    }

    /**
     * Convert a positional value into a section number (0..2).
     *
     * @param pos the position within the range
     * @param max the maximum position (minimum is zero)
     * @param limit the inset for the two edge sections
     * @return 0 if in the left inset, 1 if in the middle,
     *         2 if in the right inset
     */
    private int getSection(double pos, double max, int limit) {
        return (pos < limit ? 0 : (pos > max - limit ? 2 : 1));
    }

    /**
     * Add items to the popup menu. Nothing to add.
     */
    public void prepareActionGroup(ActionGroup actionGroup) {
    }

    /**
     * Add a component to the container using this layout (including
     * drag-and-drop) with the appropriate layout constraint.
     * If an existing component is at the new position,
     * swap it to the old position.
     *
     * @param addNode the node for the component being added
     * @param point the point at which to add the component
     * @param dimension null if the component was just dropped, or
     *                  the dragged size of the component being added
     */
    public void prepareAddComponent(ModelNode addNode, Point point,
        Dimension dimension) {
        ModelNode contNode = (ModelNode) addNode.getParent();

```

```

// Get area dropped into
String area =
    "BorderCornerLayout." + findArea(contNode, point).toUpperCase();
// Locate any existing occupant
ModelNode occupant = null;
Enumeration children = contNode.children();
while (children.hasMoreElements()) {
    Object child = children.nextElement();
    if (child instanceof ModelNode) {
        ModelNode childNode = (ModelNode)child;
        if (childNode.getConstraints().getValueSource().equals(area)) {
            occupant = childNode;
            break;
        }
    }
}
// Set the new area for the dropped component
String oldArea = addNode.getConstraints().getValueSource();
addNode.getConstraints().setValueSource(area);
if (occupant != null) {
    // And swap an existing occupant to its original position
    occupant.getConstraints().setValueSource(oldArea);
}
}

/**
 * Provide feedback to the user when about to add a component.
 * Show the container name and its prospective constraint.
 *
 * @param addNode    the node for the component being added
 * @param contNode    the node for the container using the layout
 * @param point       the point at which to add the component
 * @param dimension   null if the component was just dropped, or
 *                    the dragged size of the component being added
 * @return the feedback string
 */
public String prepareAddStatus(ModelNode addNode, ModelNode contNode,
    Point point, Dimension dimension) {
    return contNode.getName() + " (" + getLayoutClassName(contNode) +
        "): " + findArea(contNode, point);
}

/**
 * Initialisation when setting this layout as the manager to use.
 *
 * @param contNode    the node having its layout set
 */
public void prepareChangeLayout(ModelNode contNode) {
    contNode.getModel().getComponentSource().getSourceFile().
        addImport("wood.keith.opentools.layouts.*");
    super.prepareChangeLayout(contNode);
}

/**
 * Provide feedback to the user when hovering over components.
 * Show the current component name and its constraint, or,
 * if no component, the container name and prospective constraint.
 *
 * @param curNode     the node for the component the mouse is over
 * @param contNode     the node for the container using the layout
 * @param point        the point at which the mouse is located
 * @return the feedback string
 */
public String prepareMouseMoveStatus(ModelNode curNode,
    ModelNode contNode, Point point) {
    return (curNode != null ? curNode.getName() + ": " +
        curNode.getConstraints().getValueText() :
        contNode.getName() + " (" + getLayoutClassName(contNode) + ")");
}

```

```

/**
 * Provide feedback to the user when dragging a component.
 * Show the container name and its prospective constraint.
 * Also display a rectangle to show the corresponding location.
 *
 * @param curNode      the node for the component being moved
 * @param contNode     the node for the container using the layout
 * @param location     the point to which to move the component
 * @param selectBoxes  reference for drawing the feedback rectangle
 * @param offset       the point within the component where
 *                    the mouse clicked
 * @return the feedback string - displayed on the status line
 */
public String prepareMoveStatus(ModelNode curNode,
    ModelNode contNode, Point location, SelectBoxes selectBoxes,
    Point offset) {
    Container container = contNode.getSubcomponent().getAsContainer();
    Dimension size = container.getSize();
    Point absLocation = DesignView.componentAbsLocation(container);
    int limitW = (int)(size.getWidth() / 5);
    int limitH = (int)(size.getHeight() / 5);
    int areaIndex =
        getSection(location.getX(), size.getWidth(), limitW) +
        getSection(location.getY(), size.getHeight(), limitH) * 3;
    // Display a positioning outline
    Point point =
        new Point((int)absLocation.getX() + (areaIndex % 3 == 0 ? 0 :
            (areaIndex % 3 == 1 ? limitW : (int)size.getWidth() - limitW)),
            (int)absLocation.getY() + ((int)(areaIndex / 3) == 0 ? 0 :
                ((int)(areaIndex / 3) == 1 ? limitH :
                    (int)size.getHeight() - limitH)));
    Dimension showSize = new Dimension((areaIndex % 3 == 1 ?
        (int)size.getWidth() - 2 * limitW : limitW),
        ((int)(areaIndex / 3) == 1 ?
            (int)size.getHeight() - 2 * limitH : limitH));
    selectBoxes.show(0, point, showSize, 2);
    // Return the feedback value for the status bar
    return contNode.getName() + " (" + getLayoutClassName(contNode) +
        "): " + BorderCornerLayout.AREAS[areaIndex];
}

/**
 * Resize the component's preferred dimensions.
 *
 * @param node        the node for the component selected
 * @param selectNib   the resizing handle being dragged
 */
public void prepareResizeComponent(ModelNode node,
    SelectNib selectNib) {
    JComponent component = (JComponent)node.getLiveComponent();
    component.setPreferredSize(selectNib.getRectangleDimension());
}

/**
 * Provide feedback for resizing a component's preferred dimensions.
 *
 * @param node        the node for the component being resized
 * @param point       the location of the mouse
 * @param dimension   the new dimensions of the component
 * @return the feedback string - displayed on the status line
 */
public String prepareResizeStatus(ModelNode node, Point point,
    Dimension dimension) {
    return node.getName() + ": " +
        (int)dimension.getWidth() + "x" + (int)dimension.getHeight();
}

/**
 * Set up when user selects a component.
 * Provide a resizing handle if JComponent selected.

```

```

*
* @param node        the node for the component selected
* @param designView  the designer
*/
public void prepareSelectComponent(ModelNode node,
    DesignView designView) {
    super.prepareSelectComponent(node, designView);
    if (node.getLiveComponent() instanceof JComponent) {
        // Establish a resizing handle for JComponents -
        // #3 is bottom right
        SelectNib[] nibs = designView.assureNibs(4);
        nibs[3].setBackground(Color.black);
        nibs[3].setLayoutAssistant(this);
        nibs[3].setSelectable(true);
        nibs[3].target = node;
    }
}

/**
 * Provide feedback for resizing a component's preferred dimensions.
 * Also draw an outline showing the new size.
 *
 * @param point        the current location of the mouse
 * @param designView  the designer
 * @param selectNib    the resizing handle being dragged
 * @return the feedback string - displayed on the status line
 */
public String resizeAction(Point point, DesignView designView,
    SelectNib selectNib) {
    Point nibPoint = selectNib.getRectangleLocation();
    Dimension dimension = selectNib.getRectangleDimension();
    // Update rectangle size based on nib type and location
    designView.adjustPositionForNib(nibPoint, dimension, point,
        selectNib.type);
    // Draw the outline for the new size
    designView.getTempComponent().show(0, nibPoint, dimension, 2);
    return prepareResizeStatus((ModelNode)selectNib.target, point,
        dimension);
}
}

```

The assistant divides up the container area in a standardized manner, allocating one fifth of the height and width to the border regions, and leaving the remainder to the center. This overcomes complex calculations for each contained component, as well as working when there are no components along a particular edge. The `findArea` method converts a point within the container into one of the constraint values based on the divisions above. Retrieve the size of the container through its CMT node as shown below:

```
Dimension size = node.getLiveComponent().getSize();
```

The constraints for this layout come from a limited list of string values. Hence the `getConstraintsType` routine returns “`java.lang.String`” as the type of the constraints used. To support this within the designer, the layout assistant returns an instance of a property editor from the `getPropertyEditor` method. Property editors follow the JavaBeans specification and the `java.beans.PropertyEditor` interface. In this case, the editor simply returns the list of possible values, as well as the fully qualified constant equivalent to each of these values. The latter are used to generate the code snippet that sets the value for the constraint within the source file.

Used internally, the `getLayoutClassName` method retrieves the name of the layout manager class for the specified component (the container), while the

`getSection` method converts a position on the screen into a value indicating which portion of the layout it belongs to.

This assistant adds no entries to the popup menu in the designer, however its ancestor class does add items. Thus the `prepareActionGroup` method must be defined and left empty to avoid this behavior.

When a new component is added to the container or an existing component is dragged to a new position and dropped, JBuilder calls the `prepareAddComponent` method. This first determines the constraint for the new addition before checking whether or not that position is already occupied. If there is an occupant, its location is set to that of the new one before its move, swapping their positions. The constraint values are set via the corresponding CMT nodes:

```
addNode.getConstraints().setValueSource(area);
```

After a component is selected from the Palette, the layout assistant provides feedback through the `prepareAddStatus` method as the mouse indicates the position to place it. Here you display the name of the container and its layout manager class, along with the constraint for the current point under the mouse.

When moving the mouse around without having selected a component, the status bar shows the current component and its constraint via the `prepareMouseMoveStatus` method. If the mouse is not over a contained component, the name of the container itself is displayed, along with the name of its layout class.

If you drag an existing component within its container, JBuilder invokes the `prepareMoveStatus` method. Return the text to display in the Status Pane – in this case the container name, the name of its layout class, and the constraint for the current mouse position. Retrieve a reference to the container itself with the following code, along with its absolute position within the designer window:

```
Container container = contNode.getSubcomponent().getAsContainer();
Point absLocation = DesignView.componentAbsLocation(container);
```

In addition, you can draw a selection box on the container to indicate graphically the new location. Use the `SelectBoxes` reference passed into the routine to display this box. Its `show` method takes an integer index value, the origin point of the rectangle (with respect to the designer itself) and its size, and the width of the line used.

```
selectBoxes.show(0, point, showSize, 2);
```

Although this layout ultimately controls the sizes of the contained components, it does take into account the preferred sizes of those around the edges. Thus, the `prepareResizeComponent` method responds to changes in a suitable resizing nib to alter the component's preferred dimensions.

```
component.setPreferredSize(selectNib.getRectangleDimension());
```

Meanwhile, the `prepareResizeStatus` method provides suitable status bar feedback for the resizing nib as it is dragged. This nib is established in the `prepareSelectComponent` method by modifying one of the standard nibs. Further feedback comes from the `resizeAction` method in the form of an outline showing the new size. You must adjust the rectangle for the outline based on the current position of the mouse and the type of nib being dragged. This is achieved via the following call:

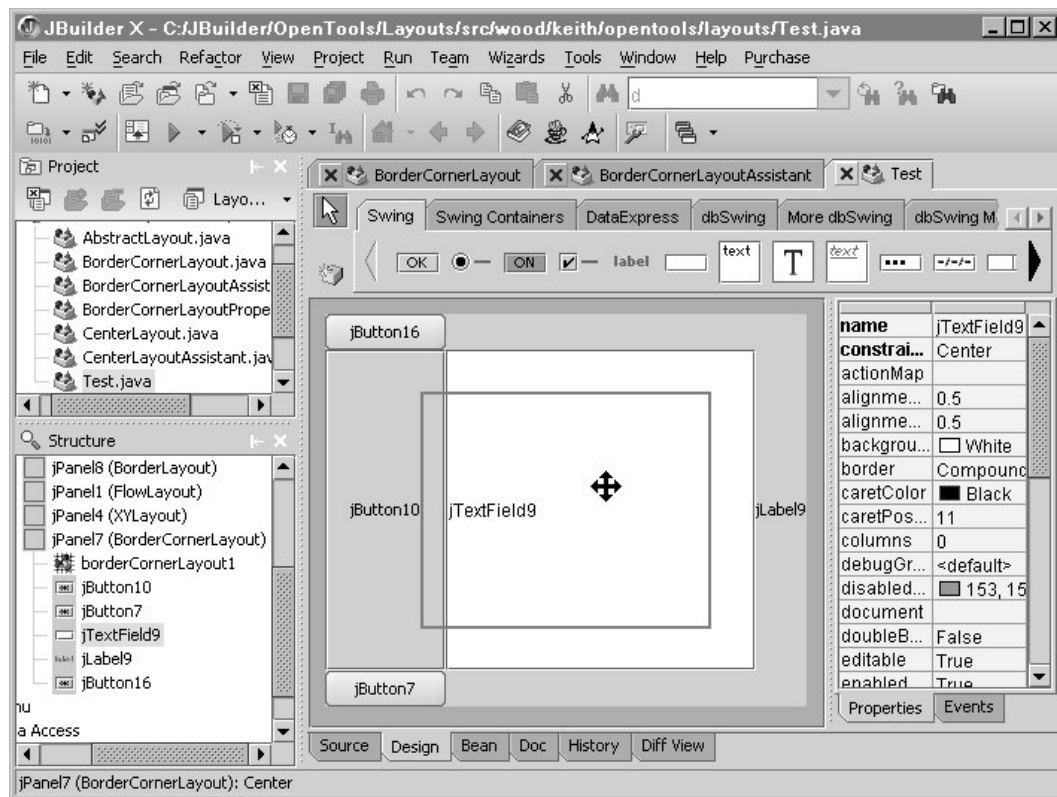
```
DesignView.adjustPositionForNib(
    nibPoint, dimension, point, selectNib.type);
```

As before, you need to compile the classes for the layout itself, the layout assistant, and its property editor, and place them into a JAR file. The manifest entry for this assistant should read:

```
OpenTools-Designer:
  wood.keith.opentools.layouts.BorderCornerLayoutAssistant
```

Place the completed JAR file in the {JBuilder}/lib/ext directory and restart JBuilder. The `BorderCornerLayout` is now available for selection as a layout property. Once in use, the assistant comes into play to guide you through its layout policy as shown in Figure 22–3.

Figure 22–3. The `BorderCornerLayoutAssistant` in action.



Summary

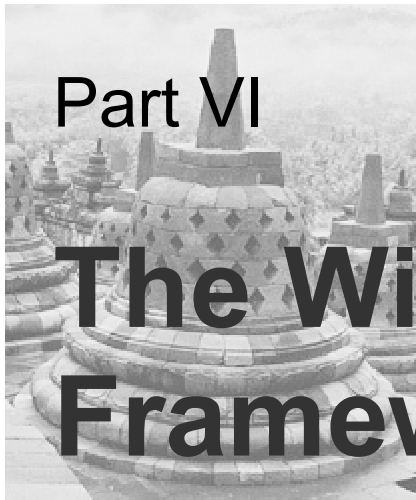
Although layout managers are great at managing the placement and sizing of components within a GUI application, they are not so friendly to graphical development environments. JBuilder comes to the rescue with the `LayoutAssistant` interface and `BasicLayoutAssistant` abstract class.

These provide feedback during placement of components within the container, and during repositioning and resizing operations. Unfortunately, the layout assistant API is undocumented, making it difficult to use to its full effect. This chapter should provide enough detail to overcome that limitation and includes an example to get you on your way.

A single sample of the use of a layout assistant comes with JBuilder, as shown below.

```
{JBuilder}/samples/OpenToolsAPI/LayoutAssistant/  
LayoutAssistant.jpx
```

A very basic assistant that provides some feedback for the `RowLayout` manager. This layout extends `GridLayout`, but restricts it to a single row with columns of equal width. The assistant, as well as adding the layout to the drop-down list, displays the new position as a continually movable selection box, rather than the jumping box of the `GridLayout` assistant.



Part VI

The Wizard Framework

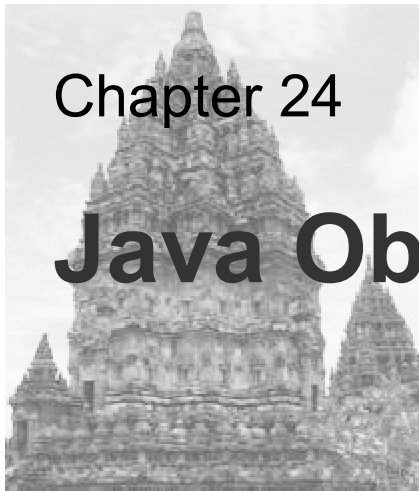
JBuilder's purpose as a Java IDE is to make things easier for the developer. Its many enhancements within the editor environment allow you to attach keystroke combinations to actions – either those that are built-in or those that you create yourself. You can call up CodeInsight features, or have them pop up automatically, to remind you of what methods and fields are available for an object, and to insert references to them to reduce typing. The designer lets you drag-and-drop components when constructing your user interface as well as some non-visual aspects of your code.

Another important area where JBuilder assists you in the development process is through the use of *wizards*. Wizards are usually presented as a series of pages within a dialog box that lead you through some, often complex, procedure. They are also used for simple tasks such as creating a new class. Although a similar result could be achieved by calling on a code template, a wizard is interactive, allowing you to alter the generated code through data entered in the dialog.

More complex activities can be automated and simplified through wizards so that you do not have to remember all the steps necessary to reach your goal. Thus you end up with more complete and correct code every time.

To assist you in working with Java files (often the input or output of a wizard), JBuilder also provides the Java Object Toolkit (JOT). This set of interfaces and classes let you easily interact with Java code, in both source and compiled formats. You can use it to parse existing files to retrieve information, and to generate new Java code from scratch. It also enables you to make very precise modifications to existing code.

Of course, the framework that supports wizards in JBuilder is available for your use through the OpenTools API. Chapter 23 describes the wizard framework, while Chapter 24 discusses JOT, the object model for Java code.



Chapter 24

Java Object Toolkit

The Java Object Toolkit (JOT) is a collection of interfaces and classes that allows you to easily interact with Java code, in both source and compiled formats. You can use it to parse existing files to retrieve information, and to generate new Java code from scratch. It also enables you to make very precise modifications to existing code.

Several parts of JBuilder make use of JOT. The Component Modeling Tool (CMT) discussed in Chapter 20 relies on JOT to locate the components and their properties that it deals with. Numerous wizards use JOT to extract information from existing Java code and process it (such as the Resource Strings wizard), to alter existing code (like the Implement Interface wizard), or to generate new Java code (most items in the Object Gallery).

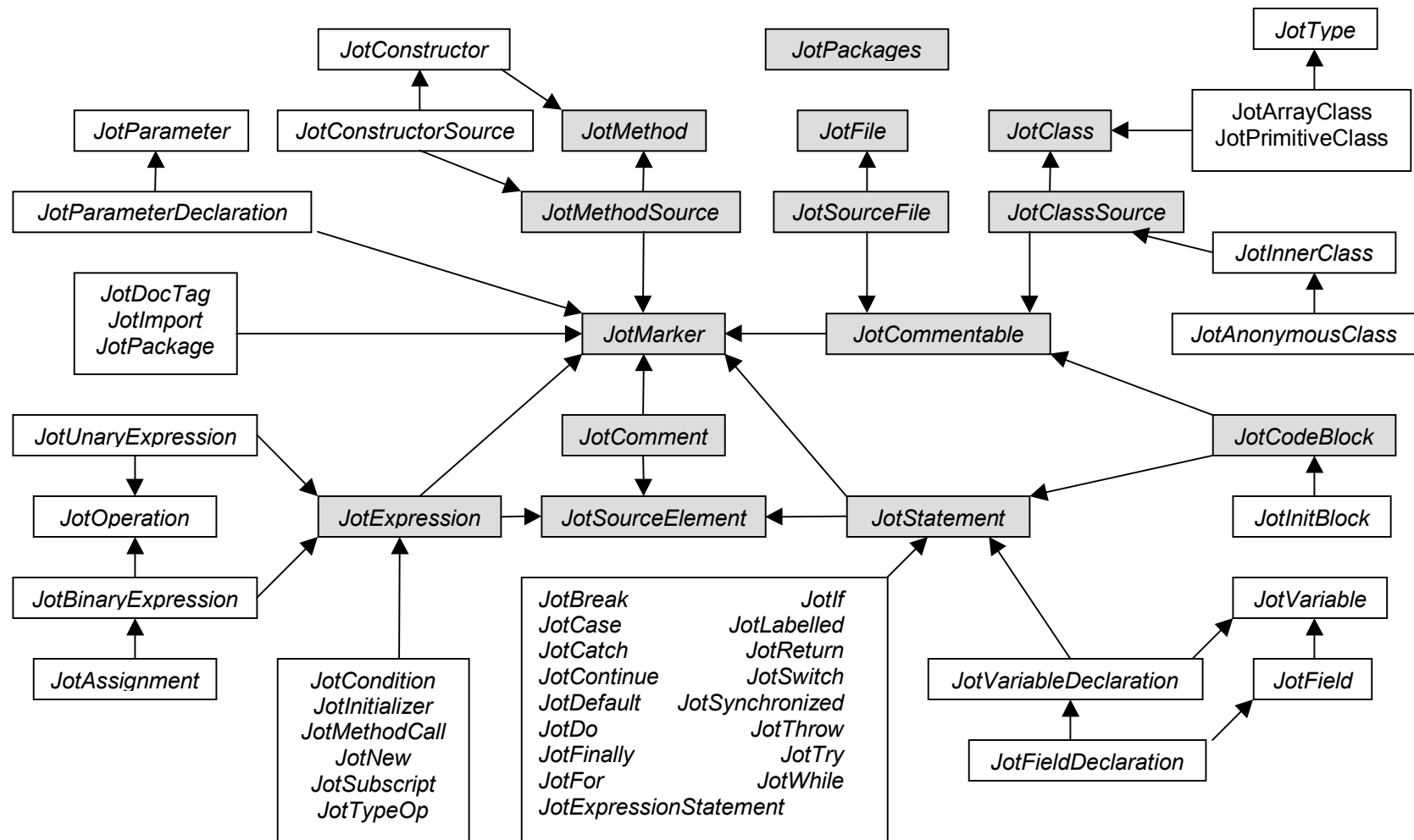
The Tag Library Descriptor wizard presented in Chapter 23 draws on JOT to examine the Java contents of a project to extract the information that it needs to operate. It starts by obtaining the package manager (`JotPackages`) from the project, and then accesses each Java file in the project (`JotFile`) from which it obtains the list of classes (`JotClass`) declared there. Each class is examined to see whether it implements one of the JSP tag interfaces. If so, the wizard retrieves further details about that class. At the end it frees up any resources acquired by JOT during this process.

```
JotPackages jpackages = project.getJotPackages();
JotFile jfile = jpackages.getFile(fileNode.getUrl());
JotClass[] jclassses = jfile.getClasses();
for (int index = 0; index < jclassses.length; index++) {
    // Process this class - jclassses[index]
}
jpackages.release(jfile);
```

Within each class, the wizard reads the comment (`JotComment`) describing it and then scans through each of its methods (`JotMethod`) looking for setters that may indicate attributes of the tag. All the information is displayed in the wizard to allow the user to review and alter or enhance it before generating the TLD file.

```
JotComment jcomment =
    ((JotSourceFile)jfile).getComment((JotClassSource)jclass, BEFORE);
JotMethod[] jmethods = jclass.getClasses();
for (int index = 0; index < jmethods.length; index++) {
    if (jmethods[index].getName().substring(0, 3).equals("set")) {
        // Process this method
    }
}
```

Figure 24-1. JOT class hierarchy.



The hierarchy of interfaces and classes that make up JOT is shown in Figure 24-1. Interfaces are shown in italics, while classes are in normal type. Inheritance is indicated by the arrows (since most of the items are interfaces, multiple inheritance is possible). The items that are shaded are those that are described in detail in this chapter.

NOTE

The JOT interfaces and classes belong to the `com.borland.jbuilder` branch of the package structure since they are directly related to Java code. This follows the split between the PrimeTime packages (the generic IDE framework) and JBuilder (the Java-specific IDE).

JotPackages Interface

When using JOT you usually start out with the package manager as encapsulated in the `com.borland.jbuilder.jot.JotPackages` interface. From it you have access to the Java files and classes within the project. You retrieve an instance of this interface from your current (JBuilder) project:

```
JotPackages jpkgs = project.getJotPackages();
```

The methods of this interface are:

```
public boolean checkReread(JotSourceFile file);
```

Determine whether some other process has changed the source file, and force a reload if it has. Any object references that you had on the original file are then invalid.

`file` is the source file to verify.

```
public void commit(JotSourceFile file);
```

Write out any changes to the file to its underlying stream.

`file` is the modified file to save.

```
public void commitAs(JotSourceFile file, String fileName);
```

```
public void commitTo(JotSourceFile file, String fileName);
```

Send the changes to a file to a different destination.

`file` is the modified file to save.

`fileName` is the name of the new file to write to.

WARNING

The `commitAs` and `commitTo` methods are deprecated in all JBuilder versions.

```
public JotClass getClass(String className, int needs);
```

```
public JotClass getClass(JotSourceFile importContext,
    String className, int needs);
```

Retrieve JOT class information for a given class, or null if it cannot be found.

`importContext` is the file to search.

`className` is the full name of the class required.

`needs` is one of the constants below to define whether you require source or class files.

```
public String getEncoding();
```

Get the character encoding used for the files.

```
public JotFile getFile(Url url);
```

Given a `Url`, find the JOT file for it. Use `getSourceFile` to obtain an updateable version.

`url` is the `Url` to read.

```
public Iterator getFiles(String packageName, int needs);
```

```
public ArrayList getFilesArray(String packageName, int needs);
```

Obtain an iterator for or a list of the files (as `Urls`) in a package.

`packageName` is the full name of the package to scan. Either dots or the file separator character may separate the name parts.

`needs` is one of the constants below to define whether you require source or class files.

```
public String getPackage(JotSourceFile importContext, String className, int needs);
```

Find the package name for a given class.

`importContext` is the file to search.

`className` is the full name of the class required.

`needs` is one of the constants below to define whether you require source or class files.

```
public Iterator getPackages(int needs);
```

```
public ArrayList getPackagesArray(int needs);
```

Obtain an iterator for or a list of the package names (as `Strings`) for this project.

`needs` is one of the constants below to define whether you require source or class files.

```
public JotSourceFile getSourceFile(Url url);
```

Given a `Url`, find the JOT source file for it. Use `getFile` to obtain a read-only version.

`url` is the `Url` to read.

```
public String getSourceVersion();
```

Returns the source version (minimum JDK perhaps?) to use when parsing the code. Currently returns "1.2".

```
public Url getUrl(String className, int needs);
```

Retrieve the `Url` for the file that contains a class.

`className` is the full name of the class required.

`needs` is one of the constants below to define whether you require source or class files.

```
public Class loadClass(String className) throws ClassNotFoundException;
```

Get standard class information for a class. An exception is thrown if the class cannot be found.

`className` is the full name of the class required.

```
public void release(JotFile file);
```

Free up any resources allocated to a file, and flush its buffers.

`file` is the file to free up.

```
public void releaseAll();
```

Release all resources and flush buffers for all the files attached to this package manager.

```
public void shutdown();
```

Close down the package manager and clear all caches.

The following constants are defined in this interface:

```
public static final int NEED_ANY;
```

```
public static final int NEED_CLASS;
```

```
public static final int NEED_SOURCE;
```

Use these in various methods above to define what sort of access you require to the Java code: anything, only class files, or only source files.

JotFile Interface

From the package manager for JOT, you can access the files that make up a package. Each one is represented by an instance of the `com.borland.jbuilder.jot.JotFile` interface and may be a Java source or class file. In the latter case, you have read-only access to the file.

Retrieve the file reference by using the following code:

```
JotFile jfile = jpackages.getFile(url);
```

Its methods are:

```
public JotClass getClass(String className);
```

Get JOT class details for a particular class in the file, or `null` if it cannot be found.

`className` is the name of the class required, either just the class name or including its full path.

```
public JotClass[] getClasses();
```

Obtain a list of all the top-level classes defined in this file. If there are none, an empty array is returned.

```
public String getName();
```

Return the full name of the file.

```
public String getPackage();
```

Get the name of the package that contains this file.

```
public JotPackages getPackageManager();
```

Obtain a reference to the package manager for this file.

```
public long getTimestamp();
```

Find out the last modified date for this file, expressed as a long value.

```
public Url getUrl();
```

Retrieve the `Url` for this file.

JotSourceFile Interface

If the file retrieved above is based on Java source code, then you have the ability to update it through the `com.borland.jbuilder.jot.JotSourceFile` interface, which extends `JotFile`. As well as modifying the items returned through the parent interface, you can alter other items that apply to the file as a whole.

You can use the package manager to locate the source for you:

```
JotSourceFile jsource = jpackages.getSourceFile(url);
```

Or you can use `instanceof` to check whether you have the source available and then cast a JOT file reference accordingly:

```
if (jfile instanceof JotSourceFile)
    JotSourceFile jsource = (JotSourceFile)jfile;
```

This interface adds the following methods:

```
public JotClassSource addClass(JotMarker marker, boolean
    before, String name, boolean isInterface);
```

Construct a class source object from the given details and add it to the file. A reference to the new class is returned.

`marker` is an existing item in the file to place the new class beside. If `null`, the new class is added at the end.

`before` is true to insert the new class before the marker in the file, or false to place it after the marker.

`name` is the name of the new class (without any package name).

`isInterface` is true if this is an interface declaration, or false if it is a class.



WARNING

The `addClass` method is the only way to create new classes or interfaces within a JOT file. You cannot convert an existing class into an interface, nor the reverse. The best you can do is to create a new class or interface and copy the contents of the original into it.

```
public JotImport addImport(String name);
```

Create an import statement and insert it into the file. A reference to it is returned.

`name` is the full name of the class or package to import, such as “`java.util.*`” or “`javax.swing.JPanel`”.

```
public void addJotFileListener(JotFileListener listener);
```

Add a new object to notify of events affecting the file.

`listener` is the object to add.

```
public int getComparableLocation(JotMarker marker);
```

Convert marker locations into integer values that can be compared for order. The marker with the lower integer value occurs earlier in the file.

`marker` is the item from the file to locate.

```
public String getFullClassName(String className);
```

Obtain the full class name for a given class as if it had been declared in this file.

`className` is the base name of the class.

```
public JotImport getImport(String name);
```

```
public JotImport[] getImports();
```

Retrieve object(s) representing one or all of the import statements in the file. A `null` is returned from the first version if the specified import is not found, while an empty array is returned from the second if none exist.

`name` is the name of the class or package being imported, like “`java.util.*`”.

```
public boolean isReadOnly();
    Returns true if the file cannot be modified, or false if it can be altered.
    Although the file comes from a Java source file, it may be marked as read-
    only itself.
```

```
public String out();
    Obtain the content of the source file from this method.
```

```
public void removeClass(JotClass clazz);
    Erase a class from the file.
    clazz is a reference to the class to remove.
```

```
public void removeImport(JotImport imp);
    Delete an import statement from the file.
    imp is a reference to the import statement to remove.
```

```
public void removeJotFileListener(JotFileListener
    listener);
    Delete a listener for events on the file.
    listener is the object to remove.
```

```
public void reRead();
    Force a reload of this file from its source. Any references to existing objects
    from the file then become invalid. This method is intended for internal use
    only.
```

```
public void setPackage(String packageName);
    Establish the package to which this file belongs.
    packageName is the full name of the package.
```

```
public void setTimestamp(long time);
    Update the timestamp for this file.
    time is the new timestamp as a long value.
```

JotMarker Interface

Most source elements also implement the `com.borland.jbuilder.jot.JotMarker` interface to allow you to locate it within the file. Generally when adding new code you place it either immediately before or after another element.

The abilities of this interface are:

```
public void addUserData(Object key, Object data);
    Associate additional data with this element.
    key is the identifier for the data.
    data is the extra information encapsulated in an object. Erase the data for a
    key by sending this as null.
```

```
public int getEndPosition();
public int getStartPosition();
    Find the character position of the end or start of this marker within the file.
```

```
public Object getUserData(Object key);
    Retrieve some additional data about this element, or null if there is none
    there.
    key is the identifier for the data.
```

JotFileListener Interface

You can react to several events occurring on JOT files by registering an instance of the `com.borland.jbuilder.jot.JotFileListener` interface with each file.

The methods called in response to the file events are:

```
public void fileClassChanged(JotFileEvent event);
```

When a class or interface is added to or removed from the file this method fires.

`event` holds details about the file and the change.

```
public void fileImportChanged(JotFileEvent event);
```

Obtain notification of changes to the file's import statements through this method.

`event` holds details about the file and the change.

```
public void fileMiscChanged(JotFileEvent event);
```

Any changes to the file apart from those listed in the other methods are notified through this one.

`event` holds details about the file and the change.

```
public void filePackageChanged(JotFileEvent event);
```

Respond to changes to the name of the package for the file.

`event` holds details about the file and the change.

JotFileEvent Class

Information about an event happening to a JOT file is transferred via a `com.borland.jbuilder.jot.JotFileEvent` object, which extends `com.borland.primetime.util.DispatchableEvent`. The one event class uses different fields based on the type of event and is passed to different methods on the listener, hence the `dispatch` method here.

The methods of this event class are as follows:

```
public JotFileEvent(JotSourceFile source, int id)
```

```
public JotFileEvent(JotSourceFile source, JotClassSource  
clazz)
```

```
public JotFileEvent(JotSourceFile source, JotImport imp)
```

Create a new JOT file event.

`source` is the file that the event affects.

`id` is one of the constants below, identifying the type of event (and thus the method it is sent to).

`clazz` refers to the class that was added or removed.

`imp` refers to the import statement that was changed.

```
public void dispatch(EventListener listener)
```

Call the appropriate method on the listener, based on the `id` of this event object.

`listener` is a `JotFileListener` instance to inform about this event.

```
public JotClassSource getClazz()
```

Returns the class that was added or removed, or `null` if this is not a class change event.

```
public JotSourceFile getFile()
```

Retrieve the file that generated this event.

```
public JotImport getImport()
```

Get the import statement that changed, or `null` if this is not an import change event.

Several constants are also defined in this class:

```
public static final int CLASS_CHANGED;
public static final int IMPORT_CHANGED;
public static final int MISC_CHANGED;
public static final int PACKAGE_CHANGED;
```

Use these as the `id` value in the first constructor to create an appropriate type of file event. The other constructors automatically set the correct value internally.

JotClass Interface

Each class defined within a file is available through JOT as an instance of the `com.borland.jbuilder.jot.JotClass` interface. This interface is a key part of parsing and creating new Java code. Once you have the class, you can access its fields, constructors, and methods, as well as its visibility and ancestor. It is very similar to the standard Java class `java.lang.Class`.

From the file you retrieve an individual class as follows:

```
JotClass jclass = jfile.getClass("wood.keith.questions.QuestionTag");
```

Or you can find all of them in the file with this:

```
JotClass[] jclasss = jfile.getClasses();
```

The methods of this interface are:

```
public void addUserData(Object key, Object data);
```

Associate user-defined data with this class, linked to a key value. Use the `getUserData` method to retrieve it again.

`key` is the identifying value for the data.

`data` is the user-defined object containing information about this class.

Passing a `null` here erases any previous data held against the key.

```
public JotClass getComponentType();
```

If this class represents an array (see the `isArray` method), then this method returns the type of the elements of that array. Otherwise it returns `null`.

```
public JotConstructor getConstructor(JotClass[]
parameterTypes);
```

Obtain a reference to a particular constructor for this class, or `null` if no match can be found. A `java.lang.SecurityException` may be thrown if you do not have access to this information. Use `getDeclaredConstructor` to consider only those constructors actually defined in this class.

`parameterTypes` is an array of the types of parameters that this constructor accepts in the order that they are declared. Pass an empty array if it takes no parameters. Do not pass `null`.

```
public JotConstructor[] getConstructors();
```

Access all the constructors of this class through this method, including any default one. It throws a `java.lang.SecurityException` if this information is inaccessible. Use `getDeclaredConstructors` for only those constructors defined in this class.

```
public JotConstructor getDeclaredConstructor(JotClass[]
parameterTypes);
```

Locate a reference to a particular constructor defined within this class, or null if it cannot be found. A `java.lang.SecurityException` may be thrown if you do not have access to this information. Use `getConstructor` to include consideration of any default constructor.

`parameterTypes` is an array of the types of parameters that this constructor accepts in the order that they are declared. Pass an empty array if it takes no parameters. Do not pass null.

```
public JotConstructor[] getDeclaredConstructors();
```

Find all the constructors defined within this class, or receive an empty array if it refers to an interface or a primitive type. Use `getConstructors` to include any default constructor.

```
public JotField getDeclaredField(String name);
```

Retrieve a reference to a particular field declared in this class (not inherited), or null if the field cannot be found. You may receive a `java.lang.SecurityException` if you do not have access to the field information. Use `getField` to look at all fields in this class, including inherited ones.

`name` is the name of the field.

```
public JotField[] getDeclaredFields();
```

Get a list of the fields declared in this class, or an empty array if none are defined here. No inherited fields are present. A `java.lang.SecurityException` may be thrown if you do not have access to this information. Use `getFields` for all fields, including inherited ones.

```
public JotClass[] getDeclaredInnerClasses();
```

Discover all the inner class declared in this class (but not those that are inherited), or an empty list if there are none. Use `getInnerClasses` for all inner classes, including inherited ones.

```
public JotMethod getDeclaredMethod(String name, JotClass[]
parameterTypes);
```

Obtain a reference to a particular method declared in this class (not inherited), or null if the method is not found. You may receive a `java.lang.SecurityException` if you do not have access to the method information. Use `getMethod` for all methods, including inherited ones.

`name` is the name of the method.

`parameterTypes` is an array of the types of parameters that this method accepts in the order that they are declared. Pass an empty array if it takes no parameters. Do not pass null.

```
public JotMethod[] getDeclaredMethods();
```

```
public JotMethod[] getDeclaredMethods(String name);
```

Get a list of the methods declared in this class, or an empty array if none are defined here. No inherited methods are present. A `java.lang.Security-`

Exception may be thrown if you do not have access to this information. Use `getMethods` for all methods, including inherited ones.

`name` is the name of a particular method to match. This is useful for overloaded methods with multiple versions. If not specified, all the methods of this class are returned.

```
public JotField getField(String name);
```

Retrieve a reference to a particular field from this class or its ancestors, or `null` if the field cannot be found. You may receive a `java.lang.SecurityException` if you do not have access to the field information. Use `getDeclaredField` to examine only those fields defined in this class itself.

`name` is the name of the field.

```
public JotField[] getFields();
```

Discover all the fields of this class and its ancestors through this method. It returns an empty array if this class or interface has no fields. For an array type, it does not include the implicit length field. You may receive a `java.lang.SecurityException` if the fields are not accessible. Use `getDeclaredFields` for only those fields defined in this class.

```
public JotFile getFile();
```

Retrieve the file to which this class belongs.

```
public JotClass[] getInnerClasses();
```

Get a list of the inner classes defined in this class and its ancestors, or an empty list if there are none. Use `getDeclaredInnerClasses` for only those inner classes declared in this class.

```
public JotType[] getInterfaces();
```

Find out what interfaces this class implements with this method. For a class, it returns an array of type references for those interfaces directly implemented here. For an interface, it returns the set of superinterfaces. If no interfaces are present, an empty array is returned.

```
public JotMethod getMethod(String name, JotClass[]
parameterTypes);
```

Obtain a reference to a particular method from this class or its ancestors, or `null` if the method is not found. You may receive a `java.lang.SecurityException` if you do not have access to the method information. Use `getDeclaredMethod` to only search through methods declared in this class.

`name` is the name of the method.

`parameterTypes` is an array of the types of parameters that this method accepts in the order that they are declared. Pass an empty array if it takes no parameters. Do not pass `null`.

```
public JotMethod[] getMethods();
```

```
public JotMethod[] getMethods(String name);
```

Retrieve the public methods of this class and its ancestors through these methods. They return an empty array if this class or interface has no methods. You may receive a `java.lang.SecurityException` if the fields are not accessible. Use `getDeclaredMethods` for methods of this class excluding any inherited ones.

name is the name of a particular method to match. This is useful for overloaded methods with multiple versions. If not specified, all the public methods are returned.

```
public int getModifiers();
```

Determine the modifiers for this class: its visibility and its abstract, final, and static status. Use the constants or methods from the `java.lang.reflect.Modifier` class to query these settings.

```
public String getName();
```

Get the full name of this class or interface.

```
public JotType getSuperclass();
```

Retrieve an object representing the type of the ancestor of this class, or `null` if this class is a primitive type, an interface, or refers to `Object` itself.

```
public JotType getType();
```

Obtain the JOT type object that corresponds to this class.

```
public Object getUserData(Object key);
```

Recall user-defined data associated with this class, or `null` if no such data exists.

key is the identifying value for the data.

```
public boolean isArray();
```

Returns true if this class denotes an array class, or false if it does not. The type of the elements is available through the `getComponentType` method.

```
public boolean isAssignableFrom(JotClass clazz);
```

Discover whether a class is assignment compatible with this class. If this class represents a primitive type then it returns true if the parameter class is exactly the same, and false otherwise. For normal classes and interfaces, the method returns true if this class is an ancestor of the parameter class, and false if it is not.

clazz is the class to assign to.

```
public boolean isInstance(Object obj);
```

Determine whether an object is an instance of this class. If this class is an interface then the method returns true if the object or one of its ancestors must implement that interface, and false otherwise. If this class is a true class then it returns true if the object or one of its ancestors is that class, and false if not. If this class is an array type then it returns true if the object is an array of the element class or one of its descendents, and false otherwise. If this class represents a primitive type the method always returns false.

obj is the object to match to this class.

```
public boolean isInterface();
```

Returns true if this class represents an interface, or false if it is a class or a primitive type.

```
public boolean isPrimitive();
```

Find out if this class denotes a primitive class, returning true if it does, or false if it does not. The `JotPrimitiveClass` class defines nine fields that correspond to the eight primitive Java types and to void (`booleanType`, `byteType`, `charType`, `doubleType`, `floatType`, `intType`, `longType`, `shortType`, and `voidType`). Only these return true for this method; all other instances return false.

```
public Object newInstance() throws InstantiationException,
    IllegalAccessException;
```

Try to create a new instance of this class and return a reference to it. An `InstantiationException` is thrown if this is an abstract class, an interface, or a primitive type, while an `IllegalAccessException` indicates that the class or its initializer is not accessible.

Derived from `JotClass` are the `JotAnonymousClass` and `JotInnerClass` interfaces, which represent (as their names suggest) anonymous and inner classes within your class.

JotClassSource Interface

If a source file is available for a class, the `JotClass` object also implements the `com.borland.jbuilder.jot.JotClassSource` interface, allowing you to update its contents. In this way you can build new classes, or alter existing classes in the file.

Its extended abilities are:

```
public JotConstructorSource addConstructor(JotMarker
    marker, boolean before);
```

Generate a new constructor for this class, add it to the code, and return a reference to it.

`marker` is a reference to another item in the class to use as a base for positioning this constructor. If `null`, it is added at the end of the class.

`before` is `true` to insert the constructor before the marker above, or `false` to place it after the marker.

```
public JotFieldDeclaration addField(JotMarker marker,
    boolean before, String variableType, String name);
```

Insert a field declaration into the class and return a reference to it.

`marker` and `before` are the same as above.

`variableType` is the data type for the variable.

`name` is the name of the new variable.

```
public JotInitBlock addInitBlock(JotMarker marker, boolean
    before, int modifiers);
```

Add a new initialization block to the class and return a reference to it.

`marker` and `before` are the same as above.

`modifiers` is either `java.lang.reflect.Modifier.STATIC` or zero. If the former, the block become a static initializer, otherwise it becomes an instance initializer.



WARNING

Unfortunately, there is no way to retrieve the initialization block once created, so hang onto the returned reference.

```
public JotInnerClass addInnerClass(JotMarker marker,
    boolean before, String name, boolean isInterface);
```

Create a new inner class or interface and add it to this class, returning a reference to the new item.

`marker` and `before` are the same as above.

`name` is the name of the new class or interface.

`isInterface` is true to construct an interface, or false to create a class.

```
public JotType addInterface(JotMarker marker, boolean
before, String name);
```

Add a new interface to this class and return a reference to its type.

`marker` and `before` are the same as above.

`name` is the name of the interface, and does not have to be fully qualified.

```
public JotMethodSource addMethod(JotMarker marker, boolean
before, String returnType, String name);
```

Generate a new method implementation for this class and return a reference to it. Use the `addMethodDeclaration` method to add abstract methods.

`marker` and `before` are the same as above.

`returnType` is the data type of the return value from the method.

`name` is the name of the new method.

```
public JotMethodSource addMethodDeclaration(JotMarker
marker, boolean before, String returnType, String name);
```

Create a new abstract method or a prototype for an interface, insert it into the class, and return a reference to it. Use the `addMethod` method for methods implemented in this class.

`marker` and `before` are the same as above.

`returnType` is the data type of the return value from the method.

`name` is the name of the new method.

```
public int getComparableLocation(JotMarker marker);
```

Convert marker locations into integer values that can be compared for order. The marker with the lower integer value occurs earlier in the file.

`marker` is the item from the file to locate.

```
public int getDeclaredModifiers();
```

Retrieve the modifiers actually declared in the file. Use the fields and methods of `java.lang.reflect.Modifier` to query this value.

```
public JotSourceFile getDeclaringFile();
```

Obtain a reference to the source file that contains this class.

```
public void removeConstructor(JotConstructor constructor);
```

```
public void removeField(JotField field);
```

```
public void removeInitBlock(JotInitBlock block);
```

```
public void removeInnerClass(JotInnerClass inner);
```

```
public void removeInterface(JotType iface);
```

```
public void removeMethod(JotMethod method);
```

Remove the specified item from this class.

`constructor`, `field`, `block`, `inner`, `iface`, or `method` refer to the item to be deleted.

```
public void setModifiers(int modifiers);
```

Update the modifiers for this class – its visibility and whether it is abstract, final, and/or static.

`modifiers` is the collection of modifiers for this class. Use the constants from the `java.lang.reflect.Modifier` class to build the value, OR-ing them together.

```
public void setName(String name);
```

Set the unqualified name for this class.

name is the new class name (without any package name).

```
public void setSuperclass(String superClass);
```

Alter the name of the ancestor for this class.

superClass is the name of the class from which this class derives. It does not need to be fully qualified.

JotType Interface

The `com.borland.jbuilder.jot.JotType` interface represents a type identifier within JOT. This may be a class, a primitive type, or an array.

Its methods are:

```
public String getFullName();
```

Obtain the full name of this type.

```
public JotClass getJotClass();
```

Retrieve a reference to the JOT class that this type represents, or `null` if it is not primitive or an array and has not been compiled.

```
public JotClassSource getJotClassSource();
```

Get the JOT source object for this type, or `null` if it is a primitive type or an array.

```
public String getName();
```

Return the full name of this type.

```
public void setName(String value) throws
    IllegalAccessException;
```

Update the name of this type.

value is the new name.

JotMethod Interface

From the `JotClass` object you can locate existing methods within the class. Each is an instance of the `com.borland.jbuilder.jot.JotMethod` interface, which provides access to details about each method.

The methods of this interface are:

```
public JotClass getDeclaringClass();
```

Obtain a reference to the class that contains this method.

```
public JotParameter getFirstParameter();
```

Retrieve information about the first parameter of this method, or `null` if it has none. The returned class is not covered in this book.

```
public int getModifiers();
```

Determine the modifiers for this method: its visibility and its abstract, final, and static status. Use the constants or methods from the `java.lang.reflect.Modifier` class to query these settings.

```
public String getName();
```

Return the name of this method.

```
public JotParameter getParameter(String name);
```

Locate a particular parameter and return its details, or `null` if it does not exist.

name is the name of the parameter to find.

```
public JotParameter[] getParameters();
```

Discover further details about the parameters that this method accepts. If it takes none, an empty array results.

```
public JotClass[] getParameterTypes();
```

Find out the types of parameters that this method accepts. An empty array is returned if it takes no parameters.

```
public JotType getReturnType();
```

Obtain a reference to the type of the method's return value.

```
public JotType[] getThrowSpecifiers();
```

Retrieve a list of the exception types that this method declares that it throws, or an empty array if it does not define any.

The `JotConstructor` interface derives from this one, returning a name of “<init>” and adding the ability to create a new instance of the class.

JotMethodSource Interface

When the source for a class is available you may be able to update its methods through the `com.borland.jbuilder.jot.JotMethodSource` interface. Check your reference to the `JotMethod` and cast it down to this interface if available.

```
if (jmethod instanceof JotMethodSource) then
    JotMethodSource jmethodSource = (JotMethodSource)jmethod;
```

The extended abilities of this interface are:

```
public JotParameterDeclaration addParameter(JotMarker
    marker, boolean before, String type, String name);
```

Add a new parameter to this method and return a reference to it.

`marker` is a reference to another parameter in the method to use as a base for positioning this parameter. If `null`, it is added at the end of the parameter list.

`before` is true to insert the parameter before the marker above, or false to place it after the marker.

`type` is the name of the type for this parameter. It does not have to be a fully qualified name.

`name` is the name of the new parameter.

```
public JotType addThrowSpecifier(JotMarker marker, boolean
    before, String type);
```

Include a new exception type in the method's `throws` list and return a reference to its type.

`marker` and `before` are the same as above.

`type` is the name of the exception class. It does not have to be a fully qualified name.

```
public JotCodeBlock getCodeBlock();
```

Gain access to the body of the method through this method. It returns `null` if the method is abstract. Get a list of the methods statements with the `getStatements` method instead.

```
public int getDeclaredModifiers();
```

Retrieve the modifiers for this method actually declared in this file: its visibility and its abstract, final, and static status. Use the constants or methods from the `java.lang.reflect.Modifier` class to query these settings.

```
public JotStatement[] getStatements();
```

Obtain a list of the statements that make up the body of this method. An empty array results if the method is abstract or currently has an empty body. Add new statements through the object returned by the `getCodeBlock` method.

```
public void removeParameter(JotParameter parameter);
```

Delete the given item from this method.

```
public void removeThrowSpecifier(JotType spec);
```

`parameter` and `spec` are references to the item to remove.

```
public void setModifiers(int modifiers);
```

Update the modifiers for this method – its visibility and whether it is abstract, final, and/or static.

`modifiers` is the collection of modifiers for this method. Use the constants from the `java.lang.reflect.Modifier` class to build the value, OR-ing them together.

```
public void setName(String name);
```

Modify the name of this method.

`name` is the new name.

```
public void setParameterText(String parameters);
```

Set the entire parameter collection for this method as one string.

`parameters` is the list of parameters, like “`JotMarker marker, boolean before`”.

```
public void setReturnType(String type);
```

Alter the return type of this method.

`type` is the new return type, such as “`void`” or “`String`”. It does not have to be a fully qualified name.

As for `JotMethod` and `JotConstructor`, the `JotConstructorSource` interface derives from this one, adding its own special abilities.

JotCodeBlock Interface

The body of a method, constructor, or initialization block is represented by an instance of the `com.borland.jbuilder.jot.JotCodeBlock` interface. It extends `JotStatement`, which is described below, and allows you to add to, remove from, or alter the statements that make up the block.

Its methods are:

```
public JotAssignment addAssignment(JotMarker marker,
    boolean before, String variableName, String value);
public JotDo addDoStatement(JotMarker marker, boolean
    before, String condition);
public JotFor addForStatement(JotMarker marker, boolean
    before, String condition);
```

```

public JotIf addIfStatement(JotMarker marker, boolean
    before, String condition);
public JotIf addIfStatement(JotMarker marker, boolean
    before, String condition, boolean hasElse);
public JotInnerClass addInnerClass(JotMarker marker,
    boolean before, String className);
public JotMethodCall addMethodCall(JotMarker marker,
    boolean before, String methodName, String parameters);
public JotReturn addReturnStatement(JotMarker marker,
    boolean before, String expression);
public JotStatement addStatement(JotMarker marker, boolean
    before, String text);
public JotStatement addStatement(JotMarker marker, boolean
    before, boolean semi, String text);
public JotTry addTryStatement(JotMarker marker, boolean
    before, String type, String variableName);
public JotTry addTryStatement(JotMarker marker, boolean
    before, String[] types, boolean hasFinally);
public JotVariableDeclaration
    addVariableDeclaration(JotMarker marker, boolean before,
        String variableType, String variableName);
public JotWhile addWhileStatement(JotMarker marker, boolean
    before, String condition);

```

Add a statement of the specified type to this block. Note that these statements appear at the top-most level within the block, and may have their own sub-statements.

`marker` is a reference to another item in the class to use as a base for positioning this statement. If `null`, the statement is added at the end of the class.

`before` is true to insert the constructor before the marker above, or false to place it after the marker.

`className` is the name of the inner class to define.

`condition` is the text for the condition to evaluate for an `if` statement or a loop. In a `for` statement, this text includes the loop initialization and step expression, such as `"int index = 0; index < a.length; index++"`.

`expression` is the text for the value to return from this method. Use an empty string to return nothing.

`hasElse` is true if the `if` statement has an `else` part, or false (the default) if it does not.

`hasFinally` is true if the `try` statement has a `finally` clause, or false if it does not.

`methodName` is the name of the method to invoke. It may include an object reference and/or nested calls, such as `"text.substring(3, 6).toUpperCase"`.

`parameters` is the list of parameters for a method call expressed as text, such as `"3, 6"`. Pass `null` if there are no parameters.

`semi` is true to add a terminating semicolon to the statement, or false to leave it as is.

`text` is the entire statement as a string. A terminating semicolon is added as necessary.

`type` is the name of the exception being caught by the `try` statement. It does not have to be a fully qualified name. This version of the method creates a `try` with a single `catch` clause and no `finally` clause.

`types` is a list of the names of exceptions to catch in this `try` statement. They do not have to be fully qualified names. If an empty array is provided then the statement has no `catch` clauses.

`value` is the text for the value to assign to a variable.

`variableName` is the name of the variable in the statement.

`variableType` is the type of the variable being declared. If this is a class, it does not have to be a fully qualified name.

```
public JotAssignment[] getAssignments();
```

Find all the assignment statements in this code block and return them, or an empty array if there are none.

```
public int getComparableLocation(JotMarker marker);
```

Convert marker locations into integer values that can be compared for order. The marker with the lower integer value occurs earlier in the file.

`marker` is the item from the file to locate.

```
public JotClass[] getDeclaredInnerClasses();
```

Locate all the inner classes declared in this block and return them, or an empty array if none are defined.

```
public JotMethodCall getMethodCall(String methodName);
```

```
public JotMethodCall[] getMethodCalls();
```

```
public JotMethodCall[] getMethodCalls(String methodName);
```

Get a list of one of all of the method calls in this code block. A `null` or an empty array is returned if there are no matching method calls. For nested method calls, like `"text.substring(3, 6).toUpperCase()"`, only one entry appears and refers to the outermost call.

`methodName` is the name of a particular method to locate. In the version that returns a single object, only the first matching call (the outermost one in a nested call) is retrieved.

```
public JotStatement[] getStatements();
```

Obtain a list of all the statements in this block, or an empty array if there are none. The list does not include sub-statements within another statement, such as the body of a loop.

```
public JotVariableDeclaration getVariableDeclaration(String variableName);
```

```
public JotVariableDeclaration[] getVariableDeclarations();
```

Find one or all of the variable declarations in this code block. A `null` or an empty array is returned if the declarations do not exist.

`variableName` is the name of the variable to locate.

```
public void removeAssignment(JotAssignment assignment);
```

```
public void removeInnerClass(JotInnerClass inner);
```

```
public void removeMethodCall(JotMethodCall methodCall);
```

```
public void removeStatement(JotStatement statement);
```

```
public void removeVariableDeclaration(JotVariableDeclaration declaration);
```

Delete the specified item from this code block. Nothing happens if the item does not belong to this block.

assignment, inner, methodCall, statement, and declaration are references to the item to be removed.

The `JotInitBlock` interface represents the initialization block for a class and derives from this one, recording the modifiers for the block.

JotSourceElement Interface

The `com.borland.jbuilder.jot.JotSourceElement` interface is the base for all elements derived from a source file. If JOT is working with a class file then this interface is not available.

Its methods are described below:

```
public int getIndentLevel();
```

Obtain the logical indentation level for this element, with zero being against the left margin.

```
public JotSourceElement getParent();
```

Find the parent element for this one. For example, an expression element may be part of an if statement or a loop.

```
public String getText();
```

Retrieve the text for this element. It includes text for all sub-elements as well, such as the entire code for a method.

```
public boolean isModified();
```

Returns true if the text for this element has been altered since it was last parsed, or false if it is still the same.

```
public void setModified(boolean modified);
```

Mark the element as having changed with this method.
modified is true to note a change, or false to ignore it.

```
public void setText(String text);
```

Update the entire contents of this element by setting its text value. Any existing sub-elements are discarded and new ones are created that correspond to the altered text. The modified flag for this element is also set, forcing a re-parse when `getText` is next called.

text is the new string version of this element and its contents.

The `JotComment`, `JotExpression`, and `JotStatement` interfaces listed below also implement `JotSourceElement` since they are all part of the contents of a source file.

JotCommentable Interface

Comments may only be applied to elements that implement the `com.borland.jbuilder.jot.JotCommentable` interface. These items include `JotSourceFile`, `JotClassSource`, and `JotCodeBlock`.

The available methods are:

```
public void addBlankLine(JotMarker marker, boolean before);
```

Add a blank line to the file.

marker is a reference to another item in the class to use as a base for positioning this line. If null, the line is added at the end of the file.

`before` is true to insert the line before the marker above, or false to place it after the marker.

```
public JotComment addComment(JotMarker marker, boolean
    before, int type, String text);
```

Insert a new comment into the file and return a reference to it. Delimiters appropriate to the type of comment are added to the supplied text. If the comment is multi-line or a Javadoc comment and it contains newline characters, these are preceded by an asterisk.

`marker` and `before` are the same as above.

`type` is one of the constant values defined in the `JotComment` class.

`text` is the text of the comment.

```
public JotComment getComment(JotMarker marker, boolean
    before);
```

Retrieve the comment before or after a particular element in the file, or `null` if there is no comment there.

`marker` and `before` are the same as above.

```
public void removeComment(JotComment comment);
```

Delete a comment from the file. Nothing happens if the comment is not in this file.

`comment` is a reference to the comment to remove.

JotComment Interface

Create or read comments within a file through the `com.borland.jbuilder.jot.JotComment` interface. Comments are only available if the file is based on source code.

The methods of this interface are:

```
public String getCommentText();
```

Get the entire text of the comment through this method, but excluding any comments tags (starting with “@”). A `null` is returned if no text is found.

```
public String getNormalText();
```

Although this method should return the text for a Javadoc comment, it always returns `null`.

```
public String getSummaryText();
```

Retrieve the first sentence from a comment – up to the first period (.) – or `null` if there is no text.

```
public int getType();
```

Discover the type of this comment. It returns one of the constant values listed below.

The following constants define the type of comment:

```
public static final int BLOCK;
```

```
public static final int DOC;
```

```
public static final int LINE;
```

```
public static final int NONE;
```

These represent a standard multi-line comment, a Javadoc comment, a single-line comment, and an unknown type.

JotExpression Interface

Expressions within the code are represented by an instance of the `com.borland.jbuilder.jot.JotExpression` interface. These items appear within several statement types including `if` statements, loops, and assignment statements. From an `if` statement you would access it with the following code:

```
JotExpression jexpr = jif.getCondition();
```

For an assignment you would use:

```
JotExpression jexpr = jassign.getRHS();
```

An expression may be a literal value, such as `1`, `true`, or “Yes”, a reference to a variable or object, like `index` or `acct.getBalance()`, or a calculation, such as `width / 2`, among others.

The abilities of this interface are:

```
public JotAssignment getAssignment();
public JotCondition getCondition();
public JotMethodCall getMethodCall();
public JotNew getNew();
public JotExpression getOperation();
public JotValue getValue() throws IllegalArgumentException;
public JotVariable getVariable();
```

Obtain an element that more fully describes the expression and its contents, or a `null` if the expression is not of the requested format.

```
public boolean isConstant();
```

Returns `true` if this expression is a constant value, or `false` if it is not.

```
public boolean isNull();
```

Returns `true` if the text of this expression is “`null`”, or `false` if it is anything else. Even though an expression may result in `null` as the outcome of some other calculation, it is not evaluated to determine this, and so returns `false`.

Sub-interfaces of this one provide greater detail about each type of expression. Instances of these are returned by the various methods in this one and are shown in Table 24-1.

Table 24-1. JotExpression descendents

Interface	Purpose
<code>JotAssignment</code>	An assignment to a variable.
<code>JotBinaryExpression</code>	An expression that contains a binary operator, including assignments.
<code>JotCondition</code>	An expression representing the conditional operator (<code>? :</code>).
<code>JotInitializer</code>	A set of initialization expressions for an array.
<code>JotMethodCall</code>	A call to a method, providing the full method name and any arguments.
<code>JotNew</code>	The instantiation of a new object, including any arguments.
<code>JotSubscript</code>	An item within an array.

Interface	Purpose
JotTypeop	A type operation such as a cast or an instanceof expression.
JotUnaryExpression	An expression that contains a unary operator.

JotStatement Interface

Defining the base abilities of all statements, the `com.borland.jbuilder.jot.JotStatement` interface is subclassed into specialized interfaces for each type of statement.

The common functionality is:

```
public JotCodeBlock getCodeBlock();
```

Retrieve the code block that represents this statement, allowing you to add and remove statements from it.

```
public JotStatement[] getStatements();
```

Obtain a list of the sub-statements that make up the body of this statement, or an empty array if there is no body.

Many interfaces derive from this basic one, corresponding to each of the statement types possible in Java, as shown in Table 24-2.

Table 24-2. JotStatement descendents

Interface	Purpose
JotBreak	A <code>break</code> out of a loop. Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new breaks.
JotCase	A <code>case</code> label within a <code>switch</code> statement. Retrieve these from the <code>JotSwitch.getCases</code> method.
JotCatch	A <code>catch</code> block from a <code>try</code> statement. Access these via the <code>getCatch</code> or <code>getCatches</code> methods of <code>JotTry</code> .
JotCodeBlock	A block of code that is the body of another statement or method. Retrieve it through the <code>getCodeBlock</code> method of <code>JotStatement</code> , although its exact meaning changes from one statement type to another.
JotContinue	A <code>continue</code> statement. Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new continues.
JotDefault	The default label within a <code>switch</code> statement. Retrieve it from the <code>JotSwitch.getDefault</code> method.
JotDo	A <code>do</code> loop. Create new <code>do</code> loops with the <code>addDoStatement</code> method of <code>JotCodeBlock</code> .

Interface	Purpose
JotExpressionStatement	A statement that consists of a single expression, such as an assignment or a method call. Use the <code>addAssignment</code> or <code>addMethodCall</code> methods of <code>JotCodeBlock</code> to create these types of statements.
JotFieldDeclaration	The declaration of a field within a class or interface. The <code>addField</code> method of <code>JotClassSource</code> generates new declarations for you.
JotFinally	The <code>finally</code> clause of a <code>try</code> statement. Access it via the <code>JotTry.getFinally</code> method.
JotFor	A <code>for</code> loop. Create new <code>for</code> loops with the <code>addForStatement</code> method of <code>JotCodeBlock</code> .
JotIf	An <code>if</code> statement, and optional <code>else</code> clause. Use the <code>addIfStatement</code> methods of <code>JotCodeBlock</code> to add these to your code.
JotInitBlock	A class or instance initialization block. The <code>addInitBlock</code> method of <code>JotClassSource</code> creates new blocks for you.
JotLabelled	A labeled statement – the destination for a <code>break</code> or <code>continue</code> . Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new labels.
JotReturn	A <code>return</code> from a method, including an optional return value. Create a new <code>return</code> statement with the <code>addReturnStatement</code> method of <code>JotCodeBlock</code> .
JotSwitch	A <code>switch</code> statement, providing access to the expression it evaluates and the <code>case</code> and <code>default</code> clauses it refers to. Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new <code>switch</code> statements.
JotSynchronized	A <code>synchronized</code> statement. Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new <code>synchronized</code> sections.
JotThrow	A <code>throw</code> statement. Use the generic <code>addStatement</code> methods of <code>JotCodeBlock</code> for new throws.
JotTry	A <code>try</code> statement, providing access to any <code>catch</code> clauses and the optional <code>finally</code> clause. See the <code>addTryStatement</code> methods of <code>JotCodeBlock</code> to create these.
JotVariableDeclaration	The declaration of a local variable. Use the <code>addVariableDeclaration</code> method of <code>JotCodeBlock</code> to create a new declaration.
JotWhile	A <code>while</code> loop. Create new <code>while</code> loops with the <code>addWhileStatement</code> method of <code>JotCodeBlock</code> .

JSPTagWizard Example

Complementing the Tag Library Descriptor wizard from the last chapter, the JSP Tag wizard generates Java classes that support custom JSP tags. It is placed in the Object Gallery since it creates a new file for the current project and appears on the Web tab.

When invoked, the wizard displays a single-page dialog that asks for the version of JSP to use (1.1, 1.2, or 2.0), the name of the package and class to create, then what functionality the tag needs to support, and what attributes it accepts. Figure 24-2 shows the wizard in action.

Figure 24-2. The JSP Tag wizard.



Firstly you have to establish the wizard that utilizes JOT to generate the code. The `JSPTagWizard` class shown in Listing 24-1 does this. It defines a static field for the `WizardAction` (placing it in the Object Gallery (true) on the Web tab), and registers this with the wizard manager in the OpenTools initialization routine.

Listing 24-1. Wizard support for generating JSP tag classes.

```
package wood.keith.opentools.wizards.jsptags;

import java.io.File;
import java.util.ResourceBundle;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

import com.borland.jbuilder.node.JBProject;
import com.borland.jbuilder.paths.ProjectPathSet;
import com.borland.jbuilder.personality.WebAppPersonality;
import com.borland.primetime.PrimeTime;
import com.borland.primetime.help.HelpManager;
```

```

import com.borland.primetime.help.ZipHelpTopic;
import com.borland.primetime.ide.Browser;
import com.borland.primetime.node.FileNode;
import com.borland.primetime.personality.Personality;
import com.borland.primetime.util.VetoException;
import com.borland.primetime.vfs.Url;
import com.borland.primetime.wizard.BasicWizard;
import com.borland.primetime.wizard.Wizard;
import com.borland.primetime.wizard.WizardAction;
import com.borland.primetime.wizard.WizardHost;
import com.borland.primetime.wizard.WizardManager;
import com.borland.primetime.wizard.WizardPage;

/**
 * This class defines the "JSP Tag" wizard at the highest level.
 * It sets the dialog pages to be displayed and generates the
 * JSP tag class as defined by user input.
 *
 * @author Keith Wood
 * @version 1.0 9 November 2000
 * @version 2.0 6 February 2002 - JSP 1.2
 * @version 3.0 10 February 2004 - JSP 2.0
 */
public class JSPTagWizard extends BasicWizard {

    private static final ResourceBundle res = Res.getResource();
    private static final String VERSION = "3.0";

    public static final String TITLE = res.getString("JSPWizardTitle");

    // Internal variables
    private JSPTagWizardPage _tagPage;

    /**
     * Register the "JSP Tag" tool.
     *
     * Provides the needed OpenTools interface required to register the
     * WizardAction which defines and creates this wizard.
     *
     * @param majorVersion the major version of the current OpenTools API
     * @param minorVersion the minor version of the current OpenTools API
     */
    public static void initOpenTool(byte majorVersion, byte minorVersion) {
        if (majorVersion != PrimeTime.CURRENT_MAJOR_VERSION) {
            return;
        }
        WizardManager.registerWizardAction(WIZARD_JSPTag);
        if (PrimeTime.isVerbose()) {
            System.out.println("Loaded JSP Tag wizard v" + VERSION);
            System.out.println("Written by Keith Wood (kwood@iprimus.com.au)");
        }
    }

    /**
     * Definition of "JSP Tag" WizardAction.
     *
     * This is an OpenTools registered action defining where the wizard
     * will appear in the IDE and a factory for creating each instance
     * of this wizard.
     */
    public static final WizardAction WIZARD_JSPTag = new WizardAction(
        res.getString("JSPWizardName"),
        res.getString("JSPWizardChar").charAt(0),
        res.getString("JSPWizardTip"),
        new ImageIcon(ClassLoader.getResource(
            "wood/keith/opentools/wizards/jsptags/jsptag16.gif")),
        new ImageIcon(ClassLoader.getResource(
            "wood/keith/opentools/wizards/jsptags/jsptag32.gif")), true,
        res.getString("ObjectGalleryPage")) {

```

```

// Override method so wizard is disabled if no active project.
public void update(Object source) {
    Browser browser = Browser.findBrowser(source);
    setEnabled((browser != null) &&
        (browser.getActiveProject() != null));
}

protected Wizard createWizard() {
    return new JSPTagWizard();
}

/**
 * JBuilder 10+ - only display for the Web app personality.
 *
 * @return the list of personalities to which this action applies
 */
public Personality[] getPersonalities() {
    return new Personality[] {WebAppPersonality.webAppPersonality};
}
};

/**
 * Wizard startup.
 *
 * "JSP Tag" wizard is to become visible,
 * create and order the wizard pages to be displayed.
 *
 * @param host the WizardHost that owns this wizard instance.
 * @return the initial WizardPage to show.
 */
public WizardPage invokeWizard(WizardHost host) {
    setWizardTitle(TITLE);
    _tagPage = new JSPTagWizardPage(
        host.getBrowser().getProjectView().getActiveProject());
    addWizardPage(_tagPage);
    return super.invokeWizard(host);
}

/**
 * Assemble file Url.
 *
 * From the given parameters generates a fully qualified location
 * where the "JSP Tag" file will be written.
 * This is returned in the form of a Url.
 *
 * @param project the JProject in which the node is created.
 * @param packageName the name of the package in which file
 * is created.
 * @param fileName the name of the file to be created.
 * @return the Url which was created.
 */
protected static Url makeFileUrl(JProject project, String packageName,
    String fileName) {
    try {
        ProjectPathSet paths = project.getPaths();
        Url[] sourcePaths = paths.getSourcePath();
        String projectPath = "";

        if (sourcePaths.length > 0) {
            projectPath = sourcePaths[0].getFile();
        }

        String dirName = projectPath;
        if (packageName != null && packageName.length() > 0) {
            dirName = dirName + '/' + packageName.replace('.', '/');
        }
        String filePath = dirName + '/' + fileName;
        File newFile = new File(filePath);
        return new Url(newFile);
    }
}

```

```

        catch (Exception ex) {
            ex.printStackTrace();
            return null;
        }
    }

    /**
     * Creates/replaces file node.
     *
     * From the given parameters create/replace a project file node for the
     * "JSP Tag" file to be written.
     *
     * @param project      the JBProject in which the node is created.
     * @param packageName  the name of the package in which file
     *                     is created.
     * @param fileName     the name of the file to be created.
     * @return the FileNode which was created.
     */
    protected static FileNode createNode(JBProject project,
        String packageName, String fileName) {
        Url filePath = makeFileUrl(project, packageName, fileName);
        // getNode() will create a node if does not exist
        FileNode newNode = project.getNode(filePath);
        newNode.setParent(project);
        return newNode;
    }

    /**
     * Perform code generation.
     *
     * Produces the "JSP Tag" file as defined by user input on the
     * wizard pages. It is added as a node to the currently active project
     * and replaces any previously generated file of the same name.
     *
     * @throws VetoException if unable to finish
     */
    protected void finish() throws VetoException {
        Browser browser = wizardHost.getBrowser();
        JBProject project =
            (JBProject)browser.getProjectView().getActiveProject();
        // Save selected JSP version for next time
        JSPTagProperties.JSP_VERSION.setValue(_tagPage.getJSPVersion());
        FileNode javaNode = null;
        try {
            if (_tagPage.isExtraInfo()) {
                // Create the extra info source file
                javaNode = createNode(project, _tagPage.getPackageName(),
                    _tagPage.getClassName() + "ExtraInfo.java");
                new JSPTagExtraGenerator().writeSource(project,
                    javaNode, _tagPage.getPackageName(),
                    _tagPage.getClassName() + "ExtraInfo",
                    _tagPage.getJSPVersion());
            }
            // Create the tag source file
            javaNode = createNode(project, _tagPage.getPackageName(),
                _tagPage.getClassName() + ".java");
            new JSPTagGenerator().writeSource(project, javaNode,
                _tagPage.getPackageName(), _tagPage.getClassName(),
                _tagPage.getJSPVersion(), _tagPage.isBodyAltered(),
                _tagPage.isRepeated(), _tagPage.isEndUsed(),
                _tagPage.isNested(),
                _tagPage.isTryCatchFinally(), _tagPage.isSimpleTag(),
                _tagPage.isDynamicAttributes(), _tagPage.getProps());
        }
        catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(wizardHost.getBrowser(),
                res.getString("GenerationError") + "\n" +
                ex.getClass().getName() + "\n" + ex.getMessage(),
                wizardTitle, JOptionPane.ERROR_MESSAGE);
        }
    }

```

```

    }
    try {
        // Open the generated/updated file in the Content Pane.
        browser.setActiveNode(javaNode, true);
    }
    catch (Exception ex) {
        // Ignore
    }
}

/**
 * Wizard termination.
 *
 * The Wizard has been completed (finished or cancelled) and any
 * resource cleanup to help the garbage collector can occur here.
 */
public void wizardCompleted() {
}

/**
 * Provide help for the wizard.
 *
 * @param page the current page in the wizard
 * @param host the wizard host
 */
public void help(WizardPage page, WizardHost host) {
    HelpManager.showHelp(
        new ZipHelpTopic(null, ClassLoader.getResource(
            getClass().getName().replace('.', '/') + ".html").toString()),
        page.getPageComponent(host));
}
}

```

Starting the wizard calls the `invokeWizard` method, which creates and adds the single page to the dialog. There is no cleanup necessary in the `wizardCompleted` method, while the `help` method returns an appropriate help topic for display.

Most of the wizard activity takes place in its `finish` method. It starts by creating a new Java source node within the current project, using the package and class name supplied by the user to correctly name and position the file. The `createNode` and `makeFileUrl` methods assist in this task. After generating the file's contents, the new node is opened in the Content Pane and made active.

Generating a Java Class

Based on the user's selections, a Java class is created using JOT. The `writeSource` method of the `JspTagGenerator` class operates on the new node to fill it with code (see Listing 24-2). First it retrieves all the information entered by the user into the wizard's UI and stores them in local variables. Then it accesses the new file through the package manager and starts adding code. Following the `import` statements, a new class is created within the file and various helper methods supply its contents. Finally, the changes are committed to the source file in memory, and then saved out to disk. Remember to free up any resources used by JOT before returning.

Listing 24-2. Generating code with JOT.

```

package wood.keith.opentools.wizards.jsptags;

import java.io.IOException;
import java.lang.reflect.Modifier;
import java.text.DateFormat;

```

```

import java.text.MessageFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.ResourceBundle;
import java.util.SortedMap;

import com.borland.jbuilder.jot.JotClassSource;
import com.borland.jbuilder.jot.JotCodeBlock;
import com.borland.jbuilder.jot.JotComment;
import com.borland.jbuilder.jot.JotFieldDeclaration;
import com.borland.jbuilder.jot.JotIf;
import com.borland.jbuilder.jot.JotMethodSource;
import com.borland.jbuilder.jot.JotPackages;
import com.borland.jbuilder.jot.JotSourceFile;
import com.borland.jbuilder.jot.JotTry;
import com.borland.jbuilder.jot.JotVariableDeclaration;
import com.borland.jbuilder.jot.JotWhile;
import com.borland.jbuilder.node.JBProject;
import com.borland.primetime.node.FileNode;

/**
 * Generate the JSP tag class.
 *
 * @author Keith Wood
 * @version 1.0 9 November 2000
 * @version 2.0 6 February 2002 - added JSP 1.2 support
 * @version 3.0 10 February 2004 - added JSP 2.0 support
 */
public class JSPTagGenerator {

    private static final ResourceBundle res = Res.getResource();
    // Meaningful names for code generation
    private static final boolean AFTER = false;
    private static final boolean BEFORE = true;

    private static final String[] OUTER_TAG = new String[] {"OuterTag"};

    /**
     * This is the "guts" of the wizard. This is where all the work
     * is done to create the source file for the tag class.
     *
     * @param project the JBuilder project
     * @param file the file node to be written to
     * @param packageName the name of the new class' package
     * @param className the name of the new class
     * @param jspVersion the version of JSP in use
     * @param isBodyAltered true if body content is altered,
     * false if not
     * @param isRepeated true if body is repeated, false if not
     * @param isEndUsed true if end tag generated, false if not
     * @param isNested true if tag expects to be nested,
     * false if not
     * @param isTryCatchFinally true if implementing TryCatchFinally,
     * false if not
     * @param isSimpleTag true if a JSP 2.0 simple tag, false if not
     * @param isDynamicAttrs true if uses dynamic attributes,
     * false if not
     * @param props details about the tag attributes
     */
    public void writeSource(JBProject project, FileNode file,
        String packageName, String className, String jspVersion,
        boolean isBodyAltered, boolean isRepeated, boolean isEndUsed,
        boolean isNested, boolean isTryCatchFinally, boolean isSimpleTag,
        boolean isDynamicAttrs, SortedMap props) {
        boolean hasProperties = false;
        boolean hasDateAttribute = false;

        Iterator names = props.keySet().iterator();
        while (names.hasNext()) {
            hasProperties = true;

```

```

        if (props.get(names.next()).equals("Date")) {
            hasDateAttribute = true;
            break;
        }
    }
    names = null;

    // Access the new file
    JotPackages jpackages = project.getJotPackages();
    JotSourceFile jsource = jpackages.getSourceFile(file.getUrl());
    // Add the package statement and imports to the JotSourceFile
    if (packageName.length() > 0) {
        jsource.setPackage(packageName);
    }
    jsource.addImport("java.io.*");
    if (hasDateAttribute) {
        jsource.addImport("java.util.*");
    }
    jsource.addImport("javax.servlet.jsp.*");
    jsource.addImport("javax.servlet.jsp.tagext.*");

    // Add the class declaration
    JotClassSource jclass = createClass(jsource, className, jspVersion,
        isEndUsed, isNested, isRepeated, isBodyAltered, isTryCatchFinally,
        isSimpleTag, isDynamicAttrs);
    if (hasProperties || isNested) {
        // Add internal fields
        addInternalFields(jclass, props, isNested);
    }
    // Add attribute setters
    addAttributeSetters(jclass, props);
    if (isDynamicAttrs) {
        // Add DynamicAttributes method setDynamicAttribute
        addSetDynamicAttributeMethod(jclass);
    }
    if (isSimpleTag) {
        // Add SimpleTag method doTag override
        addDoTag(jclass, isNested, isRepeated, isBodyAltered,
            hasProperties);
    }
    else {
        // Add Tag method doStartTag override
        addDoStartTag(jclass, jspVersion, isEndUsed, isNested, isRepeated,
            isBodyAltered, hasProperties);
        if (isBodyAltered || isRepeated) {
            // Add BodyTag method doInitBody override
            addDoInitBody(jclass);
            // Add BodyTag method doAfterBody override
            addDoAfterBody(jclass, jspVersion, isRepeated);
        }
        if (isEndUsed) {
            // Add Tag method doEndTag override
            addDoEndTag(jclass);
        }
    }
    if (isTryCatchFinally) {
        // Add TryCatchFinally methods doCatch and doFinally
        addTryCatchFinallyMethods(jclass);
    }

    // Save the changes
    jpackages.commit(jsource);

    // Write the changes out
    try {
        file.save();
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}

```

```

    // Free up resources used
    jpackages.release(jsource);
}

/**
 * Add the class declaration.
 *
 * @param jsource      the source file being generated
 * @param className    the name of the new class
 * @param jspVersion   the version of JSP in use
 * @param isEndUsed    true if content is written after the body,
 *                    false if none
 * @param isNested     true if the tag looks for an enclosing
 *                    tag, false if standalone
 * @param isRepeated   true if the tag iterates over its body,
 *                    false if it processes it just once
 *                    (if at all)
 * @param isBodyAltered true if the tag changes the body content,
 *                    false if it leaves it unchanged
 * @param isTryCatchFinally true if the tag deals with its own errors
 *                    during processing, false if it does not
 * @param isSimpleTag  true if tag is a JSP 2.0 simple tag,
 *                    false if not
 * @param isDynamicAttrs true if tag accepts unspecified
 *                    attributes, false if not
 * @return the class being generated
 */
private JotClassSource createClass(JotSourceFile jsource,
    String className, String jspVersion, boolean isEndUsed,
    boolean isNested, boolean isRepeated, boolean isBodyAltered,
    boolean isTryCatchFinally, boolean isSimpleTag,
    boolean isDynamicAttrs) {
    JotClassSource jclass =
        jsource.addClass(null, AFTER, className, false);
    StringBuffer comment = new StringBuffer();
    comment.append(isEndUsed ? res.getString("TagIsEndUsed") : "").
        append(isNested ? res.getString("TagIsNested") : "").
        append(isRepeated ? res.getString("TagIsRepeated") : "").
        append(isBodyAltered ? res.getString("TagIsBodyAltered") : "").
        append(isTryCatchFinally ? res.getString("TagIsTryCatchFinally") :
            "").append(isSimpleTag ? res.getString("TagIsSimpleTag") : "").
        append(isDynamicAttrs ? res.getString("TagIsDynamicAttrs") : "");
    jsource.addComment(jclass, BEFORE, JotComment.DOC,
        MessageFormat.format(res.getString("JSPTagClassHeader"),
            new Object[] {jspVersion}) + (comment.length() == 0 ? "" :
                " " + res.getString("JSPTagClassHeaderCharacteristics") +
                comment.toString()) +
            ".\n\n@author JSP Tag Wizard\n@version 1.0 " +
            DateFormat.getDateInstance(DateFormat.LONG).format(new Date()));
    jclass.setModifiers(Modifier.PUBLIC);
    jclass.setSuperclass(isSimpleTag ? "SimpleTagSupport" :
        (isBodyAltered || isRepeated ? "BodyTagSupport" : "TagSupport"));
    if (isDynamicAttrs) {
        jclass.addInterface(null, AFTER, "DynamicAttributes");
    }
    if (isTryCatchFinally) {
        jclass.addInterface(null, AFTER, "TryCatchFinally");
    }
    return jclass;
}

/**
 * Add internal fields for the tag class.
 *
 * @param jclass the class being generated
 * @param props  the names and types of the properties for this tag
 * @param isNested true if the tag looks for an enclosing tag,
 *                false if standalone
 */
private void addInternalFields(JotClassSource jclass, SortedMap props,

```

```

        boolean isNested) {
    JotComment jcomment = jclass.addComment(null, AFTER, JotComment.LINE,
        " " + res.getString("InternalFields"));
    jclass.addBlankLine(jcomment, BEFORE);
    if (props.size() > 0) {
        jclass.addComment(null, AFTER, JotComment.DOC,
            " @todo " + res.getString("InternalFieldsTodo") + " ");
    }
    Iterator names = props.keySet().iterator();
    while (names.hasNext()) {
        String name = (String)names.next();
        JotFieldDeclaration jfield = jclass.addField(
            null, AFTER, (String)props.get(name), "_" + name);
        jfield.setModifiers(Modifier.PRIVATE);
    }
    if (isNested) {
        jclass.addComment(null, AFTER, JotComment.DOC,
            " @todo " + MessageFormat.format(res.getString("OuterTagTodo"),
                OUTER_TAG) + " ");
        JotFieldDeclaration jfield =
            jclass.addField(null, AFTER, "OuterTag", "_enclosingTag");
        jfield.setModifiers(Modifier.PRIVATE);
        jfield.setInitializer("null");
    }
}

/**
 * Add attribute setters for the tag.
 *
 * @param jclass the class being generated
 * @param props the names and types of the properties for this tag
 */
private void addAttributeSetters(JotClassSource jclass,
    SortedMap props) {
    Iterator names = props.keySet().iterator();
    while (names.hasNext()) {
        String name = (String)names.next();
        JotMethodSource jmethod = jclass.addMethod(null, AFTER, "void",
            "set" + name.substring(0, 1).toUpperCase() + name.substring(1));
        JotComment jcomment = jclass.addComment(
            jmethod, BEFORE, JotComment.DOC,
            MessageFormat.format(res.getString("AttributeSetterMethod"),
                new Object[] {name}) + "\n\n@param value " +
                res.getString("ValueParam"));
        jclass.addBlankLine(jcomment, BEFORE);
        jmethod.setModifiers(Modifier.PUBLIC);
        jmethod.addParameter(null, AFTER, (String)props.get(name), "value");
        jmethod.getCodeBlock().
            addAssignment(null, AFTER, "_" + name, "value");
    }
}

/**
 * Add DynamicAttributes method setDynamicAttribute.
 *
 * @param jclass the class being generated
 */
private void addSetDynamicAttributeMethod(JotClassSource jclass) {
    JotMethodSource jmethod =
        jclass.addMethod(null, AFTER, "void", "setDynamicAttribute");
    JotComment jcomment =
        jclass.addComment(jmethod, BEFORE, JotComment.DOC,
            res.getString("SetDynamicAttributeMethod") +
            "\n\n@param uri " +
            res.getString("UriParam") + "\n\n@param localName " +
            res.getString("LocalNameParam") + "\n\n@param value " +
            res.getString("ValueParam") + "\n\n@throws JspException " +
            res.getString("SetDynamicAttributeJspException"));
    jclass.addBlankLine(jcomment, BEFORE);
    jmethod.setModifiers(Modifier.PUBLIC);
}

```

```

jmethod.addParameter(null, AFTER, "String", "uri");
jmethod.addParameter(null, AFTER, "String", "localName");
jmethod.addParameter(null, AFTER, "Object", "value");
jmethod.addThrowSpecifier(null, AFTER, "JspException");
JotCodeBlock jcode = jmethod.getCodeBlock();
jcode.addComment(null, AFTER, JotComment.DOC,
    " @todo " + res.getString("SetDynamicAttributeTodo") + " ");
jcode.addStatement(null, AFTER, "throw new JspException(\"" +
    MessageFormat.format(res.getString("SetDynamicAttributeThrow"),
        new Object[] { "\" + localName + "\", \"" + uri + "\"" }) + "\")");
}

/**
 * Add SimpleTag method doTag override.
 *
 * @param jclass      the class being generated
 * @param isNested    true if the tag looks for an enclosing tag,
 *                    false if standalone
 * @param isRepeated  true if the tag iterates over its body,
 *                    false if it processes it just once (if at all)
 * @param isBodyAltered true if the tag changes the body content,
 *                    false if it leaves it unchanged
 * @param hasProperties true if the tag has properties,
 *                    false otherwise
 */
private void addDoTag(JotClassSource jclass, boolean isNested,
    boolean isRepeated, boolean isBodyAltered, boolean hasProperties) {
    JotMethodSource jmethod =
        jclass.addMethod(null, AFTER, "void", "doTag");
    JotComment jcomment =
        jclass.addComment(jmethod, BEFORE, JotComment.DOC,
            res.getString("DoTagMethod") + (hasProperties ?
                "\n" + res.getString("DoTagHasProperties") : "") +
                "\n\n@throws IOException " +
                res.getString("DoTagIOException") +
                "\n@throws JSPException " +
                res.getString("DoTagJspException") +
                "\n@throws SkipPageException " +
                res.getString("DoTagSkipPageException"));
    jclass.addBlankLine(jcomment, BEFORE);
    jmethod.setModifiers(Modifier.PUBLIC);
    jmethod.addThrowSpecifier(null, AFTER, "IOException");
    jmethod.addThrowSpecifier(null, AFTER, "JspException");
    JotCodeBlock jcode = jmethod.getCodeBlock();
    if (isNested) {
        jcode.addComment(null, AFTER, JotComment.LINE,
            res.getString("DoTagFindEnclosing"));
        jcode.addComment(null, AFTER, JotComment.DOC,
            " @todo " + MessageFormat.format(res.getString("OuterTagTodo"),
                OUTER_TAG) + " ");
        jcode.addAssignment(null, AFTER, "_enclosingTag",
            "(OuterTag)findAncestorWithClass(this, OuterTag.class)");
        JotIf jif =
            jcode.addIfStatement(null, AFTER, "_enclosingTag == null");
        jif.getThen().getCodeBlock().addStatement(null, AFTER,
            "throw new JspException(\"" +
                MessageFormat.format(res.getString("DoTagThrowNestingError"),
                    OUTER_TAG) + "\")");
    }
    JotVariableDeclaration jvar =
        jcode.addVariableDeclaration(null, AFTER, "JspWriter", "out");
    jvar.setInitializer("getJspContext().getOut()");

    if (isRepeated || isBodyAltered) {
        jcode.addComment(null, AFTER, JotComment.DOC,
            " @todo " + res.getString("TagOpeningOutputTodo") + " ");
        jcode.addStatement(null, AFTER, "out.print(\"" +
            res.getString("TagOpeningOutputValue") + "\")");
        jvar = jcode.addVariableDeclaration(
            null, AFTER, "JspFragment", "body");
    }
}

```

```

jvar.setInitializer("getJspBody()");
if (isRepeated) {
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + res.getString("RepeatInitTodo") + " ");
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + res.getString("RepeatTestTodo") + " ");
    JotWhile jwhile =
        jcode.addWhileStatement(null, AFTER, "repeatBody");
    jcode = jwhile.getCodeBlock();
}
jcode.addComment(null, AFTER, JotComment.DOC,
    " @todo " + res.getString("DoTagAccessBodyTodo") + " ");
jcode.addMethodCall(null, AFTER, "body.invoke", "out");
jcode = jmethod.getCodeBlock();
jcode.addComment(null, AFTER, JotComment.DOC,
    " @todo " + res.getString("TagClosingOutputTodo") + " ");
jcode.addStatement(null, AFTER, "out.print(\"" +
    res.getString("TagClosingOutputValue") + "\")");
}
else {
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + res.getString("TagOutputTodo") + " ");
    jcode.addStatement(null, AFTER, "out.print(\"" +
        res.getString("TagOutputValue") + "\")");
}
}
}

/**
 * Add Tag method doStartTag override.
 *
 * @param jclass      the class being generated
 * @param jspVersion  the version of JSP in use
 * @param isEndUsed    true if content is written after the body,
 *                    false if none
 * @param isNested    true if the tag looks for an enclosing tag,
 *                    false if standalone
 * @param isRepeated   true if the tag iterates over its body,
 *                    false if it processes it just once (if at all)
 * @param isBodyAltered true if the tag changes the body content,
 *                    false if it leaves it unchanged
 * @param hasProperties true if the tag has properties,
 *                    false otherwise
 */
private void addDoStartTag(JotClassSource jclass, String jspVersion,
    boolean isEndUsed, boolean isNested, boolean isRepeated,
    boolean isBodyAltered, boolean hasProperties) {
    boolean isJSP11 = jspVersion.equals(JSPTagProperties.JSP_11);
    JotMethodSource jmethod = jclass.addMethod(null, AFTER,
        "int", "doStartTag");
    JotComment jcomment =
        jclass.addComment(jmethod, BEFORE, JotComment.DOC,
            res.getString("DoStartTagMethod") + "\n\n" +
            (hasProperties ? res.getString("DoStartTagHasProperties") : "") +
            "\n\n@return " +
            (!isBodyAltered && !isRepeated ? "" : MessageFormat.format(
                res.getString("DoStartTagReturnBody"), new Object[]
                {(isJSP11 ? "EVAL_BODY_TAG" : "EVAL_BODY_BUFFERED"),
                (isJSP11 ? res.getString("DoStartTagReturnIterates") : "")})) +
            "\n\n" + MessageFormat.format(
                res.getString("DoStartTagReturn"), new Object[]
                {"EVAL_BODY_INCLUDE", "SKIP_BODY"}) +
            "\n@throws JspException " +
            res.getString("DoStartTagJspException"));
    jclass.addBlankLine(jcomment, BEFORE);
    jmethod.setModifiers(Modifier.PUBLIC);
    jmethod.addThrowSpecifier(null, AFTER, "JspException");
    JotTry jtry = jmethod.getCodeBlock().
        addTryStatement(null, AFTER, "IOException", "ex");
    JotCodeBlock jcode = jtry.getCodeBlock();
    JotIf jif = null;

```

```

if (isNested) {
    jcode.addComment(null, AFTER, JotComment.LINE,
        " " + res.getString("DoTagFindEnclosing"));
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + MessageFormat.format(res.getString("OuterTagTodo"),
            OUTER_TAG) + " ");
    jcode.addAssignment(null, AFTER, "_enclosingTag",
        "(OuterTag)findAncestorWithClass(this, OuterTag.class)");
    jif = jcode.addIfStatement(null, AFTER, "_enclosingTag == null");
    jif.getThen().getCodeBlock().addStatement(null, AFTER,
        "throw new JspException(\"" +
            MessageFormat.format(res.getString("DoTagThrowNestingError"),
                OUTER_TAG) + "\")");
}
JotVariableDeclaration jvar =
    jcode.addVariableDeclaration(null, AFTER, "JspWriter", "out");
jvar.setInitializer("pageContext.getOut()");
jcode.addComment(null, AFTER, JotComment.DOC,
    " @todo " + res.getString("TagOpeningOutputTodo") + " ");
jcode.addStatement(null, AFTER, "out.print(\"" +
    (isEndUsed ? res.getString("TagOpeningOutputValue") :
        res.getString("TagOutputValue")) + "\")");
jtry.getCatches()[0].getCodeBlock().
    addStatement(null, AFTER, "ex.printStackTrace()");
jtry.getCatches()[0].getCodeBlock().addStatement(null, AFTER,
    "throw new JspException(ex.getMessage())");

if (isRepeated) {
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + res.getString("RepeatInitTodo") + " ");
    jcode.addComment(null, AFTER, JotComment.DOC,
        " @todo " + res.getString("RepeatTestTodo") + " ");
    jif = jcode.addIfStatement(null, AFTER, "repeatBody", true);
    jcode = jif.getThen().getCodeBlock();
    jcode.addComment(null, AFTER, JotComment.DOC, " @todo " +
        res.getString("DoStartTagFirstIterationTodo") + " ");
    jcode.addReturnStatement(null, AFTER,
        (isJSP11 ? "EVAL_BODY_TAG" : "EVAL_BODY_BUFFERED"));
    jif.getElse().getCodeBlock().
        addReturnStatement(null, AFTER, "SKIP_BODY");
}
else {
    jcode.addReturnStatement(null, AFTER, (isBodyAltered ?
        (isJSP11 ? "EVAL_BODY_TAG" : "EVAL_BODY_BUFFERED") :
        (isEndUsed ? "EVAL_BODY_INCLUDE" : "SKIP_BODY")));
}
}

/**
 * Add Tag method doInitBody override.
 *
 * @param jclass the class being generated
 */
private void addDoInitBody(JotClassSource jclass) {
    : // Code removed
}

/**
 * Add Tag method doAfterBody override.
 *
 * @param jclass the class being generated
 * @param jspVersion the version of JSP in use
 * @param isRepeated true if the tag iterates over its body,
 *                  false if it processes it just once (if at all)
 */
private void addDoAfterBody(JotClassSource jclass, String jspVersion,
    boolean isRepeated) {
    : // Code removed
}

```

```

/**
 * Add Tag method doEndTag override.
 *
 * @param jclass the class being generated
 */
private void addDoEndTag(JotClassSource jclass) {
    : // Code removed
}

/**
 * Add TryCatchFinally methods doCatch and doFinally.
 *
 * @param jclass the class being generated
 */
private void addTryCatchFinallyMethods(JotClassSource jclass) {
    : // Code removed
}
}

```

The `createClass` method actually adds the new class to the file. It also supplies a Javadoc comment for the class and sets its visibility and parent class. If the JSP 1.2 try-catch-finally or the JSP 2.0 dynamic attributes functionality is required, the appropriate interface is added as well.

The `addInternalFields` method prepares the class, if necessary, by declaring fields for the attributes that it accepts. Each field is created with a type and name, and then its visibility is set. In the `addAttributeSetters` method, setter methods are inserted for each attribute entered by the user. This method shows how to add a parameter to a method, and how to generate assignment statements.

The `addDoStartTag` method illustrates adding a more complex method. A possible exception thrown by the method is specified, as is its visibility, and a Javadoc comment. The main body of the method is enclosed within a `try` statement, with code added to its first (and only) `catch` clause to report the error and transform it into another type of exception. You also see a variable declaration with an initializer value, `if` statements (one with an `else`), various assignments, method calls, and `return` statements.

TIP

Add Javadoc comments that start with “@todo” at appropriate places in your code. These are automatically found by JBuilder and are presented in the Structure Pane for the file under the To Do node. The entries highlight parts of the code where the user needs to make some changes, enabling them to quickly complete the skeleton created by the wizard.

Code for some of the remaining methods is not included in this listing since it follows a similar pattern to `addDoStartTag`. The full code is available on the accompanying Web site.

Compile the wizard and its supporting classes, place it in a JAR file along with a manifest containing the following entry, move the JAR file to the {JBuilder}/lib/ext directory, and restart JBuilder.

```
OpenTools-Wizard: wood.keith.opentools.wizards.jsptags.JSPTagWizard
```

Open the Object Gallery (File | New on the menu), go to the Web tab, and start the new wizard. The result of running it with the data shown in Figure 24-1 is the new class in Listing 24-3.

Listing 24-3. A new JSP tag class.

```

package wood.keith.qn.tags;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

/**
 * A JSP 2.0 tag with the following characteristics
 * - it refers to a surrounding tag.
 *
 * @author JSP Tag Wizard
 * @version 1.0 27 April 2004
 */

public class TestTag extends TagSupport {

    // Internal fields
    /** @todo set default values */
    private int _size;
    /** @todo replace OuterTag with the actual class name */
    private OuterTag _enclosingTag = null;

    /**
     * Save the value of size
     *
     * @param value the attribute value
     */

    public void setSize(int value) {
        _size = value;
    }

    /**
     * Process the start tag for this instance.
     * When this method is invoked, the body has not yet been evaluated.
     *
     * The doStartTag() method assumes that all property setter
     * methods have been invoked before this call.
     *
     * @return EVAL_BODY_INCLUDE if the tag just evaluates and includes the
     *         body content, or SKIP_BODY if it does not want to process it
     * @throws JspException if a problem arises
     */

    public int doStartTag() throws JspException {
        try {
            // Find enclosing tag for a nested tag
            /** @todo replace OuterTag with the actual class name */
            _enclosingTag = (OuterTag)findAncestorWithClass(
                this, OuterTag.class);
            if (_enclosingTag == null) {
                throw new JspException("This tag must appear within OuterTag");
            }
            JspWriter out = pageContext.getOut();
            /** @todo specify any opening output value for this tag */
            out.print("tag value");
            return SKIP_BODY;
        }
        catch(IOException ex) {
            ex.printStackTrace();
            throw new JspException(ex.getMessage());
        }
    }
}

```

Summary

The Java Object Toolkit assists you in parsing existing Java code, in altering it, and in generating entire new classes and files. It provides interfaces that mirror the structure of Java files, classes, and their contents. You navigate down through the different levels to find the items of interest, or create new classes, their methods, and fields.

JBuilder uses JOT internally as the basis for the Component Modeling Tool (see Chapter 20) and for many of its own wizards. Although wizards often draw on JOT's abilities, it can also be put to good use in other types of tools.

JOT is used in the Tag Library Descriptor wizard from last chapter to scan a project for prospective tag and supporting classes, and to extract details about them. In this chapter, the JSP Tag wizard helps you create these tag classes using JOT. After asking the user for information about the required tag, the tool generates an entire new Java class with the requested functionality, adds it to the current project, and opens it in the Content Pane.

Several of the samples that come with JBuilder also use JOT to provide their abilities.



WARNING

Because these samples are declared as belonging to the “Wizards” category in their `classes.opentools` files, you must bring up the Object Gallery or the Wizards menu before they appear on the Tools menu.

```
{JBuilder}/samples/OpenToolsAPI/jot/PackageTree/  
PackageTree.jpx
```

This tool displays a window showing the class ancestry of every class in the package of the currently active Java node. It appears on the Tools menu as **Package Tree** and is disabled if the current node is not a Java source file in a JBuilder project. The `PackageTree` and `PackageTreeReader` classes are the ones that interact with JOT.

```
{JBuilder}/samples/OpenToolsAPI/jot/ReadingSource/  
ReadingSource.jpx
```

Demonstrating how to read a Java source file, this tool reads the current Java source file and opens a tab in the Message Pane to display details about. It runs from the **Reading Source** option on the Tools menu, and shows general statistics, information about the first class in the file, that class' first field and method, and then some of the top-level statements in that method.

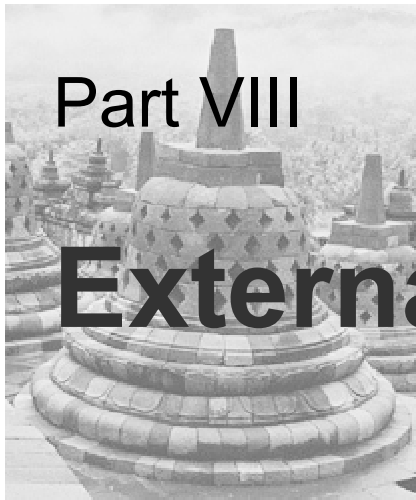


WARNING

There is an error in the code for this tool. In the `interrogateFile` method it retrieves the parameters for the first method in the first class (around line 190-200). The correct code should read `“firstMethod.getParameters().length > 0”` rather than `“firstMethod.getParameters().length < 0”`.

```
{JBuilder}/samples/OpenToolsAPI/jot/WritingSource/  
WritingSource.jpx
```

Complementing the previous tool, this one generates a simple class that illustrates several of the aspects of using JOT to write code. A `JotWriting.java` file is added to the current project when the wizard is run from the **Writing Source** option on the Tools menu.



Part VIII

External Systems

Although JBuilder provides a lot of functionality built-in, and allows you to enhance its abilities through the OpenTools API, there are certain tasks that are better preformed by external systems. These include the management of code bases by a Version Control System and the running of an application server for your EJBs and other J2EE components.

Several companies offer products in these two areas, and by tying into these instead of trying to implement it all, JBuilder allows you to integrate best-of-breed applications into its IDE, gaining the best of both worlds.

Chapter 27 looks at incorporating an external Version Control System into JBuilder so that you can easily check code in and out, determine the status of projects and individual classes, and compare different versions of a particular source file.

Chapter 28 explains how to link an application server into JBuilder, allowing you to easily deploy your application to it, to start and stop it, and to monitor your code in that environment.

Chapter 27

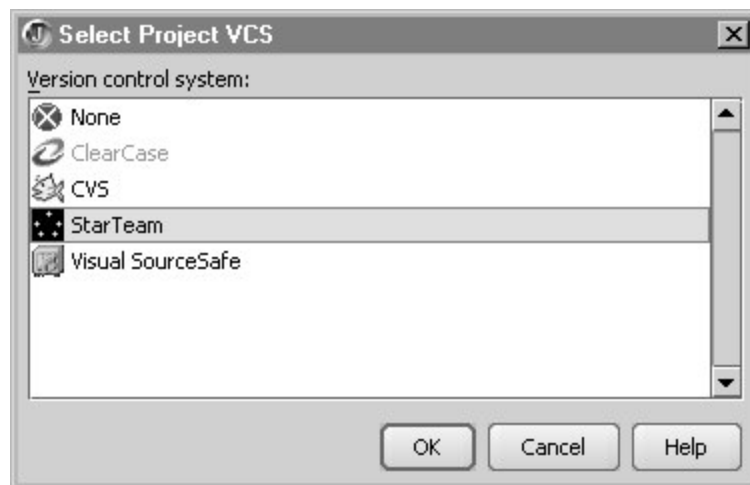
Version Control Systems

For any large project, and for many smaller ones, a *Version Control System* (VCS) is an essential part of managing the code base. It lets you handle multiple requests for a single object, ensuring that no changes are lost, as well as providing a history of modifications that can be tracked over time. Because of the complexity of the management task, and its ability to apply to many different programming environments, most VCSs are standalone packages.

JBuilder integrates external VCS implementations into its IDE, so that you can check files in and out immediately, review the status of all the files in a project, or resolve conflicts between multiple versions of the one file. Each VCS requires an adapter package that merges the abilities of that system into JBuilder, passing user requests onto the actual external tool. Although this adapter package is Java-based, the VCS itself can be a native executable. Currently JBuilder supports CVS, Visual SourceSafe, ClearCase, and StarTeam out of the box.

Each adapter registers itself with JBuilder (within the “UI” category) so that its abilities can be used. One of these is then chosen as the active VCS for each project from the Team | Select Project VCS menu item as shown in Figure 27-1. Following configuration of the VCS through a property page provided by the adapter, the external VCS is ready to use from the Team menu (see Figure 27-2).

Figure 27-1. Select a VCS for a project.



To establish your own VCS interface, you must create a VCS descendent and register it with the `VCSFactory` class. From your class, the IDE extracts the VCS's name and icon for selection, followed by its property page for configuration. Once chosen, this class provides the actions that appear on the **Team** menu and in the Project Pane's context menu. These actions may perform any kind of operation appropriate to this VCS.

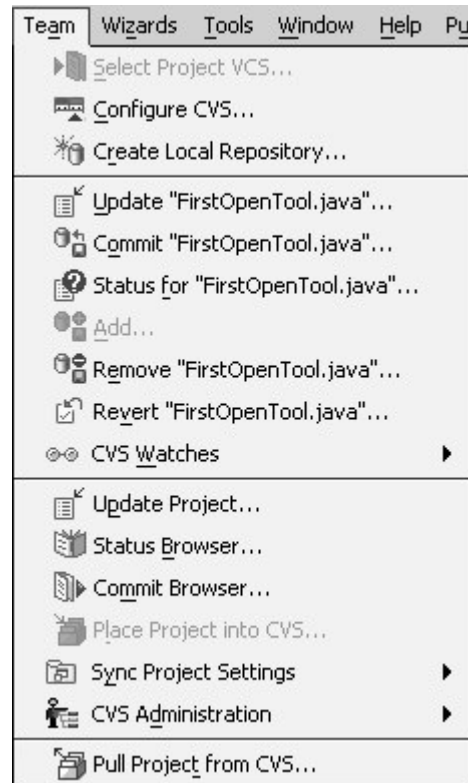
Details about individual files are provided through `VCSFileInfo`, `VCSFileStatus`, and `RevisionInfo` objects. These items are returned by methods in the other classes, and are passed back as parameters to subsequent calls.

Project-wide committing of changes to the VCS is handled through the `VCSCommitBrowser`, a standard dialog that lists the changed files and allows selection of which ones to actually apply. You would call this dialog via the `VCSUtils` class and pass it a `CommitAction` to use for the VCS interactions. `VCSUtils` provides many other useful methods for integrating your VCS application.

The VCS is also integrated into the History tab that appears in the Content Pane for all nodes. A list of all the revisions for a file are retrieved from the VCS and displayed at the top of the Contents page, showing the revision number, label, timestamp, and author. Then, as a version is selected, its content is loaded and presented at the bottom of the page. The Info page is similar but shows the VCS labels and full comment for the file instead of its content.

Comparing versions of a file is done on the Diff page of the history. Select the revisions to compare from the lists at the top of the page, and the differences are highlighted in the combined text at the bottom. This functionality is part of JBuilder and does not rely on any differencing abilities of the underlying VCS. All it needs is access to the source for the two revisions. It is not possible to compare non-text files.

Figure 27-2. VCS options on the **Team** menu.



VCSFactory Class

The `com.borland.primetime.teamdev.vcs.VCSFactory` class maintains the list of available VCSs within JBuilder. Each VCS implementation must register itself with this class before it appears in JBuilder.

Its (all static) methods are:

```
public static void addVCS(VCS vcs);
```

Register a new VCS implementation with JBuilder (under the “UI” category).

`vcs` is the interface to an external VCS.

```
public static String[] getNames();
```

Retrieve the list of names of registered VCSs. You receive an empty array if none have been registered yet.

```
public static VCS getVCS(String vcsName);
```

Obtain a reference to a particular VCS interface, or `null` if it cannot be found.

`vcsName` is the name of the VCS required.

VCS Class

For each VCS that wants to integrate with JBuilder, there must be a corresponding `com.borland.primetime.teamdev.vcs.VCS` implementation. This new class indicates the functionality provided by the underlying VCS and supplies the connections to it.

The methods for this interface/class are listed below:

```
public void addToIgnoreList(URL[] filesAndDirsToIgnore);
```

Add entries to the list of files and directories ignored by the VCS.

`filesAndDirsToIgnore` is the list of files and/or directories to add.

```
public abstract String getDescription();
```

Provide a longer description of the VCS underlying this implementation.

```
public String getMergeConflictDividerMarker();
```

```
public String getMergeConflictEndMarker();
```

```
public String getMergeConflictStartMarker();
```

These methods return text strings that are used to delimit areas of conflict during a merge process. By default the values are “=====” for the divide marker, “>>>>>>>” for the end marker, and “<<<<<<<” for the start.



VERSION

None of the `getMerge*` methods are available in JBuilder 7.

```
public abstract String getName();
```

Obtain the display name for this VCS fronted by this implementation.

```
public WizardAction getNewProjectFromVCSWizardAction();
```

To support retrieving a project from the VCS from the Object Gallery or Team menu, return an appropriate wizard action, or `null` (the default) if this activity is not supported.

```
public abstract PropertyPage getProjectConfigPage(
    JBuilderProject project);
```

```
public abstract PropertyPage getProjectConfigPageNew(
    Project project);
```

Return a property page used to configure this VCS. It appears within the Repository dialog.

`project` is the project being configured.

**VERSION**

The `getProjectConfigPage` method is not available in JBuilder 9 and 10, where it has been replaced by the `getProjectConfigPageNew` method.

**NOTE**

Any property values captured by this page should be stored with the project as automatic properties (see Chapter 7), using the project's `setAutoProperty` method to save them and `getAutoProperty` to retrieve them. The category (first parameter) should always be the standard `VCS.CATEGORY` value. The property name must be unique across all instances, so include the name of your VCS, like "SourceSafeVCS_user".

```
project.setAutoProperty(
    VCS.CATEGORY, SourceSafeVCS.SS_USER, pnlConfig.getSourceSafeUserId());
```

```
public Map getProjectStatus(Project project);
```

Discover the status of each file in the project. The returned map contains `VCSFileInfo` objects as keys representing the directories for the project. Each matching value is then a `List` of more `VCSFileInfo` objects for each file found there. Displaying the full status of the project with the `VCSCommitBrowser` object invokes this method. You can trigger this browser through the `showProjectStatus` method of the `VCSUtils` class.

`project` is the project to scan.

```
public UpdateAction getRefactorCheckoutAction(Url[] urls,
    Component parentComponent, ChangeListener listener);
```

Provide a checkout action for the refactoring viewer when a read-only file needs to be updated, or `null` if this activity is not supported.

`urls` is the list of files to checkout.

`parentComponent` is a parent to use for any dialogs.

`listener` is the object to notify following a checkout attempt.

**VERSION**

The `getRefactorCheckoutAction` method is only available in JBuilder 9 and 10.

```
public Vector getRevisions(Url url);
```

Find the revision history for a file. The returned list contains `RevisionInfo` objects (described below).

`url` is the file to check for history.

```
public abstract byte[] getSource(Url url, RevisionInfo
    rev);
```

Retrieve the source for a given revision of a file. The returned byte array may contain text or binary data depending on the type of the file.

`url` is the file to retrieve.

`rev` indicates which revision is required.

```
public abstract ActionGroup getVCSContextMenuGroup();
```

Supply actions to add to the Project Pane popup menu for this VCS, or `null` if none apply.

```
public abstract ActionGroup getVCSFileMenuGroup();
```

Obtain the file-based actions for the Team menu, or `null` if there are none.

```
public abstract ActionGroup getVCSGlobalMenuGroup();
```

Get the actions not of a project- or file-based nature for this VCS, or null if none apply.

```
public abstract Icon getVCSIcon();
```

Provide an icon to represent this VCS. It should be 20 x 20 pixels in size.

```
public abstract ActionGroup getVCSProjectMenuGroup();
```

Supply the project-based actions to use in the Team menu, or null if no such actions apply.

```
public boolean isBinary(Url url);
```

Discover whether the VCS regards the file as containing binary data, returning true (the default) if it does, or false if it does not. Often binary files have no deltas (changes) recorded within the VCS.
url is the file to check.

```
public boolean isConfigureVCSMenuEnabled();
```

Determine whether the configuration menu entry for this VCS should be enabled. By default it returns true.

```
public boolean isSelectVCSMenuEnabled();
```

Indicate whether the Select Project VCS menu item should be enabled, returning true (the default) if so, or false if it should be disabled.

**VERSION**

The `isSelectVCSMenuEnabled` method is not available in JBuilder 7.

```
public abstract boolean isUnderVCS(Url url);
```

Returns true if the given file is under the control of the VCS, or false if it is not.
url is the file to check.

```
public void notifyVCSSelected(Project project);
```

Receive notification that the given project has just been assigned to this VCS through the Select Project VCS dialog.
project is the project affected.

**VERSION**

The `notifyVCSSelected` method is not available in JBuilder 7.

```
public void removeFromIgnoreList(Url[]
    filesAndDirsToStopIgnoring);
```

Remove entries from the list of files and directories ignored by the VCS.
filesAndDirsToStopIgnoring is the list of files and/or directories to remove.

These fields are also defined in the interface:

```
public static final String CATEGORY;
```

The name of the property category for VCS entries.

```
public static final String PROP_DELETE_FROM_REPO;
```

```
public static final String PROP_IGNORE;
```

```
public static final String PROP_RENAME_IN_REPO;
```

The names of properties.

**VERSION**

The `PROP_RENAME_IN_REPO` field is only available in JBuilder 9 and 10.

RevisionInfo Class

The `com.borland.primetime.teamdev.vcs.RevisionInfo` class encapsulates details about a particular revision for a file. A vector of these objects is returned by the `getRevisions` method of the VCS interface for use in the History tab in the Content Pane.



UNDOCUMENTED

The `RevisionInfo` class has not yet been documented.

Its methods are as shown here:

```
public RevisionInfo();
public RevisionInfo(String revNumber, long date, String
    author, String comment, String label);
    Create a new revision information object.
    revNumber, date, author, comment, and label initialize the appropriate
    fields below.
public String getAuthor();
public String getComment();
public long getDate();
    Find out the author for a revision, any comment for it, or its timestamp.
public String getLabel();
    Retrieve the label for this revision. A label is a tag that identifies a grouping
    of revisions across different files. For example, it may indicate the latest
    production release. If there are no labels an empty string is returned. The first
    label is returned if there are several.
public String[] getLabels();
    Retrieve the full set of labels for this revision, or an array with one empty
    string if there are none.
public AbstractRevisionNumber getRevisionNumber();
    Get the revision number for this revision.
public boolean isUnderVCS();
    Return true if this file under the control of the VCS, or false if it is not.
public boolean isWorkingRevision();
    Return true if this file is the revision currently being worked on, or false if it
    is not.
public void setAuthor(String author);
public void setComment(String comment);
public void setDate(long timestamp);
public void setLabel(String label);
public void setLabels(String[] labels);
public void setRevisionNumber(int revNumber);
public void setRevisionNumber(String revNumber);
public void setVCSFlag(boolean underVCS);
public void setWorkingRevision(boolean workingRev);
    Establish values for the corresponding fields for this revision.
```

Two constants are also defined for this class:

```
public static final String BUFFER_REVISION;
public static final String FILE_REVISION;
```

These identify revisions coming from the internal buffer or the file (working copy).

AbstractRevisionNumber Class

Each revision has an identifying number associated with it as provided by descendants of the `com.borland.primetime.teamdev.vcs.AbstractRevisionNumber` class. It is subclassed to provide the `IntegerRevisionNumber` (like 9), `NumericRevisionNumber` (like 1.4 or 2.1.3.3), and `StringRevisionNumber` (like “~23~”) classes from the same package.

Its methods are listed below:

```
public int compareTo(Object other);
```

Compare this revision with another and return a negative value if this object is less than the other, zero if they are equal, or a positive value if this one is greater. If the revision number objects are of different subclass types, their precedence is compared (see `getPrecedence`). If this is equal they are compared as string values. If the two objects are the same subclass then `doComparison` is called to compare them.

`other` is the other revision to compare.

```
public abstract int doComparison(AbstractRevisionNumber
arn);
```

Compare two instances of a particular subclass of `AbstractRevisionNumber`, returning the same values as `compareTo` above.

`arn` is the other revision number.

```
public abstract int getPrecedence();
```

Return a value indicating how different revision number subclasses sort relative to each other: strings come before numerics, which come before integers.

```
public static AbstractRevisionNumber
getRevisionNumberInstance(String revNumber);
```

Generate an appropriate subclass instance for the given revision number. This is `IntegerRevisionNumber` if the value starts with a digit and has no periods, `NumericRevisionNumber` if it starts with a digit and contains a period (.), or `StringRevisionNumber` otherwise.

`revNumber` is the revision number to represent.

```
public abstract String getRevisionString();
```

Present the revision number as a string value.

VCSFileInfo Class

Details about each file within a project are returned from the `VCS` class' `getProjectStatus` method as `com.borland.primetime.teamdev.vcs.VCSFileInfo` instances.

**UNDOCUMENTED**

Although this class does appear in the documentation, its methods have no explanations.

This class' methods are shown below:

```
public VCSFileInfo(VCSFileInfo info);
public VCSFileInfo(VCSFileStatus status);
public VCSFileInfo(Url url, VCSFileStatus status);
```

Create a new information object for a file.

info is another information object to copy.

status is the file status to wrap.

url is the file to which this status applies.

```
public String getComment();
```

Return any comment set for this file.

```
public File getFile();
```

Obtain a reference to the actual file to which this information applies.

```
public String getName();
```

Retrieve the name of the file represented by this object.

```
public VCSFileStatus getStatus();
```

Find the status of this file. The returned class is covered below.

```
public Url getUrl();
```

Get the file to which this information applies.

```
public void setComment(String comment);
```

Establish a comment for this file.

comment is the comment text.

```
public void setStatus(VCSFileStatus newStatus);
```

Update the status for this file. The parameter class is covered below.

newStatus contains the new settings.

```
public void setUrl(Url url);
```

Modify the file to which this information applies.

url is the new location of the file.

VCSFileStatus Class

Representing the different statuses applying to files managed by the VCS is the `com.borland.primetime.teamdev.vcs.VCSFileStatus` class. It is abstract, so must be subclassed by your implementation to provide appropriate information.

Its methods are listed here:

```
public abstract String getDescription();
```

Get a textual description of the status.

```
public int getStatus();
```

Retrieve the status of this file. These values are defined by your implementation and have no meaning outside of it.

```
public abstract Icon getStatusIcon();
```

Obtain an icon representing the file's status.

```
public abstract VCSFileActions getVCSFileActions();
```

Find the list of actions applicable to this file and its status. Return null if none apply. The `VCSFileActions` class is not covered in this book.

```
public boolean isCommentRequired();
```

Discover whether a comment is required for any of the possible actions on this file. The default is true.

```
public abstract boolean isModifiedInVCS();
```

Return true if this file has been changed in the VCS, or false if it is the same as the last check out.

```
public abstract boolean isModifiedLocally();
```

Return true if this file has been modified locally (in JBuilder), or false if it has not been touched since the last check in.

```
public abstract boolean isNew();
```

Return true if this is a new file, or false if it is not.

```
public void setStatus(int newStatus);
```

Update the status for this file. These values are defined by your implementation and have no meaning outside of it.

`newStatus` is the new setting.

This field is also defined:

```
protected int status;
```

The current status of the file.

VCSUtils Class

The `com.borland.primetime.teamdev.vcs.VCSUtils` class provides several utility methods that you can use in your VCS implementation, and in other OpenTools.

Its methods are all static, as shown below:

```
public static void addPersonalIgnoreFiles(Url[]
    ignoreFilesAndDirs);
```

```
public static void addPersonalIgnoreFiles(Url[]
    ignoreFilesAndDirs, Project project);
```

Add a list of files and/or directories to be ignored during VCS processing. These are saved in the local project file (`.local` extension) and so only apply to this machine. Use `addTeamIgnoreFiles` to share the list of excluded files, and `removePersonalIgnoreFiles` to delete entries.

`ignoreFilesAndDirs` is the list to ignore.

`project` is the project to apply the list to. The active project is used if this is not specified.



VERSION

The versions of these methods that take a `Project` reference are not available in JBuilder 7.

```
public static void addTeamIgnoreFiles(Url[]
    ignoreFilesAndDirs);
public static void addTeamIgnoreFiles(Url[]
    ignoreFilesAndDirs, boolean justModifyProjectFile);
public static void addTeamIgnoreFiles(Url[]
    ignoreFilesAndDirs, boolean justModifyProjectFile,
    Project project);
```

Add a list of files and/or directories to be ignored during VCS processing. These are saved in the project file (.jpr or .jpx extension) and apply to all users. Use `addPersonalIgnoreFiles` for private excluded files, and `removeTeamIgnoreFiles` to delete entries.

`ignoreFilesAndDirs` is the list to ignore.

`justModifyProjectFile` is true to only change the project file locally, or false (the default) to also commit the alterations to the VCS.

`project` is the project to apply the list to. The active project is used if this is not specified.

```
public static boolean checkProjectLocal(Url url);
```

Return true if this `Url` points to the project local file (".local" extension).
`url` is the file to test.



VERSION

The `checkProjectLocal` method is only available in JBuilder 9 and 10.

```
public static void createBackupAndOutputDirs();
```

Create the backup and output directories for the active project if they do not already exist.

```
public static boolean doesProjectTreeNeedRefreshed();
```

This method is called by the Commit Browser to determine whether the Project Tree needs to be refreshed, returning true if it does, or false if it does not.



VERSION

The `doesProjectTreeNeedRefreshed` method is only available in JBuilder 10.

```
public static void fixConflictsForEjbGrpXmlSource(File
    conflictFile, MessageCategory category);
```

Process conflicts resulting from a merge action on a file, and display progress in the Message Pane.

`conflictFile` is the EJB group file that contains conflict markers arising from a merge operation.

`category` identifies the tab in the Message Pane to write to.



VERSION

The `fixConflictsForEjbGrpXmlSource` method is not available in JBuilder 7.

```
public static void fixConflictsForJavaSource(File
    conflictFile, MessageCategory category);
```

Process conflicts resulting from a merge action on a file, and display progress in the Message Pane.

`conflictFile` is the Java source file that contains conflict markers arising from a merge operation.

`category` identifies the tab in the Message Pane to write to.

```
public static VCS getActiveVCS();
```

Obtain a reference to the current VCS, or null if there is none specified, or if the edition of JBuilder does not permit access to the VCS.

```
public static String getActiveVCSName();
```

```
public static String getActiveVCSName(Browser browser);
```

Returns the name of the current VCS, or null if none is specified.

`browser` is the browser to use to find the project. The active browser is examined if not specified.

```
public static Url getBackupUrl(Url url);
```

```
public static Url getBackupUrl(Url url, JBProject project);
```

```
public static Url getBackupUrl(Url url, Project project);
```

Find the location for a backup of a file within a project.

`url` is the file from the source path to find the backup for.

`project` is the project to locate the file within. If not specified, the active project is used.



VERSION

The `getBackupUrl` method that takes a `Project` parameter is only available in JBuilder 8 and up, replacing the version that takes a `JBProject` reference.

```
public static FileNode getBrowserActiveNode();
```

Get a reference to the file currently displayed in the active browser, or null if there is no file, or it is not a `FileNode` instance.

```
public static Url[] getExcludedPaths();
```

```
public static Url[] getExcludedPaths(Project project);
```

Obtain a list of all the shared excluded paths for a project. Use `getPersonalExcludedPaths` to find any local paths. Also see `addTeamIgnoreFiles` and `removeTeamIgnoreFiles`.

`project` is the project to examine, or the active project if not specified.

```
public static String[] getFilesNeededByVCS();
```

Retrieve a list of the names of files needed by the VCSs. These include files that record versions on the local machine.

```
public static Vector getLocalRevisions(FileNode fileNode);
```

Discover what revisions of a file are stored locally. The returned `Vector` contains `RevisionInfo` objects.

`fileNode` is the node to find backups for.

```
public static String getPathRelativeToProjectDirectory(
    Browser browser, Url url);
```

Find the relative path from the project's base directory to a given file.

`browser` is the active browser.

`url` is the file to locate.

```
public static Url[] getPersonalExcludedPaths();
public static Url[] getPersonalExcludedPaths(Project
project);
```

Obtain a list of all the private excluded paths for a project. Use `getExcludedPaths` to find any local paths. Also see `addPersonalIgnoreFiles` and `removePersonalIgnoreFiles`.

`project` is the project to examine, or the active project if not specified.

```
public static String getRelativePath(Url parent, Url
child);
```

Find the relative path from one `Url` to another.

`parent` is the base `Url` to start the path from.

`child` is the destination for the relative path.

```
public static FileNode[] getSelectedNodesInProjectPane();
```

Obtain a list of the nodes currently selected in the Project Pane.

```
public static boolean handleOldStyleProjects(Component
parentComponent, JBPProject project);
```

Deal with converting old style JBuilder projects (`.jpr` extensions) to the newer format (`.jpx` extension). The project is inspected, and if it is the old style, the user is prompted to convert it to the new style. A true value is returned if the project was converted after user confirmation, or if it was already in the new style. Otherwise false is returned.

`parentComponent` becomes the owner for any dialog that appears. It may be null.

`project` is the project to examine and convert if necessary.

```
public static boolean isBinaryFileNode(Class clazz);
```

Determine whether a class represents a binary file. It returns false if it descends from `TextFileNode`, or true otherwise.

`clazz` is the class to inspect.

NOTE

In JBuilder 9 and 10 you can register additional class types to ignore as binary – see `registerSourceClass`.

```
public static boolean isFileType(File file, String
fileType);
```

Discover whether a file is of a particular type with this method, which returns true if it is, or false if it is not.

`file` is the file to examine.

`fileType` is the extension of the type of file being checked (without any period).

```
public static boolean isInProjectDirectory(Url file);
```

Returns true if the specified file is in the same directory as the project file (`.jpr` or `.jpx` extension), or false if it is not.

`file` is the file to examine.

```
public static boolean isVCSFileOrDir(String fileOrDir);
```

Determine whether a given file or directory is one registered as being reserved for use by the VCS (see the `registerFileOrDirNeededByVCS` method), returning true if it is reserved, or false if not.

`fileOrDir` is the name of the file or directory to test.

**VERSION**

The `isVCSFileOrDir` method is not available in JBuilder 7.

```
public static void makeLocalBackup(Url file, int
    backupCount);
```

Make a backup of a file, with a specified number of historical copies.

`file` is the file to backup.

`backupCount` is the number of historical copies to retain.

**TIP**

Use the `EditorPropertyGroup.BACKUP_LEVEL` global property to find the system setting for the number of backups to make.

```
public static void notifyProjectTreeHasBeenRefreshed();
```

This method is called by the Commit Browser to notify that the project tree has been refreshed.

**VERSION**

The `notifyProjectTreeHasBeenRefreshed` method is only available in JBuilder 10.

```
public static void refreshHistoryPane();
public static void refreshHistoryPane(Url url);
```

Refresh the history pane because a label or branch has been added to the repository (first version), or a new revision of a file has been added or updated (second version).

`url` is the file that has been committed.

**VERSION**

The `refreshHistoryPane` methods are only available in JBuilder 9 and 10.

```
public static void registerFileOrDirNeededByVCS(String
    fileOrDirName);
```

Register the name of a file or directory used by the VCS, and so should be ignored during normal file processing.

`fileOrDirName` is the name of the reserved file or directory.

**VERSION**

The `registerFileOrDirNeededByVCS` method is not available in JBuilder 7.

```
public static synchronized void registerSourceClass(Class
    clazz);
```

List additional class types that are treated as source files. These types return false from the `isBinaryFileNode` method.

`clazz` is the class type that represents a source file type.

**VERSION**

The `registerSourceClass` method is only available in JBuilder 9 and 10.

```
public static boolean removeFile(Browser browser, Url
    file);
```

Delete the file from the browser and on disk, returning true if successful, or false if not.

browser is the browser to clear out.

file is the file to delete.

```
public static void removePersonalIgnoreFiles(Url[]
    ignoreFilesAndDirs);
```

```
public static void removePersonalIgnoreFiles(Url[]
    ignoreFilesAndDirs, Project project);
```

```
public static void removeTeamIgnoreFiles(Url[]
    ignoreFilesAndDirs);
```

```
public static void removeTeamIgnoreFiles(Url[]
    ignoreFilesAndDirs, Project project);
```

Delete files and/or directories that were previously marked to be ignored (see `addPersonalIgnoreFiles` and `addTeamIgnoreFiles`).

ignoreFilesAndDirs is the list to not ignore any more.

project is the project to remove the list from. The active project is used if this is not specified.

**VERSION**

The `removePersonalIgnoreFiles` and `removeTeamIgnoreFiles` methods that take a `Project` reference are not available in JBuilder 7.

```
public static void setStatusText(String text, Color color);
```

```
public static void setStatusText(String text, int type);
```

Send a new message to JBuilder's Status Pane.

text is the text to display.

color is the color to use for this text.

type is one of `TYPE_NORMAL`, `TYPE_WARNING`, or `TYPE_ERROR` from the `StatusView` class, indicating what format to use for the message.

**WARNING**

The `setStatusText` method that takes a `Color` parameter has been deprecated in JBuilder 9 and 10 in favor of the other version. The version that takes an integer parameter is not available in JBuilder 7.

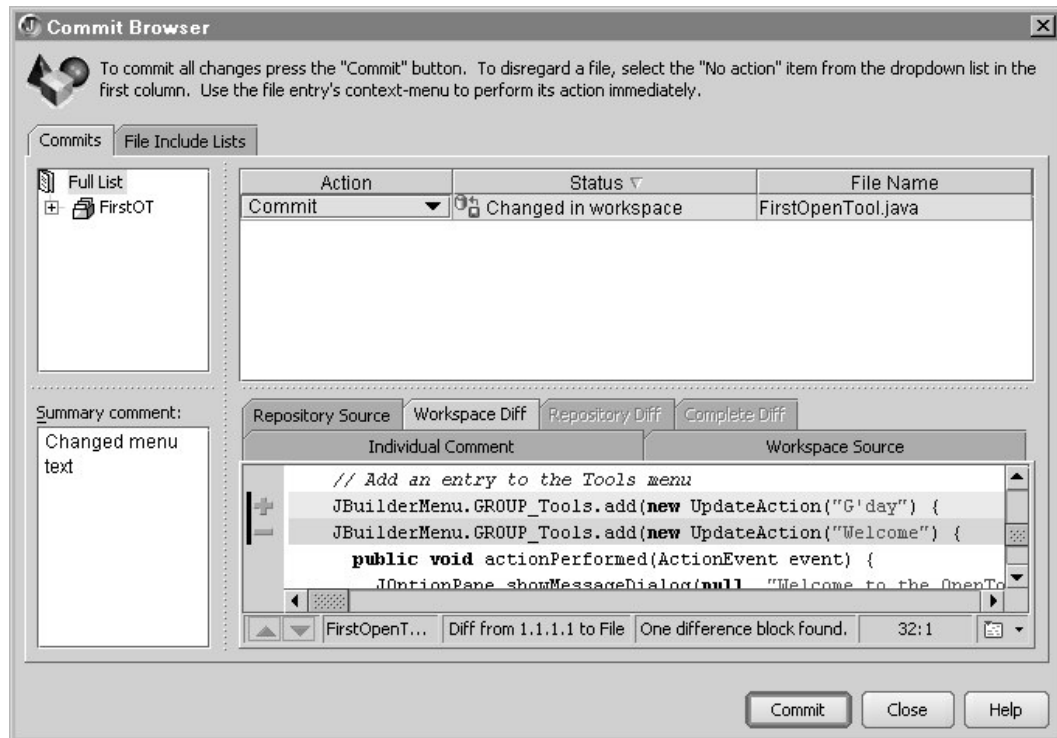
```
public static void showProjectStatus(Browser browser,
    CommitAction checkinAction);
```

Display a dialog to allow the user to examine the status of all updated files in the project and to commit changes to the underlying VCS. The dialog is an instance of the `VCSommitBrowser` class, and is shown in Figure 27-3.

browser is the browser to work with.

checkinAction is the action for a particular VCS that performs the check in of the selected files. See the next section for more details.

Figure 27-3. Preparing to commit changes to the project.



```
public static void showVcsConfigurationDialog();
```

Bring up the VCS configuration page in a dialog. This page may not allow updates, or may not be shown at all, depending on the VCS selected.

CommitAction Class

The abstract `com.borland.primetime.teamdev.vcs.CommitAction` class lets you define the operations necessary to commit changes to the VCS after files are selected in the `VCSCCommitBrowser`. You pass an object of this type to the `showProjectStatus` method of the `VCSUtils` class to let the user review the current situation and perform bulk operations on the VCS.

Its methods are shown below:

```
public void cancelOperation();
```

If the commit can be cancelled (see `isCancellable`) then this method must perform that activity, stopping the `performAction` method. In this class the method does nothing.



VERSION

The `cancelOperation` method is only available in JBuilder 9 and 10.

```
public abstract String getErrorMessage();
```

Supply an error message when the commit processing did not successfully finish.

```
public PropertyPage getPropertyPage();
```

Provide a custom property page (see Chapter 7) for the Commit Browser, or `null` (the default) if there is none.

**VERSION**

The `getPropertyPage` method is only available in JBuilder 9 and 10.

```
public boolean isCancellable();
```

Return true if the commit may be cancelled, or false (the default) if it cannot. If the former, then you must override the `cancelOperation` method to actually halt the commit.

**VERSION**

The `isCancellable` method is only available in JBuilder 9 and 10.

```
public abstract void performAction(VCSFileInfo[]
    fileInfos);
```

Update the VCS as indicated by the user's selections.

`fileInfos` is the list of files to process.

```
public abstract void setRunnerListener(
    OutputRunnerListener listener);
```

Attach a listener to the output from this action. The parameter class is not covered in this book, but captures output to standard output and error for later processing.

`listener` is the object to inform about any output.

```
public abstract boolean wasCommitSuccessful();
```

Returns true if the commit process completed successfully, or false if it did not.

**WARNING**

Note the spelling error in `Successful` in this method's name.

SourceSafeVCS Example

Illustrating how you can interface with an external VCS, David Brouse provides an `OpenTool` that talks to Visual SourceSafe (VSS). This VCS from Microsoft is commonly used on Windows platforms and is accessible through a command-line interface. The tool has a property page to establish a link with VSS and then adds several actions to both the Team menu and the popup Project Pane context menu. Through these actions you can perform all the usual VCS actions (except for difference).

**NOTE**

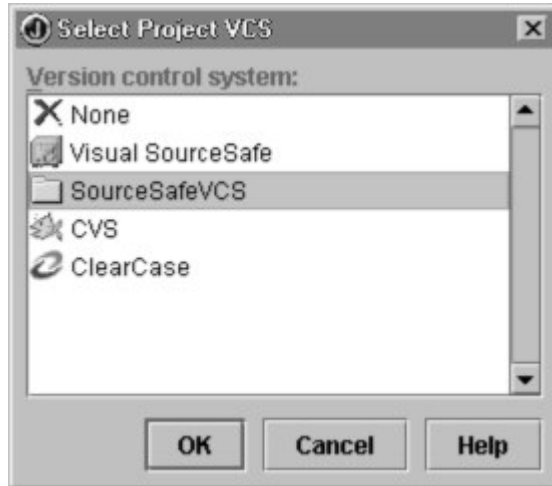
Support for Visual SourceSafe is built into JBuilder from version 5 onwards, but only in the Enterprise edition. David Brouse has also updated his `OpenTool` to access SourceSafe through its native API and is selling this version. He is no longer supporting the version shown here.

**BIO**

David Brouse has been in software development for 13 years. The last six years he has been designing and implementing n-tier web applications in Java. He is currently employed as a software engineer. He started the company Devious Bard Software (<http://www.deviousbard.com>) to sell some products that he has developed in his free time. The first product up for sale is his Native Source Safe Open Tool for JBuilder.

Although there are many classes that make up the tool, the main one is `SourceSafeVCS`, as shown in Listing 27-1. It extends the abstract `VCS` class and overrides the methods that implement the VCS functionality. In its `OpenTools` initialization routine it registers an instance of itself with the `VCSFactory` class. Once it is known to the system, this VCS can be assigned to a project via the **Team | Select Project VCS** menu option (see Figure 27-4). The name and icon come from the `getName` and `getVCSIcon` methods.

Figure 27-4. Selecting the new SourceSafe VCS.



Listing 27-1. SourceSafe OpenTool.

```
package com.deviousbard.ssvcs;

import com.borland.primetime.teamdev.vcs.*;
import com.borland.primetime.teamdev.frontend.*;
import com.borland.primetime.util.runner.*;
import com.borland.primetime.properties.PropertyPage;
import com.borland.primetime.node.*;
import com.borland.primetime.ide.*;
import com.borland.primetime.vfs.Url;
import com.borland.primetime.actions.ActionGroup;
import com.borland.jbuilder.node.JBProject;
import com.borland.primetime.ui.*;
import java.util.*;
import javax.swing.*;
import java.io.*;
import com.borland.jbcl.layout.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.*;
import com.borland.jbuilder.*;
import com.borland.primetime.*;
import com.deviousbard.opentool.util.*;
import java.text.*;

public class SourceSafeVCS extends VCS
{
    public static final String SS_EXE = "SourceSafeVCS_exe";
    public static final String SS_MAPPINGS = "SourceSafeVCS_mappings";
    public static final String SS_USER = "SourceSafeVCS_user";
    public static final String SS_PASSWORD = "SourceSafeVCS_password";
    public static final String SS_AUTH_REQ = "SourceSafeVCS_authreq";
    public static final String SS_COMMAND_OUTPUT =
        "SourceSafeVCS_commandOutput";
    public static final String SS_HIST_REV = "SourceSafeVCS_histRev";
}
```

```

SourceSafeVCSLogWriter log = null;

GetFileDialog getFileDialog = null;
CheckoutFileDialog checkoutFileDialog = null;
CheckinFileDialog checkinFileDialog = null;
UndoCheckoutFileDialog undoCheckoutFileDialog = null;
AddFileDialog addFileDialog = null;
MultiFileActionDialog multiFileActionDialog = null;
ProjectListDialog projectListDialog = null;

SimpleDateFormat df = new SimpleDateFormat("MM/dd/yy hh:mm");

private Node[] nodes = null;
private String msg = null;

public SourceSafeVCS() {
    try {
        log = new SourceSafeVCSLogWriter();
    }
    catch (IOException ioe) {
    }

    try {
        jbInit();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * Register the SourceSafeVCS with the VCS Factory. This will make
 * the class available in the Team drop-down list.
 */
public static void initOpenTool( byte major, byte minor ) {
    // VCS support was added starting from JBuilder 4
    if ( major < 4 ) {
        return;
    }
    VCSFactory.addVCS(new SourceSafeVCS());
}

/**
 * Return the property page to display when the user clicks
 * on the Team tab.
 */
public PropertyPage getProjectConfigPage(JBProject project) {
    return( new ConfigPage(project) );
}

public boolean isUnderVCS(Url url) {
    JBProject prj =
        (JBProject)Browser.getActiveBrowser().getActiveProject();
    String auth="";
    String ssExe = prj.getAutoProperty(VCS.CATEGORY, SS_EXE);
    String ssProjectFile =
        this.getSourceSafeProjectFile(url.getFileObject(), prj);
    String ssUser = prj.getAutoProperty(VCS.CATEGORY, SS_USER);
    String ssPass = prj.getAutoProperty(VCS.CATEGORY, SS_PASSWORD);
    String ssAuthRequired =
        prj.getAutoProperty(VCS.CATEGORY, SS_AUTH_REQ);
    if (ssAuthRequired.equals("TRUE")) {
        auth = " -Y" + ssUser + "," + ssPass + " ";
    }
    ModifiedShellRunner shellrunner = new ModifiedShellRunner();
    String command = "\"" + ssExe + "\"" + " Status " + "\"" +
        ssProjectFile + "\" -I-Y " + auth;
    if (prj.getAutoProperty(VCS.CATEGORY,
        SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))

```

```

        shellrunner.enableMessageViewOutput(true);
        shellrunner.run(command);
        if (prj.getAutoProperty(VCS.CATEGORY,
            SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))
            shellrunner.enableMessageViewOutput(false);
        if (shellrunner.getStderr().size() > 0)
            return false;
        for (Iterator iterator = shellrunner.getStdout().iterator();
            iterator.hasNext();) {
            String message = (String)iterator.next();
            if (message.
                indexOf("is not an existing filename or project") > 0)
                return false;
        }
        return true;
    }
}

public byte[] getSource(Url url, RevisionInfo revInfo) {
    SourceSafeRevisionInfo ssRevInfo = (SourceSafeRevisionInfo)revInfo;
    JBPProject prj =
        (JBPProject)Browser.getActiveBrowser().getActiveProject();
    String ssExe = prj.getAutoProperty(VCS.CATEGORY, SS_EXE);
    String ssUser = prj.getAutoProperty(VCS.CATEGORY, SS_USER);
    String ssPass = prj.getAutoProperty(VCS.CATEGORY, SS_PASSWORD);
    String ssAuthRequired =
        prj.getAutoProperty(VCS.CATEGORY, SS_AUTH_REQ);
    String auth="";
    if (ssAuthRequired.equals("TRUE")) {
        auth = "-Y" + ssUser + "," + ssPass + " ";
    }
    File tempFile = null;
    try {
        tempFile = File.createTempFile("SourceSafeVCS", null);
    }
    catch (IOException ioe) {
    }
    String path = tempFile.getParent();
    tempFile.delete();
    String fileName =
        url.getFile().substring(url.getFile().lastIndexOf('/') + 1);
    String mappedPath = this.getMappedPath(url.getFileObject(), prj);
    String fileDir = url.getFileObject().getParent();
    String filePath = path + "\\\" + fileName;
    int commandCount = 3;
    String[] commands = new String[commandCount];
    String drive = path.substring(0,2);
    commands[0] = drive;
    commands[1] = "cd " + "\"" + path.substring(2) + "\"";
    commands[2] = "\"" + ssExe + "\"" + " Get " + "\"" +
        ssRevInfo.getProjectFile() + "\"" + "-I-Y -GWR -V" +
        ssRevInfo.getSourceSafeRevision() + auth;
    ModifiedShellRunner shellrunner = new ModifiedShellRunner();
    if (prj.getAutoProperty(VCS.CATEGORY,
        SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))
        shellrunner.enableMessageViewOutput(true);
    shellrunner.run(commands);
    if (prj.getAutoProperty(VCS.CATEGORY,
        SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))
        shellrunner.enableMessageViewOutput(false);
    tempFile = new File(filePath);
    BufferedReader reader = null;
    String line = null;
    StringBuffer sourceFile = new StringBuffer();
    try {
        reader = new BufferedReader(new FileReader(tempFile));
        while ((line = reader.readLine()) != null) {
            sourceFile.append(line);
            sourceFile.append("\n");
        }
        reader.close();
    }
}

```

```

        tempFile.delete();
    }
    catch (FileNotFoundException fnfe) {
        System.out.println("Could not find file: " + fnfe);
    }
    catch (IOException ioe) {
        System.out.println("Could not read File: " + ioe);
    }
    return sourceFile.toString().getBytes();
}

public java.util.Map getProjectStatus(Project proj) {
    JBProject prj =
        (JBProject)Browser.getActiveBrowser().getActiveProject();
    FileScanner scanner = new FileScanner((JBProject)proj);
    scanner.scan();
    return null; //scanner.getFileChanges();
}

public VCSBrowserContextActionProvider
    getVCSBrowserContextActionProvider() {
    return null;
}

public boolean isBinary(Url url) {
    String extensions = ".gif.jpg.jpeg.jar.zip.au.wav.midi";
    return extensions.indexOf(url.getFileExtension()) != -1;
}

public String getName() {
    return "SourceSafeVCS";
}

public String getDescription() {
    return "A sample VCS implementation";
}

public Icon getVCSIcon() {
    return BrowserIcons.ICON_FOLDER;
}

public ActionGroup getVCSProjectMenuGroup() {
    JBProject prj =
        (JBProject)Browser.getActiveBrowser().getActiveProject();
    String ssExe = prj.getAutoProperty(VCS.CATEGORY, SS_EXE);
    String ssUser = prj.getAutoProperty(VCS.CATEGORY, SS_USER);
    if (ssExe == null && ssExe.equals(""))
        ssUser == null && ssUser.equals("")) {
        return null;
    }
    String ssPass = prj.getAutoProperty(VCS.CATEGORY, SS_PASSWORD);
    String ssAuthRequired =
        prj.getAutoProperty(VCS.CATEGORY, SS_AUTH_REQ);
    if (ssAuthRequired.equals("TRUE")) {
        if (ssPass == null && ssPass.equals("")) {
            return null;
        }
    }
    ActionGroup group = new ActionGroup();
    group.add(Actions.CREATE_PROJECT);
    group.add(Actions.GET_PROJECT);
    return group;
}

public ActionGroup getVCSFileMenuGroup() {
    JBProject prj =
        (JBProject)Browser.getActiveBrowser().getActiveProject();
    String ssExe = prj.getAutoProperty(VCS.CATEGORY, SS_EXE);
    String ssUser = prj.getAutoProperty(VCS.CATEGORY, SS_USER);

```

```

    if (ssExe == null || ssExe.equals(""))
        ssUser == null || ssUser.equals("")) {
        return null;
    }
    String ssPass = prj.getAutoProperty(VCS.CATEGORY, SS_PASSWORD);
    String ssAuthRequired =
        prj.getAutoProperty(VCS.CATEGORY, SS_AUTH_REQ);
    if (ssAuthRequired.equals("TRUE")) {
        if (ssPass == null || ssPass.equals("")) {
            return null;
        }
    }
    ActionGroup aGroup = new ActionGroup();
    aGroup.add(Actions.ADD_FILE);
    aGroup.add(Actions.CHECKIN_FILE);
    aGroup.add(Actions.CHECKOUT_FILE);
    aGroup.add(Actions.UNDO_CHECKOUT_FILE);
    aGroup.add(Actions.GET_FILE);
    aGroup.add(Actions.STATUS_FILE);
    return aGroup;
}

/** Always return true */
public boolean isConfigureVCSMenuEnabled() {
    return true;
}

/**
 * Return the list of actions that will be presented in the context
 * menu in the project pane.
 */
public ActionGroup getVCSContextMenuGroup() {
    : // Code removed - similar to getVCSProjectMenuGroup
}

public DeleteDialogInterface getDeleteDialog() {
    return null;
}

: // UI and utility code removed

public Vector getRevisions(URL url) {
    Vector revs = new Vector();
    try {
        JBProject prj =
            (JBProject)Browser.getActiveBrowser().getActiveProject();
        File workFile = url.getFileObject();
        String ssExe =
            prj.getAutoProperty(VCS.CATEGORY, SourceSafeVCS.SS_EXE);
        String ssUser = prj.getAutoProperty(VCS.CATEGORY,
            SourceSafeVCS.SS_USER).toLowerCase();
        String ssProjectFile = getSourceSafeProjectFile(workFile, prj);
        ModifiedShellRunner shellrunner = new ModifiedShellRunner();
        String auth = "";
        String ssPass =
            prj.getAutoProperty(VCS.CATEGORY, SourceSafeVCS.SS_PASSWORD);
        String ssAuthRequired =
            prj.getAutoProperty(VCS.CATEGORY, SourceSafeVCS.SS_AUTH_REQ);
        if (ssAuthRequired.equals("TRUE")) {
            auth = "-Y" + ssUser + "," + ssPass + " ";
        }
        String[] commands = new String[1];
        String ssHistRev =
            prj.getAutoProperty(VCS.CATEGORY, SourceSafeVCS.SS_HIST_REV);
        commands[0] = "\"" + ssExe + "\"" + " History " + "\"" +
            ssProjectFile + "\" -L- " + auth;
        if (prj.getAutoProperty(VCS.CATEGORY,
            SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))
            shellrunner.enableMessageViewOutput(true);
        shellrunner.run(commands);
    }
}

```

```

if (prj.getAutoProperty(VCS.CATEGORY,
    SourceSafeVCS.SS_COMMAND_OUTPUT).equals("TRUE"))
    shellrunner.enableMessageViewOutput(false);
if (shellrunner.getStdout().size() > 0) {
    Iterator iterator = shellrunner.getStdout().iterator();
    // First couple of rows of output are throw away lines
    String histRow = "";
    Date changeDate = null;
    int revCount = 0;
    while (iterator.hasNext() &&
        revCount < Integer.parseInt(ssHistRev)) {
        String revNumber = "";
        String who = "";
        String date = "";
        String time = "";
        String label = "";
        String desc = "";
        while (histRow.indexOf("***") == -1) {
            if (iterator.hasNext()) {
                histRow = (String)iterator.next();
            }
        }
        int versionIndex = histRow.indexOf("Version") + 8;
        if (versionIndex >= 8) {
            revNumber = histRow.substring(versionIndex,
                histRow.indexOf(" ", versionIndex) - 1).trim();
        }
        if (iterator.hasNext()) {
            histRow = (String)iterator.next();
            int userIndex = histRow.indexOf("User:") + 6;
            who = histRow.substring(userIndex,
                histRow.indexOf("Date:", userIndex) - 1).trim();
            int dateIndex = histRow.indexOf("Date:") + 6;
            date = histRow.substring(dateIndex,
                histRow.indexOf("Time:", dateIndex) - 1).trim();
            int timeIndex = histRow.indexOf("Time:") + 6;
            time = histRow.substring(timeIndex).trim() + "m";
            try {
                changeDate = df.parse(
                    ((date.length() < 8) ? "0" + date : date) + " " +
                    ((time.length() < 7) ? "0" + time : time));
            }
            catch (ParseException pe) {
                System.out.println("Parse Error: " + pe);
                changeDate = new Date();
            }
        }
        if (iterator.hasNext()) {
            histRow = (String)iterator.next();
            label = histRow.trim();
        }
        if (iterator.hasNext()) {
            histRow = (String)iterator.next();
            desc = null;
            try {
                int commentIndex = histRow.indexOf("Comment:") + 9;
                desc = histRow.substring(commentIndex).trim();
            }
            catch (ArrayIndexOutOfBoundsException aioobe) {
                desc = "";
            }
            catch (StringIndexOutOfBoundsException sioobe) {
                desc = "";
            }
        }
        while(iterator.hasNext() && (histRow.indexOf("***") == -1)) {
            histRow = (String)iterator.next();
            if (histRow.indexOf("***") == -1)
                desc += "\n" + histRow.trim();
        }
    }
}

```

```

    }
    SourceSafeRevisionInfo r = new SourceSafeRevisionInfo(
        revNumber, changeDate.getTime(), who, desc, label);
    r.setSourceSafeRevision(revNumber);
    r.setProjectFile(ssProjectFile);
    revs.add(r);
    revCount++;
}
}
}
catch (Exception ex) {
    log.writeLogEntry("Error In History: " + ex);
    ex.printStackTrace();
}
return revs;
}

: // Helper code removed

private class FileScanner {
    private JBProject project = null;
    private File projectDir = null;
    private String projectParent = null;
    private java.util.List fileChanges = new ArrayList();
    private java.util.List sourcePaths = new ArrayList();
    private int pParentLen=0;

    private FileScanner( JBProject project ) {
        try {
            this.project = project;
            File projectDir = project.getProjectPath().getFileObject();
            String sourcePath =
                project.getProperty(JBProject.PROPERTY_SOURCEPATH);
            String projectPath = project.getProjectPath().getFile();
            StringTokenizer st = new StringTokenizer(sourcePath, ";");
            while (st.hasMoreTokens()) {
                String tmpSrcPath = st.nextToken();
                String tmpPrjPath = projectPath;
                int index = -1;
                while ((index = tmpSrcPath.indexOf("../")) > -1) {
                    tmpPrjPath =
                        tmpPrjPath.substring(0, tmpPrjPath.lastIndexOf("/") - 1);
                    tmpSrcPath = tmpSrcPath.substring(index + 3);
                }
                sourcePaths.add(tmpPrjPath + "/" + tmpSrcPath);
            }
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public java.util.List getFileChanges() {
        return fileChanges;
    }

    public void scan() {
        for (Iterator i = sourcePaths.iterator(); i.hasNext();) {
            String path = (String)i.next();
            File f = new File(path);
            scan(f);
        }
    }

    public void scan(File path) {
        try {
            File[] list = path.listFiles();
            int len = list.length;
            for ( int i=0; i<len; i++ ) {
                // Skip status files

```

```

        if ( list[i].isDirectory() ) {
            scan( list[i] );
        }
        else {

            String fname = list[i].getCanonicalPath();
            // Ignore JBuilder and Source Safe Generated files
            if ( list[i].getName().endsWith("~")
                list[i].getName().endsWith("jpx.local")
                list[i].getName().endsWith("scc")) {
                continue;
            }
            else {
                int status =
                    SourceSafeVCSUtils.getStatus(list[i], project);
                Url url = new Url(list[i]);
                if ( SourceSafeVCSStatus.isNewFile(status) ) {
                    // Added
                    fileChanges.add(new VCSFileInfo(url,
                        new RevisionStatus(RevisionStatus.STATUS_NEW)));
                    continue;
                } else if (SourceSafeVCSStatus.
                    isDifferentFromWorkFile(status) ) {
                    // Modified
                    fileChanges.add(new VCSFileInfo(url, new
                        RevisionStatus(RevisionStatus.STATUS_WSP_CHANGE)));
                    continue;
                }
                if ( lastRev.compareTo(workRev) == 1 ) {
                    // Modified in the repo
                    mods.add(new VCSFileInfo(url,new
                        RevisionStatus(RevisionStatus.STATUS_REPO_CHANGE)));
                    continue;
                }
            }
        }
    }
}

catch (Exception ex) {
    ex.printStackTrace();
}
}
}

```

Next you need to initialize the VCS by selecting the **Team | Configure VCS** menu item. This option retrieves a property page from the `getProjectConfigPage` method in this class and displays it to the user (as shown in Figure 27-5). For VSS, the tool needs to know where to find the VSS program so that it can run it, any user name and password that are required to access it, as well as other optional information. These values are stored as automatic properties on the project (see Chapter 7) under names held in this class.

Figure 27-5. Configuring the SourceSafe VCS.

The screenshot shows a Windows-style dialog box titled "Configure Version Control". Inside, there's a section titled "SourceSafeVCS properties".

- Source Safe Executable (SS.EXE):** A text box contains "c:\winnt\system32\ss.exe" and a "Browse..." button is to its right.
- Source Safe Authentication:**
 - User ID:** A text box contains "user1".
 - Password:** A text box contains "*****".
 - A checkbox labeled "User ID and password required" is checked.
- Source Safe Project To Working Directory Mappings:**
 - A large empty rectangular box for mappings.
 - "Add" and "Remove" buttons are to the right of the box.
- History Revisions:**
 - A label "Number Of History Revisions To Display:" followed by a text box containing the number "5".
- Message Output:**
 - A checkbox labeled "Output Source Safe Commands To Message View" is unchecked.

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

Finally, the various menu group methods provide collections of actions to add to the different menus. `getVCSFileMenuGroup` and `getVCSProjectMenuGroup` go together to make up the contents of the new Team menu (see Figure 27-6), with the `isConfigureVCSMenuEnabled` method determining the state of the standard Configure VCS item, while `getVCSContextMenuGroup` supplies the entries for the Project Pane's popup menu (see Figure 27-7).

Figure 27-6. The new Team menu.

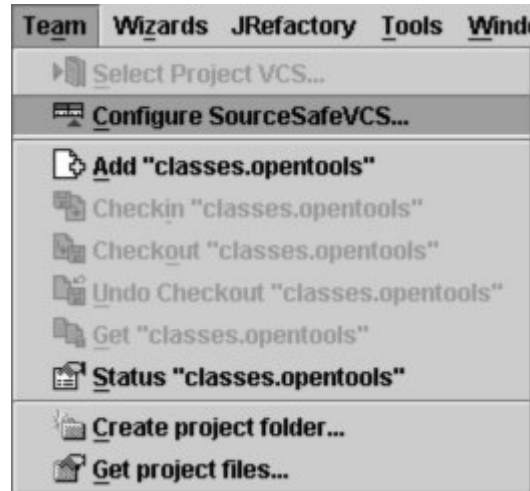
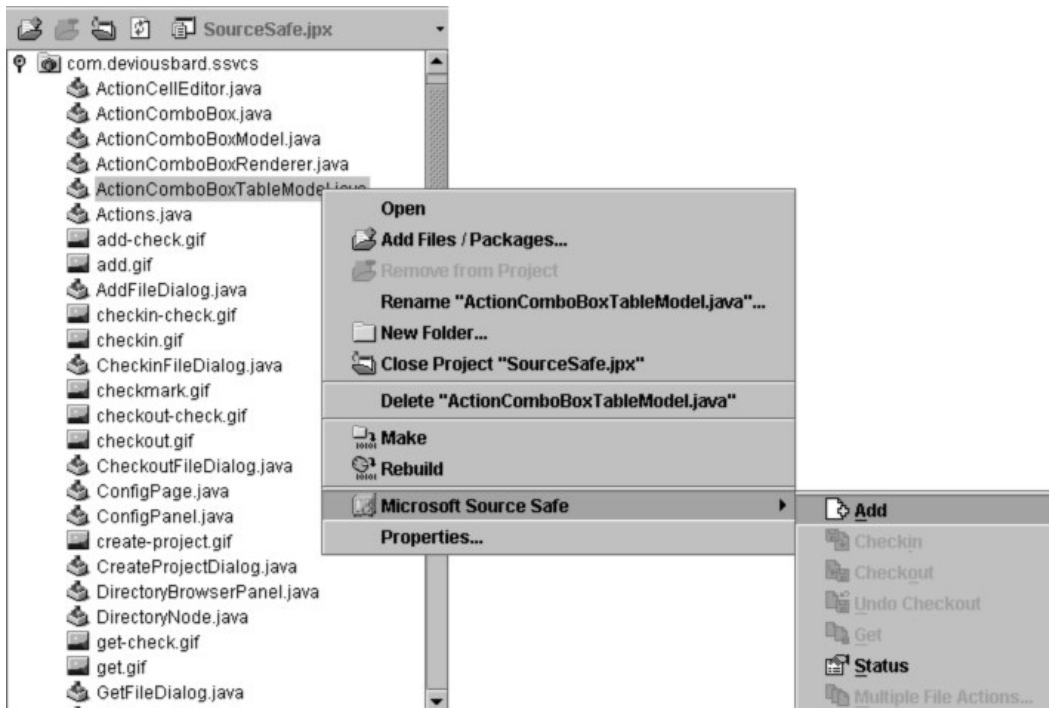


Figure 27-7. Popup options in the Project Pane.



Many of the remaining VCS methods follow a similar pattern. They retrieve the VCS property values entered by the user, and combine these into a command line that implements the requested action. The `ModifiedShellRunner` class (part of this tool) submits the command to the operating system and returns its results for interpretation by the calling method.

The tool is registered as part of the “UI” category, making it available as soon as the IDE is displayed.

```
OpenTools-UI: com.deviousbard.ssvcs.SourceSafeVCS
```

Summary

Version Control Systems provide the much-needed management of multiple demands on individual source files, and for histories of changes to these files. They allow you to return to any previous version, compare it with another version, and determine what changes were made and what effect that had on the application. Typically VCSs exist as separate applications, since they may be used in many different programming environments.

JBuilder provides access to several VCS implementations directly through its IDE, allowing you to easily check files in and out, to extract and compare earlier versions, and to review the status of the project. Currently, JBuilder supports CVS, Visual SourceSafe, ClearCase, and StarTeam with built-in functionality.

You can write your own adapters to deal with other VCS implementations, whether they are Java-based or not. As an example, the SourceSafe VCS tool from David Brouse was presented, showing how to invoke a native application via its command-line interface and interpret the results.

There is one JBuilder sample that illustrates the VCS API.

```
{JBuilder}/samples/OpenToolsAPI/VCS/samplevcs.jpx
```

This tool defines a fairly simple VCS, named `SampleVCS`, that just copies the files under its care to a parallel directory structure. The root of that structure is set as a property of the VCS for each project. Having the version number appended to the file name of the copy identifies the different revisions. This VCS stores additional for each revision in a separate `.comment` file, and even supports check out and check in.



Chapter 28

Application Servers

Another area where JBuilder needs to integrate with an external system is in working with application servers. These servers provide a hosting environment for Enterprise JavaBeans (EJBs) and other enterprise-level components. Implementations are available from many different sources, from commercial servers to open source efforts.

By integrating an application server with JBuilder, through an OpenTool adapter, you can easily configure the server, deploy your EJBs, and run or debug your application within the server, all from the JBuilder IDE. A particular server can be associated with each project. In fact, in JBuilder 7 and up, you can define a different server for each application server feature (such as EJB hosting, servlet/JSP, naming services, etc.) for each project.



NOTE

The functionality described here is only available in the Enterprise editions of JBuilder. Even then, certain additional classes are required to implement the basic application server abilities. These classes come with the Borland Enterprise Server that accompanies JBuilder and must be available on the classpath before certain options are enabled.

Up to JBuilder 6, server adapters were extensions of the `AppServer` class, and were registered with the `AppServerManager` class before they could be used. Each one then identified itself, defined its capabilities, and provided numerous details to enable its server to run. The `DeploymentDescriptor` class encapsulated descriptors of the EJBs to be deployed to that server, and these were created by a subclass of the `AbstractDescriptorConversion` class. Meanwhile the `EjbDeployer` interface let you perform the actual transfer.

Application servers, as well as other enterprise-level tools, may have property pages associated with them, allowing the user to configure the tools from within JBuilder. The `SetupManager` class keeps track of these pages, which are instances of the `Setup` class. Property pages (`SetupPropertyPage`) may appear on a tab of their own in the surrounding dialog, or they may be embedded within another page (`NestingSetupPropertyPage`) on a second set of tabs.

Starting in JBuilder 7, the server API has been restructured to support multiple, independent services from each server. You now extend the `Server`

class to integrate your server with JBuilder, and register it with the `ServerManager` class. Each server supports a number of services, identified by `Service.Type` objects, through implementations of the `Service` class. `ServerLauncher` instances provide the bridge between the services and the runtime environment. The remaining processing is the same as for previous versions.

The application server section of the OpenTools API is extensive; involving many classes and interfaces. Only the main ones are covered in this book. Several of the others are documented in the JBuilder online help; however, many are not described at all.



NOTE

All the application server parts of the OpenTools API come from the JBuilder packages (as opposed to PrimeTime) since they deal with a topic specifically related to Java.

JBoss Example

Throughout this chapter you will see snippets of code from the JBoss application server adapter. The entire code is too large to include here, but is available on the accompanying Web site.

JBoss is an Open Source, standards-compliant, application server implemented in 100% Pure Java and distributed for free. With 150,000+ downloads per month, JBoss is THE most downloaded web-app server, based on and extending beyond the J2EE specification potential.

<http://www.jboss.org>

Under the leadership of Marcus Redeker, a group of developers has written a JBuilder OpenTool to integrate JBoss into JBuilder. This tool is also open source and is available from SourceForge (<http://sourceforge.net/projects/jboss-opentool/>). It currently supports two versions of JBoss: 2.4.x and 3.x.



BIO

Marcus Redeker is currently a Senior Java Consultant at Gedoplan GmbH in Bielefeld, Germany. He is a 'Sun Certified Java Programmer' and has been teaching Java and J2EE technologies for over 3 years.

After receiving his masters in Computer Science and Economics in 1997, Marcus relocated to New York City where he worked as a Senior Consultant for three years, after which he returned to Germany.

Marcus has more than seven years of experience as a professional programmer and is highly proficient in Object-Oriented technologies, Java and Enterprise Java programming.



VERSION

The version presented here is designed for JBuilder 6. In JBuilder 7 and 8, this tool is adapted through classes in the `com.borland.jbuilder.server.legacy` package to allow it to interact with the new server framework.

ServerManager Class

The `com.borland.jbuilder.server.ServerManager` class was introduced in JBuilder 7 to offer more functionality than the earlier `AppServerManager`

one. It allows for individual services to be identified for each server and then to supply those services through different servers within the one project. For example, Tomcat could perform your JSP/servlet processing, while JBoss handles the EJBs.



VERSION

The `ServerManager` class supercedes the `AppServerManager` class. Although the latter class is still supported, the former should be used for new server tools.

The methods (all static) of the `ServerManager` class are listed below:

```
public static synchronized void findServerPathSet(Server
server);
```

Connect a server with a `ServerPathSet` instance through this method. The path set is created as necessary, and appears as `<server name>.library` files in the `.jbuildern` directory in your home directory.

`server` is the server to examine.

```
public static Service findService(Service[] services,
Service.Type serviceType);
```

Search an array of services for one matching a specific type, returning the first instance encountered, or null if none matched.

`services` is the list of services to scan.

`serviceType` is the service type of interest. See `Service.Type` for a list of the standard types.

```
public static Service.Type[] getAvailableTypes();
```

```
public static Service.Type[] getAvailableTypes(boolean
exactMatch);
```

Retrieve a list of all the service types that are provided by at least one of the registered servers, or an empty list if there are none.

`exactMatch` is true (the default) if the type must match precisely, or false if some leeway is possible.



VERSION

Neither of the `getAvailableTypes` methods is available in JBuilder 7. The version that takes a boolean argument is only available in JBuilder 9 and 10.

```
public static List getEnabledServers(boolean addNoneItem);
```

Obtain a list of all those servers that have been enabled through the `Configure Servers` dialog, or an empty list if there are none. The returned list is typically used in a GUI control to allow selection of a server.

`addNoneItem` is true to add a “<None>” item at the end of the list, or false to exclude it. Having this value is useful to indicate that no server has been chosen yet.

```
public static Server getLastRegisteredServer();
```

Find out which server was the last to have registered with this manager.

```
public static SkeletonServer getNoneServerItem();
```

Get a reference to the server object that represents the “<None>” item from `getEnabledServers`. It is a singleton object.

```
public static synchronized Server getPrimaryServer();
```

Identify the server required by JBuilder for certain actions. Currently this is the latest version of the Borland Enterprise Server.

```
public static Server getServer(String fullName);
```

Retrieve a server wrapper via its name, or null if it cannot be found.

fullName is the full name of the desired server. If this matches a legacy server name (see registerLegacyName) then an instance of the new server is returned instead.

```
public static Server[] getServers();
```

```
public static Server[] getServers(String serverTypeId);
```

Obtain a list of all the servers, or only those with a particular server type identifier, currently registered, or an empty array if there are none.

serverTypeId is the server to check.

**VERSION**

The `getServers` method that takes a string argument is only available in JBuilder 9 and 10.

```
public static Service getService(Service.Type serviceType,
    Server server);
```

```
public static Service getService(Service.Type serviceType,
    Server server, boolean exactMatch);
```

Locate a service of a given type for a particular server, or null if no matching item is found.

serviceType is the service type of interest. See `Service.Type` for a list of the standard types.

server is the server to check.

exactMatch is true (the default) if the type must match precisely, or false if some leeway is possible.

**VERSION**

The `getService` method that takes a boolean argument is only available in JBuilder 9 and 10.

```
public static Service[] getServices(Server server);
```

```
public static Service[] getServices(Service.Type
    serviceType);
```

Retrieve a list of all the services for a specific server or service type, or an empty list if there are no matching items.

server or serviceType identify the search criterion for locating the services.

```
public static Service.Type getServiceType(Class
    serviceTypeClass);
```

```
public static Service.Type[] getServiceTypes();
```

```
public static Service.Type[] getServiceTypes(Class
    serviceTypeClass);
```

Get one, several, or all of the service types registered with this manager, or null or an empty list if none can be found.

serviceTypeClass is the class of the service type to search for.

**VERSION**

The `getServiceTypes` method that takes a `Class` argument is only available in JBuilder 9 and 10.

```
public static synchronized void
    registerCustomConfigurationPageFactory(
        CustomConfigurationPageFactory factory, String
        serverName, String serverVersion);
public static synchronized void
    registerCustomConfigurationPageFactory(
        CustomConfigurationPageFactory factory, String
        serverName, String serverVersion, boolean
        registerForAllServersWithSameTypeId);
```

Register a factory against a server to provide custom configuration pages for the Configure Servers dialog. Multiple pages may be supplied through a number of factories.

`factory` is the object that produces the configuration pages. This class is not covered in this book.

`serverName` and `serverVersion` are the full name and version of the server to register the factory with.

`registerForAllServerWithTheSameTypeId` – the name says it all.

**VERSION**

The `registerCustomConfigurationPageFactory` method that takes a `boolean` argument is only available in JBuilder 9 and 10.

```
public static synchronized void registerJdkSupportProvider(
    JdkSupportProvider jdkSupportProvider, String
    serverName, String serverVersion);
public static synchronized void registerJdkSupportProvider(
    JdkSupportProvider jdkSupportProvider, String
    serverName, String serverVersion, boolean
    registerForAllServersWithSameTypeId);
```

Use this method to associate a JDK instance with a server. If such a JDK exists for a server, it is set for the project when switching to that server.

`jdkSupportProvider` is the object that locates the JDK when necessary. This class is not covered in this book.

`serverName` and `serverVersion` are the full name and version of the server to register the JDK against.

`registerForAllServerWithTheSameTypeId` – the name says it all.

**VERSION**

The `registerJdkSupportProvider` method that takes a `boolean` argument is only available in JBuilder 9 and 10.

```
public static void registerLegacyName(String legacyName,
    String currentName);
```

Allow projects using old server versions to be automatically updated to the new version through this method. For example, “Tomcat 3.0” is mapped onto “Tomcat 3.3” in JBuilder 8.

`legacyName` is the name of the old server.

`currentName` is the name of the new server to replace it with.

```
public static void registerServer(Server server);
```

Register a server implementation with the system. Its basic settings are initialized to default values, unless the server has been previously configured in JBuilder, in which case those values are retrieved. Finally, all the services for the server are registered through a call to its `registerServices` method.

`server` is the new server wrapper.

```
public static void registerService(Service service);
```

Tell the system about a new service provided by a server.

`service` is the new service.

```
public static void registerServiceType(Service.Type
serviceType);
```

Inform the system about a service type that may be provided by a server. JBuilder defines the standard types (see `Service.Type` for a list of the standard types), but you may add others as necessary. These appear as entries in the **Services** list in the **Project Properties** dialog, **Server** tab.

`serviceType` is the new service type.

```
public static synchronized void registerTargeting(
AppServerTargeting targeting, String serverName, String
serverVersion);
```

```
public static synchronized void registerTargeting(
AppServerTargeting targeting, String serverName, String
serverVersion, boolean
registerForAllServersWithSameTypeId);
```

Associate an `AppServerTargeting` object (covered below) with a server through this method. The targeter defines special build tasks for a server, and assists in building custom deployment descriptors.

`targeting` is the targeter object to link to the server.

`serverName` and `serverVersion` are the full name and version of the server to register the targeter for.

`registerForAllServerWithTheSameTypeId` – the name says it all.



VERSION

The `registerTargeting` method that takes a boolean argument is only available in JBuilder 9. Neither of the `registerTargeting` methods is available in JBuilder 10.

```
public static synchronized void save();
```

Save all changes for servers made through the **Configure Servers** dialog.

These appear as `<server name>.library` files in the `.jbuildern` directory in your home directory.

```
public static boolean serviceAvailable(Service.Type
serviceType);
```

```
public static boolean serviceAvailable(Service.Type
serviceType, boolean exactMatch);
```

Discover whether any of the registered servers provides a particular service through this method, which returns true if at least one does, or false if none do.

`serviceType` is the service type of interest. See `Service.Type` for a list of the standard types.

`exactMatch` is true (the default) if the type must match precisely, or false if some leeway is possible.



VERSION

Neither of the `serviceAvailable` methods is available in JBuilder 7. The `serviceAvailable` method that takes a boolean argument is only available in JBuilder 9 and 10.

```
public static boolean serviceSupported(Service.Type
    serviceType, Server server);
public static boolean serviceSupported(Service.Type
    serviceType, Server server, boolean exactMatch);
```

Determine whether a server supports a given service type, returning true if it does, or false if it does not.

`serviceType` is the service type of interest. See `Service.Type` for a list of the standard types.

`server` is the server to check.

`exactMatch` is true (the default) if the type must match precisely, or false if some leeway is possible.



VERSION

The `serviceSupported` method is not available in JBuilder 7. The `serviceSupported` method that takes a boolean argument is only available in JBuilder 9 and 10.

This class also defines the fields shown below:

```
public static final String NONE_NAME;
```

This is the name used to indicate no server.

```
public static final Comparator SERVER_COMPARATOR;
```

```
public static final Comparator SERVICE_COMPARATOR;
```

```
public static final Comparator SERVICE_TYPE_COMPARATOR;
```

These comparators let you determine the equality of servers (based on `getFullName` ignoring case), services (based on their service type's `getName` ignoring case), and service types (based on `getName` ignoring case) respectively.

Service.Type Class

The types of service that can be provided by a server are identified by instances of the abstract `com.borland.jbuilder.server.Service.Type` class. These types must be registered with the `ServerManager` through its `registerServiceType` method within the “ServerServices” OpenTools category. New services defined as descendents of `Service` (see below) should include a corresponding `Type` descendent (also called “Type” by convention).

The standard types defined by JBuilder are `ClientJarService.Type`, `ConnectorService.Type`, `DeployService.Type`, `EjbService.Type`, `JspServletService.Type`, `MessageService.Type`, `NamingService.Type`, `SessionService.Type`, and `TransactionService.Type`, all from

the `com.borland.jbuilder.server` package, and `JDataStoreService.Type` from the `com.borland.jbuilder.server.bes` package.

The methods to implement for this class are:

```
public void checkChildNodes(CheckTreeNode serviceTypeNode,
    ServiceCheckTree tree, Project project, boolean
    parentEnabled, Set enabledServices);
```

When the tree that displays type nodes is built, it calls this method to determine if each node needs to have any children. If the given node does need children, add them here as `Service.Type`-based nodes. Typically you would not override this method.

`serviceTypeNode` is the service type node to check.

`tree` is the tree to which any new nodes are added.

`project` is the current project.

`parentEnabled` is true if the parent node is currently enabled, or false if it is not.

`enabledServices` is the collection of currently enabled services for this project.



VERSION

The `checkChildNodes` method is only available in JBuilder 9 and 10.

```
public abstract String[] getFeatureDefinition();
```

Identify the feature (property) that this implementation is associated with. The two array values are the category and property names.



VERSION

The `getFeatureDefinition` method is only available in JBuilder 10.

```
public abstract Icon getIcon();
public abstract String getName();
```

Return the icon or name used to identify this service type. The icon should only be 16 by 16 pixels in size.

```
public abstract String getPropertyKey();
```

Provide a value to be used as the key for any properties of this service.

```
public abstract int getSkuVersion();
```

Indicate which JBuilder edition (using the `UpdateAction` constants) is the minimum required for this service to be enabled.



WARNING

The `getSkuVersion` method was deprecated in JBuilder 10 in favor of the `getFeatureDefinition` method.

```
public boolean isDefaultEnabled();
```

If this service should be enabled by default, then return true (the default). Otherwise return false.

```
public final boolean isEnabledForSku();
```

Determine whether or not this type is enabled for this edition of JBuilder, returning true if it is, or false if it is not.



VERSION

The `isEnabledForSku` method is only available in JBuilder 9 and 10.

```
public abstract boolean isRuntime();
```

Return true if this service runs within the server, or false if it runs as a separate process, such as from the command line.

Service Class

The `com.borland.jbuilder.server.Service` class encapsulates the actual services supplied by a server. These services are associated with a server through its `registerServices` method, which is called as part of the registration of the server itself.



UNDOCUMENTED

Although a page for the `Service` class exists in the documentation, it provides very little detail on the class' methods.

You derive classes specific to your server from the standard (abstract) ones defined by JBuilder – `ClientJarService`, `ConnectorService`, `DeployService`, `EjbService`, `JspServletService`, `MessageService`, `NamingService`, `SessionService`, and `TransactionService`, all from the `com.borland.jbuilder.server` package, and `JDataStoreService` from the `com.borland.jbuilder.server.bes` package.

The methods to implement are:

```
public Service(Server server);
```

Create a new service.

`server` is the server that provides this service.

```
protected void buildFeatureSet();
```

This protected method initializes the `featureSet` field to contain a set of the available features (`getFeatures`).

```
public void configureLauncher(ServerLauncher launcher);
```

Prepare the launcher for this service. The default implementation does nothing.

`launcher` is the launcher instance to be run. This class is covered below.

```
public Feature[] getAllAvailableFeatures();
```

```
public Feature[] getAllAvailableSpecFeatures();
```

Return a list of all the features or all the specification features for this service. By default, `NO_FEATURES` is returned.



VERSION

The `getAllAvailableFeatures` and `getAllAvailableSpecFeatures` methods are only available in JBuilder 9 and 10.

```
public ModuleType getAssociatedModuleType();
```

Provide the `ModuleType` associated with this service, or `null` (the default) if there is no such object. The returned class is not covered in this book, but is documented by Borland.



VERSION

The `getAssociatedModuleType` method is only available in JBuilder 10.

```
public Feature[]
    getAvailableSpecFeaturesForAssociatedModuleType(
        Project project);
```

Return the list of specification features available for the module type associated with this service (if there is one). By default the result of `getAllAvailableSpecFeatures` is returned.

`project` is the current project.

**VERSION**

The `getAvailableSpecFeaturesForAssociatedModuleType` method is only available in JBuilder 10.

```
public String getClientVmParameters();
```

Supply any additional JVM parameters required by the run configurations generated by the Application, Test, and other wizards to allow their generated code to talk to the server. By default it returns `null`, indicating that no extra parameters are necessary.

**VERSION**

The `getClientVmParameters` method is only available in JBuilder 10.

```
public Node getCompanionNode(Node node);
```

Finds the companion node for the one given, being the other of the `EJBGRPFileNode` (parent) or `JarFileNode` (child) given. Extend this in your subclass for other nodes. The default implementation returns `null`.

`node` is the `EJBGRPFileNode` or `JarFileNode` to match.

```
public String getCustomizedRunDebugClassPath(String
    currentClassPath);
```

Update the given class path to allow the service to be debugged. The default implementation just returns the class path provided unchanged.

`currentClassPath` is the class path to customize.

**VERSION**

The `getCustomizedRunDebugClassPath` method is only available in JBuilder 9 and 10.

```
public Feature[] getDefaultSupportedSpecFeatures();
```

Provide the list of specification features that a module using this service supports by default. This implementation returns `NO_FEATURES`, and so should be overridden to supply a more meaningful list.

**VERSION**

The `getDefaultSupportedSpecFeatures` method is only available in JBuilder 10.

```
public ServiceDependency[] getDependencies();
```

Return the list of services upon which this one relies. The default implementation returns `NO_DEPENDENCIES`.

```
protected Feature[] getFeatures();
```

Provide a list of the features supported by this service. By default, `NO_FEATURES` is returned. The `com.borland.jbuilder.server.Feature` class provides a type-safe enumeration and is not otherwise covered in this book.

The features defined by the standard services are shown in Table 28-1.

Table 28-1. Standard service features

** indicates specification features*

Service	Feature
ConnectorService	CONNECTOR_1_0*
	CONNECTOR_1_5*
EjbService	CMP_1x_RELATIONS
	COLLECTION_MULTI_OBJECT_FINDERS
	EJB_1_0*
	EJB_1_1*
	EJB_2_0*
	MESSAGE_DRIVEN_BEANS
	MINIMAL_JARS
	POOL_NAMES
	PRIMITIVE_PRIMARY_KEYS
JspServletService	JSP_1_0*
	JSP_1_1*
	JSP_1_2*
	JSP_1_3*
	JSP_2_0*
	SEARCH_FOR_UNUSED_PORT
	SERVLET_2_1*
	SERVLET_2_2*
	SERVLET_2_3*
	SERVLET_2_4*
	SERVLET_INVOKER
	USER_SPECIFIED_PORT

```
public PropertyPage getProjectPropertiesPage(Project
project);
```

Supply a property page for the project, or `null` (the default) if there are no settings. This page appears on the Server tab in the Project Properties dialog. `project` is the current project.

```
public PropertyPage getRunConfigPropertyPage(
ProjectServerModel model, Map propertyMap);
```

Provide a property page to configure this service, or `null` if there is none (the default). This page appears in the Runtime Configuration Properties dialog accessible from the Run tab in the Project Properties dialog.

`model` contains details about the servers and services attached to a project. This class is not covered in this book.

`propertyMap` is the map of property values (see `MapProperty` in Chapter 7).

```
public Server getServer();
```

Retrieve the server with which this service is associated (set during construction).

```
protected static Service getService(Project project, Class
    serviceTypeClass);
```

```
protected static Service[] getServices(Project project,
    Class serviceTypeClass);
```

Find the service object(s) that provides a particular service within a project. Recall that the user may specify different servers for the various services, so the correct server must first be identified before locating the service itself.

`project` is the project to find the service for.

`serviceTypeClass` is the class of the service required, such as `EjbService.Type`.



VERSION

The `getServices` method is only available in JBuilder 9 and 10.

```
public abstract Service.Type getServiceType();
```

Obtain a reference to the service type for this service, which was created and registered as part of the `initOpenTool` method. This object is an instance of the `Type` class from the abstract `Service` base class, such as `JspServletService.Type` for a JSP/servlet service.

```
public Object[] getServiceTypeKeys();
```

Provide the list of `Service.Type` based objects that can locate this service within a map. By default, a single element array with the value from `getServiceType` is returned. You can provide additional information to make this key unique if necessary.



VERSION

The `getServiceTypeKeys` method is only available in JBuilder 9 and 10.

```
public boolean isGranular();
```

Returns true (the default) if this service can be enabled or disabled separate from other services in the server, or false if it cannot.

```
public void postStart(ServerLauncher launcher);
```

```
public void postStop(ServerLauncher launcher);
```

```
public void preStart(ServerLauncher launcher) throws
    VetoException;
```

```
public void preStop(ServerLauncher launcher);
```

Add processing before or after the service starts or stops. By default nothing extra happens. You may prevent the service from starting by raising a `VetoException` in the `preStart` method.

`launcher` is the launcher instance for this service (covered below).

```
protected static boolean projectUsesService(Project
    project, Class serviceTypeClass, Service service);
```

Determine whether or not a given service is used in a project, returning true if it is, or false if it is not.

`project` is the current project.

`serviceTypeClass` is the type of service being requested.

`service` is the service to check.

**VERSION**

The `projectUsesService` method is only available in JBuilder 9 and 10.

```
public boolean supportsFeature(Feature feature);
```

Return true if this service supports a particular feature, or false if it does not. The default implementation checks for the feature's presence in the `featureSet` field.

`feature` identifies the required feature.

```
public void validate(ServerLauncher launcher) throws
    VetoException;
```

Check that the launcher is properly configured for this service, and throw a `VetoException` if this is not the case. The base implementation does nothing.

`launcher` is the launcher instance to be run.

The fields listed below also appear in this class:

```
protected Set featureSet;
```

The set of features for this service. It is initialized by the `buildFeatureSet` method.

```
protected static final ServiceDependency[] NO_DEPENDENCIES;
protected static final Feature[] NO_FEATURES;
```

These empty arrays may be used as the return values from methods when necessary, rather than creating them each time.

**VERSION**

The `NO_FEATURES` field is only available in JBuilder 7 and 8.

Server Class

The `com.borland.jbuilder.server.Server` class was introduced in JBuilder 7 to offer more functionality than the original `AppServer` one. It allows individual services to be supplied by different servers within the one project. For example, Tomcat could perform your JSP/servlet processing, while JBoss handles the EJBs. It operates in conjunction with `ServerLauncher` (see below), which provides the interface with the JBuilder runtime system for the offered services.

**VERSION**

The `Server` class supercedes (along with the `ServerLauncher` class) the earlier `AppServer` class. Although the latter class is still supported, the former should be used for new server tools.

**NOTE**

The `com.borland.jbuilder.server.legacy.LegacyServer` class provides adapters that allow for the integration of old-style `AppServer` OpenTools into the new server architecture. JBuilder automatically performs this task when the old-style tool is registered.

You register a descendent of this class with the `ServerManager` class through its `registerServer` method. Although there are many methods within this

class, you only need to override a few to implement basic functionality for a new application server: `getDefaultClassPath`, `getDefaultHomeDirectory`, `getDefaultName`, `getDefaultVersion`, `getPackages`, `isValidSetupDirectory`, `newLauncher`, and `registerServices`.

The abilities of this class are:

```
public Server();
```

```
public Server(String name);
```

Create a new application server interface.

`name` is the name of the server supported, including version.

```
protected Url[] addUniquePath(Url[] oldPath, Url url);
```

Updates and returns a set of paths by adding a new one to it, but only if it is not already there.

`oldPath` is the set of paths to modify.

`url` is the new path to add.

```
public boolean attemptDefaultConfiguration();
```

Override this method to try to configure the server, based on the location of JBuilder and the default location of the server installation. Return a flag indicating your success (true) or failure (false). Once configured, this method is no longer called.

```
protected String changeDirectoryReferencesInString(String
currentValue, String currentDirectory, String
newDirectory);
```

Updates a string value to replace old directory references with new ones.

`currentValue` is the text to scan and modify.

`currentDirectory` is the old directory to find. If this value is empty or null, nothing changes.

`newDirectory` is the replacement directory name. If this value is null, nothing changes.

```
public boolean checkSetup(Component host);
```

Use this method to check that the application server has been properly configured, returning true if it has a `CustomConfigurationPage` that has been completed, or false otherwise.

`host` is the host for a message dialog that appears if the server has a `CustomConfigurationPage` but it has not yet been completed. This value may be null.

```
protected void clear();
```

Tidy up this object by releasing any resources that may be held. By default it clears out the path set for the server.



VERSION

The `clear` method is only available in JBuilder 9 and 10.

```
public void clearProjectSettings(JBProject project);
```

This method is called when this server is no longer used by the project. It should remove anything that it specifically added to the project.

`project` is the project to modify.



VERSION

The `clearProjectSettings` method is only available in JBuilder 9 and 10.

```
public void createClientJar(Browser browser, String[]
    paths);
```

Generate the client JAR.

`paths` is the list of class paths to use.



VERSION

The `createClientJar` method is only available in JBuilder 7.

```
public PathSet createClientLibrary(String installPath);
```

Provide a library that references the classes required for a client application to access this server. By default this method just calls `createLibrary`, passing in the results of `getClientLibraryName` and `getClientLibraryClassPath`. Override it only if you need something more.

`installPath` is the directory in which the server is installed.

```
public void createLibrariesFromSetup(String installPath);
```

Call this method from a `CustomConfigurationPage`'s `writeProperties` method to create the libraries used by this application server. Unless overridden, it just calls `createClientLibrary`.

`installPath` is the location where the application server was installed.

```
public PathSet createLibrary(String libraryName, Url[]
    classPathEntries);
```

Creates a new library that references certain paths. If the library already exists, its existing paths are replaced by the supplied set.

`libraryName` is the name of the new library.

`classPathEntries` is a list of paths to include for this library.

```
public static String ensureNonNullValue(String value);
```

Replaces a null string reference with an empty string, while returning a valid string as is.

`value` is the string to check.

```
public void ensureProjectContainsServerClientLibrary(
    JBProject project);
```

This method calls `ensureProjectContainsAppServerLibrary` with parameters to add the client library settings to the project.

`project` is the project to update.

```
public void ensureProjectContainsServerLibrary(String
    libraryName, JBProject project, Url[] classPathEntries);
```

Add the named library to the project, placing it at the top of the list. This library contains the classes that provide the functionality of this application server. If it does not exist, it is created using the supplied classpath values. If the application server has not yet been set up, the library's name is added to the project, but it is not created.

`libraryName` is the name of the library for the application server files (usually the server's short name).

`project` is the project to update.

`classPathEntries` is the list of paths that make up the library.

```
public static String formatJarFileParameter(String
    jarFile);
```

Ensures that the name of a JAR file is properly represented for the current operating system, for example, surrounding names with embedded spaces in quotes. The corrected name is returned.

`jarFile` is the name of the JAR file.

```
public JDKPathSet getAssociatedJdk();
```

An alternate JDK may be connected with the application server (see `getJdkSupportProvider`), in which case its paths are returned. If none has been specified, the default JDK for JBuilder is used.

```
public Url[] getClassPath();
```

Retrieves the classpath for this application server.

```
public ClientJarService getClientJarService();
```

Supply a reference to the client JAR service for this server, or `null` if there is none.

```
public Url[] getClientLibraryClassPath(String installPath);
```

Returns the list of paths for the classes necessary for client access to the server, or `null` (the default) if there are none. Find the name of this library with the `getClientLibraryName` method.

`installPath` is the location where the application server was installed.

```
public String getClientLibraryName();
```

Return the name of the library that contains classes necessary for clients to communicate with the server. By default this returns the server's short name with version plus "Client". Find this library's classpath with the `getClientLibraryClassPath` method.

```
public Node getCompanionNode(Node node);
```

Finds the matching node for that given, being the other of the `EJBGRPFileNode` (parent) or `JarFileNode` (child) given. Extend this in your subclass for other nodes.

`node` is the `EJBGRPFileNode` or `JarFileNode` to match.

```
protected Server getCopy();
```

Return a copy of this server that has the same behavior. By default this method just creates a new server of the current type and copies the path set into it.

**VERSION**

The `getCopy` method is only available in JBuilder 9 and 10.

```
public CustomConfigurationPageFactory
    getCustomConfigurationPageFactory();
```

Supply the factory for a configuration page for this server. The resulting page appears in the Configure Servers dialog.

```
public abstract Url[] getDefaultClassPath();
public abstract Url getDefaultHomeDirectory();
public abstract String getDefaultName();
public abstract String getDefaultServerName();
public abstract String getDefaultVersion();
```

Provide the default settings for various properties. See the corresponding methods in the `AppServer` class for more details.

```
public DeployService getDeployService();
public EjbService getEjbService();
```

Give a reference to the deployment or EJB service for this server, or null if there is none.

```
public String getExtraVisiBrokerToolParameters();
```

Use this method to return additional parameters to be passed to various VisiBroker tools, such as java2iiop. The default implementation returns “”.



VERSION

The `getExtraVisiBrokerToolParameters` method is not available in JBuilder 7.

```
protected String getFileNameBasedOnProtocol(String
    protocol, String name);
```

Formats a file name as it appears for a particular file system implementation.

`protocol` is the name of the Filesystem protocol to use (see Chapter 5).

`name` is the file name to update.

```
public Url[] getFullClassPath();
```

Returns the complete classpath for this application server as a set of paths.

```
public Url[] getFullLibraryClassPath(String libraryName,
    Url[] necessaryClassPath);
```

Gets the full list of paths for a given library. If it does not exist, the library is created with the given paths, and then returned.

`libraryName` is the name of the library to locate.

`necessaryClassPath` is the minimal set of classpath entries required for this library.

```
public String getFullName();
```

Obtains the full name (name and version) for this application server.

```
public Url getHomeDirectory();
```

Discovers where the application server was installed. The returned directory is the base of that installation.

```
public String getIncompleteDescription();
```

Returns a string indicating that this application server has not yet been fully configured: “XXX has not been configured.”

```
public JdkSupportProvider getJdkSupportProvider();
```

Obtains a reference to an alternate JDK to use when running this application server, or null if none was specified. The returned class is not described in this book.

```
public JspServletService getJspServletService();
```

Supply a reference to the JSP/servlet service for this server, or null if there is none.

```
public long getLastModified();
```

Finds out the version stamp for the paths of the server. Note that this is a sequential number and not a timestamp.

```
public String getLegacyFullName();
```

Retrieves an older name previously used for this server, or null if there was none.

```
protected ArrayList
    getLibraryNamesToClearOnResetToDefaults();
```

This method is not currently used.

**VERSION**

The `getLibraryNamesToClearOnResetToDefaults` method is only available in JBuilder 9 and 10.

```
public String getName();
```

Returns the library name for this application server.

```
public static String getNameFromFullName(String fullName);
```

Extracts the name part of the application server's full name (see the `makeFullName` method).

`fullName` is the full name for the application server, including version.

```
public Url[] getNewPathsBasedOnNewHomeDirectory(String
    newHomeDirectory, Url[] paths);
```

Updates all the paths to reflect a new base directory. Any existing entries beneath the old home directory are relocated to the new one, while other paths have their first directory entry altered to the new value. The actual classpath for the server remains unchanged. Use `updateClassPathWithNewHomeDirectory` to affect the real classpath.

`newHomeDirectory` is the full path to the new base directory.

`paths` is the list of `Urls` to modify.

```
public static String getNodeValue(NodeProperty property,
    ArrayList nodes);
```

Determines whether a property setting is the same for a set of nodes. If all nodes have the same value for the property, that value is returned. In all other cases an empty string results (never `null`).

`property` is the property to retrieve.

`nodes` is a list of `Node` objects to examine.

```
public String[] getOptimizerPackages();
```

Provide the list of packages not to trace in the optimizer. You may include a trailing asterisk as a wildcard, such as `"org.apache.tomcat.*"`. The default is the result of `getPackages`.

**VERSION**

The `getOptimizerPackages` method is not available in JBuilder 7.

```
public abstract String[] getPackages();
```

Supply the list of packages not to trace in the debugger. You may include a trailing asterisk as a wildcard, such as `"org.apache.tomcat.*"`.

```
public ServerPathSet getPathSet();
```

Obtains the path set for this server. The returned class is not covered in this book; however, it extends `PathSet` (see Chapter 9).

```
public String getServerTypeId();
```

Provide a unique string to identify a type of server plugin. By default this is just the full class name.

**VERSION**

The `getServerTypeId` method is only available in JBuilder 9 and 10.

```
public Service[] getServices();
```

Return a list of the services provided by this server. By default, it queries the `ServerManager` for services registered against this server.

```
public ServerLauncher getSetupLauncher();
```

This method provides a shared instance of the `ServerLauncher` for this server. It is used during configuration, but not at runtime.

```
public String getShortName();
```

```
public String getShortNameWithVersion();
```

Gets the short name, with or without version information, for the server. By default, the former returns `getName`, while the latter returns this plus the version. Override these if you want something else. For example, the former returns “JBoss 2.4.x” for the JBoss tool.

```
public Url[] getUniqueRunDebugClassPath(Url[]
checkClassPath);
```

Adds the current classpath to the one given and filters out duplicates.

`checkClassPath` is the classpath to add to.

```
public Url getUrl();
```

Returns the path to the library for this application server.

```
public String getVersion();
```

Discovers the version of this server.

```
public String getVisiBrokerConfigurationName();
```

Provide a value that can be used to identify a `VisiBroker` configuration for a BES server plugin. By default it returns `null` if `usesVisiBrokerOrb` is false, or “`VisiBroker`” and the name of this server if true.



VERSION

The `getVisiBrokerConfigurationName` method is only available in JBuilder 9 and 10.

```
public int getWeight();
```

Supply the weight value (defaulting to 50) for the `ProjectPropertyPage-Listener` for this server.



VERSION

The `getWeight` method is only available in JBuilder 10.

```
public boolean hasClientJarCreator();
```

```
public boolean hasEjbDeployer();
```

Indicate whether or not this server provides the given functionality, returning true if it does, or false if it does not



VERSION

The `hasClientJarCreator` and `hasEjbDeployer` methods are only available in JBuilder 7.

```
public boolean hasSetup();
```

Return true if this server needs to be configured in the `Configure Servers` dialog before it can be used, or false if it can be used immediately.

```
protected void initialize();
```

Prepare this server for use.

**VERSION**

The `initialize` method is only available in JBuilder 10.

```
protected boolean isCopy();
```

Indicate whether or not this server is a copy. By default this method checks its underlying path set.

**VERSION**

The `isCopy` method is only available in JBuilder 9 and 10.

```
public boolean isGranular();
```

Return true if the server can enable or disable individual services, or false (the default) if it cannot.

```
public boolean isIncomplete();
```

Indicate whether this server has been properly configured yet, returning true if not, or false if it is ready to go. By default it checks `hasSetup` and `isSetup`.

```
public boolean isInitiallySetup();
```

If this server can be successfully auto-configured then return true and the server can be immediately enabled. Otherwise return false (the default). This method is called as part of `attemptDefaultConfiguration`.

**VERSION**

The `isInitiallySetup` method is only available in JBuilder 10.

```
public boolean isPageValid(ProjectServerModel model);
```

Indicate whether or not the current state of the model is valid (the default is true), allowing the associated property page to close.

`model` provides access to the original, current, and immediately prior selections. This class is not covered in this book.

**VERSION**

The `isPageValid` method is only available in JBuilder 10.

```
public boolean isServerEnabled();
```

Discover whether this server has been enabled in the Configure Servers dialog.

```
public boolean isSetup();
```

Returns true (the default) if the application server is configured and ready to run, or false if it requires further work. Compare this with `isSetupCompleted`.

```
public boolean isSetupCompleted();
```

Discovers whether the configuration for this application server has been run during this session, returning true if it has, or false (the default) if it has not. Even if this method returns true, the server may not be ready to run as it may require a restart of JBuilder. This value resets to false when JBuilder is restarted.

```
public abstract String isValidSetupDirectory(String  
    setupDirectory);
```

Provide a description as to why the given setup directory is invalid, or null if it is acceptable.

`setupDirectory` is the directory used to set up the server. In most cases this is the home directory of the server.

```
public boolean libraryExists(String libraryName);
```

Returns true if the given library already exists, or false if it does not.

`libraryName` is the name of the library to locate.

```
public static String makeFullName(String name, String version);
```

Combines the name and version of an application server into a single value. Use `getNameFromFullName` to obtain the first part again.

`name` and `version` are the separate identifying values for an application server.

```
public String makeServerLibraryName(String libraryNameSuffix);
```

Generate a library name for the server that combines the short server name (`getShortNameAndVersion`) and unique suffix, such as “WebLogic 6.1 Client”. If the server is a copy (`isCopy` returns true), then the name derives from the full server name (`getFullName`) and unique suffix.

`libraryNameSuffix` is the unique part of the name, e.g. “Client”.



VERSION

The `makeServerLibraryName` method is only available in JBuilder 9 and 10.

```
public String makeServerToolName(String toolNameSuffix);
```

Create a name for a tool to appear on the JBuilder menu, allowing it to be uniquely identified. Combine the full server name (`getFullName`) with the given tool name, e.g. “WebLogic 6.1 Admin Console”.

`toolNameSuffix` is the unique name for the tool, e.g. “Admin Console”.



VERSION

The `makeServerToolName` method is only available in JBuilder 9 and 10.

```
public void modifyProjectLibraryList(String libraryName, JBProject project, boolean add);
```

Adds or removes a library from the list of those required for a project. When adding, the library is placed at the top of the project’s list, or is moved there if already present.

`libraryName` is the name of the library to add or remove.

`project` is the project affected.

`add` is true to add the library, or false to remove it.

```
public abstract ServerLauncher newLauncher();
```

Create a new `ServerLauncher` for this server to run a service. Always return launchers of the same class.

```
public void preServiceSelectionChangedAndSaved(ProjectServerModel model);
```

React to changes to the set of services for a project through this method. It is called just before they are changed and saved.

`model` provides access to the original, current, and immediately prior selections. This class is not covered in this book.

**VERSION**

The `preServiceSelectionChangedAndSaved` method is only available in JBuilder 10.

```
public abstract void registerServices();
```

This method is called as the final part of the server registration process. It allows the server to register its own services. This avoids cluttering the `classes.opentools` file for your tool with a myriad of service classes. Your code would look like the following:

```
ServerManager.registerService(new MyJspServletService(this));
```

```
public void save();
```

Writes the server's settings out to disk.

```
public void serverModified(Server server);
```

React to changes to the properties of the server through this method. The default method does nothing.

`server` is the server affected.

```
public String serverRelativePath(String  
jbuilderRelativePath);
```

Adjust a path that is relative to JBuilder's installation directory to one that is relative to the server's installation.

`jbuilderRelativePath` is the JBuilder-based path.

```
public void serviceSelectionChanged(ProjectServerModel  
model);
```

```
public void serviceSelectionChangedAndSaved(  
ProjectServerModel model);
```

Respond to changes in the application server assigned to a project by overriding these methods. The "saved" version ensures that the old server's client library is removed and the new one is added. Be sure to call the inherited versions if you override either of these methods.

`model` contains details about the servers and services attached to a project. This class is not covered in this book.

`project` is the project to which they apply.

```
public void setClassPath(Url[] classPath);
```

Updates the classpath for this application server.

`classPath` is the list of paths to use.

```
public void setCustomConfigurationPageFactory(  
CustomConfigurationPageFactory factory);
```

Establish the factory for configuration pages for this server.

`factory` is the new configuration page factory.

```
public void setHomeDirectory(Url homeDirectory);
```

Updates the base location of the application server.

`homeDirectory` is the base directory where the server was installed.

```
public void setJdkSupportProvider(JdkSupportProvider  
jdkSupportProvider);
```

Specifies an alternate JDK to use when running this application server.

`jdkSupportProvider` identifies the JDK to use. This class is not covered in this book.

```
public void setPathSet(ServerPathSet pathSet);
public void setPathSetFromCopy(ServerPathSet copy);
```

Replaces or copies the application server's paths.
`pathSet` is the new set of paths to use.
`copy` is the set of paths to copy.

```
public void setServerEnabled(boolean enabled);
```

Mark the server as being enabled from the Configure Servers dialog.
`enabled` is `true` to enable the server, or `false` to disable it.

```
public void setSetupCompleted(boolean setupCompleted);
```

Call this method to change whether the configuration page for this server has been run. See `isSetupCompleted` for more details.
`setupCompleted` is `true` if it has been run, or `false` if it has not.

```
public void setVersion(String version);
```

Updates the version for this application server.
`version` is the new value. It must not contain any spaces.

```
public void setWeight(int weight);
```

Establish the weight value for the `ProjectPropertyPageListener` for this server.
`weight` is the new weight.

VERSION

The `setWeight` method is only available in JBuilder 10.

```
public boolean supportsCopy();
```

Indicate whether or not this server may be copied – a new instance can be created with different user settings. If `true`, the Copy button in the Configure Servers dialog is enabled. To be copyable, the server must satisfy a large number of conditions, including using `ServerProperty`s rather than `GlobalProperty`s, and providing unique names for each copy. See the online help for a complete list.

VERSION

The `supportsCopy` method is only available in JBuilder 9 and 10.

```
public boolean supportsCreateClientJar();
```

Return `true` to indicate that this server can create a JAR file that provides access to the EJBs from a client, or `false` (the default) if it does not.

```
public boolean supportsJavaRunnableEjbContainer();
```

Indicate whether or not this server can be run under Java.

VERSION

The `supportsJavaRunnableEjbContainer` method is only available in JBuilder 7.

```
public void updateClassPathWithNewHomeDirectory(String newHomeDirectory);
```

Updates the classpath entries to reflect a new base directory. Those entries beneath the old home directory are relocated to the new one. Use `getNewPathsBasedOnHomeDirectory` to find the new paths without modifying the actual classpath.

`newHomeDirectory` is the full path to the new base directory.

```
public void updateLastModified();
```

Increments the modified counter for the underlying path set.

```
protected void updateProjectClientSettings(JBProject project);
```

Called from `updateProjectSettings`, this method lets you update client configurations in the project. The default implementation does nothing.

`project` is the project whose configuration is updated.



VERSION

The `updateProjectClientSettings` method is only available in JBuilder 10.

```
public void updateProjectSettings(JBProject project,
    boolean updateRunConfigurations);
```

Changes in the server's settings flow through to open projects via this method. You can then apply them to the nodes in the project as necessary. The default implementation ensures that the project's JDK is set to any specified by the server, and that the project contains the client library.

`project` is the project whose configuration is updated.

`updateRunConfigurations` is true to call the `updateRunConfigurations` method before updating the project, or false to skip it.



VERSION

The `updateProjectSettings` method is not available in JBuilder 7.

```
public void updateRunConfigurations(JBProject project);
```

When the run configurations for the server need to be updated, this method is called. If you override this method be sure to call the inherited version.

`project` is the project whose run configurations are updated.



VERSION

The `updateRunConfigurations` method is not available in JBuilder 7.

```
public boolean usesVisiBrokerOrb();
```

Return true to indicate that the server uses the VisiBroker ORB, or false (the default) to show that it does not. This setting affects which JDKs are added to the `orb.properties` file.

These fields also appear in this class:

```
public static final int DEFAULT_WEIGHT;
```

The default weight for a server with respect to `ProjectPropertyPageListeners`.



VERSION

The `DEFAULT_WEIGHT` field is only available in JBuilder 10.

```
public static final String INVALID_SERVER;
```

Returned when there is no matching server.

```
public static final int NEW_SERVER_WEIGHT;
```

.

**VERSION**

The `NEW_SERVER_WEIGHT` field is only available in JBuilder 10.

```
public static final String NO_NAME;
```

The value returned when the server name is not known.

```
public static final Comparator WEIGHTED_COMPARATOR;
```

Compare servers based on their weights through this field.

**VERSION**

The `WEIGHT_COMPARATOR` field is only available in JBuilder 10.

ServerLauncher Class

A `Server` provides instances, via its `newLauncher` method, of descendents of the abstract `com.borland.jbuilder.server.ServerLauncher` class to allow its services to be run within JBuilder. Each launcher is associated with a runtime tracker that pipes the process' output back to a message panel.

Although there is only one `Server` instance, and one of each of its `Services`, there could be many launchers connected to them. You descend your launchers from this class and implement the methods that invoke the required service.

**UNDOCUMENTED**

Although this class is present in the documentation, many of its methods and all of its fields remain undocumented.

The abilities of this class are listed below:

```
public ServerLauncher(Server server);
```

Create a new launcher and associate it with a server.

`server` is the server that created the launcher.

```
public void addSourceBridge(NjplSourceBridge sourceBridge);
```

Add a new source bridge to this launcher – a handler for non-Java-programming-language translators. This class is not otherwise covered in this book.

`sourceBridge` is the bridge to add.

```
protected void appendHttpPort(StringBuffer buf);
```

If a servlet server descriptor exists for the launcher (in its property bag), append “ `http:`” and its port number to the supplied buffer. This does not seem to be very useful!

`buf` is the buffer to update.

```
public boolean canStop();
```

Indicate whether the server can be stopped, returning true if it can, or false if it cannot. The default implementation checks for `getStopper` being non-null.

**VERSION**

The `canStop` method is not available in JBuilder 7.

```
public void clearExceptionQueue();
```

Delete any queued exceptions.

```
public void clearSourceBridges();
```

Remove all source bridges.

```
public void configureLauncher(JBProject project, Map
    propertyMap, RunJavaProcessTracker tracker, Url workDir)
    throws VetoException;
```

Prepare to run by configuring the launcher. The base class copies the parameters into internal fields, and finds the required services from the property map (using `ServerRunner.getEnabledServices()`). A `VetoException` should be thrown if the server cannot launch.

`project` is the project using the server.

`propertyMap` is the collection of properties that the server was started with.

`tracker` is the object that redirects the server's output to `JBUILDER`.

`workDir` is the working directory for the server.

```
public void configureLauncher(JBProject project, Service[]
    services);
```

Configure this launcher for a set of services to simulate a run. The services are not validated and the `propertyMap`, `tracker`, and `workDir` fields are all set to null.

`project` is the project using the server.

`services` is the list of services to configure. This value should not be null, should not be empty, nor have any duplicate entries.

```
protected void configureServices();
```

Calls `configureLauncher` for each service to be run.

```
public String customizeArguments(String currentArguments);
```

Update the parameters for the application server if necessary, just before they are passed to it. You may return an empty string, but should not return a null. By default, the current parameters are simply passed back.

`currentArguments` is the current set of parameters for the server.

```
public String customizeClassPath(String currentClassPath);
```

Perform a final customization of the server classpath, just before it runs. The returned value should not be null, but may be an empty string. The default version simply returns the path supplied.

`currentClassPath` is the current path to be used.

```
public PathSet[] customizeLibraries(PathSet[] pathSets);
```

Modify the libraries in a run configuration that can be deployed to the server. The default implementation removes the server's client library and the project's `Servlet` library.

`pathSets` is the list of paths in the project.

```
public String customizeTransportAddress(String
    transportType, String transportAddress);
```

Update the transport address used for this launcher before it is used. By default, the current address is passed back unchanged.

`transportType` is the current type.

`transportAddress` is the current address.

**VERSION**

The `customizeTransportAddress` method is not available in JBuilder 7.

```
public String customizeVmParameters(String
    currentVmParameters);
```

Customize the parameters for the server's VM just before they are used. You may return an empty string, but should not return a `null`. By default, the current parameters are simply passed back.

`currentVmParameters` is the current set of parameters for the VM.

```
protected void deployLibraries();
```

Copy all required libraries to the server using the `deployLibrary` method.

```
protected void deployLibrary(PathSet library);
```

For each path in the library, call `deployLibraryEntry` to copy it to the server.

`library` is the library to copy.

```
protected void deployLibraryEntry(Url source);
```

Copy a directory or file to the server.

`source` is the item to be copied.

```
protected String escapeParameter(String arg);
```

Check a value for embedded spaces, and surround the entire text with quotes if found. Otherwise, just return the supplied value.

`arg` is the original text.

```
public String[] getArchivesToDeployOnRun();
```

Define the archives to be deployed when the server starts. By default the routine returns the value of the `ServerRunner.ARCHIVES` property if the server supports this functionality. Return the list of archives to deploy on start up, or `null` if this feature is not supported.

```
public String getArguments();
```

Retrieves the user-specified parameters for the application server.

```
public abstract String getCommand();
```

Construct the complete command-line to run the server through this launcher and return it.

```
public Url getCurrentWorkingDirectory();
```

Find the current directory for starting the server.

```
public abstract String getDefaultArguments();
```

```
public abstract String getDefaultNecessaryArguments();
```

```
public abstract String getDefaultNecessaryVmParameters();
```

```
public abstract String getDefaultVmParameters();
```

Return the default values for the suggested or required server parameters, or the suggested or required VM parameters. None of the parameters should return as `null`, but may be empty strings. The necessary parameters are not displayed to the user and are always prepended to the values sent to the server when it is run.

```
public Url getDefaultWorkingDirectory();
```

Supply the specific working directory needed for the server to run. If this is returned as `null` (its default implementation), a directory with the name of the server is created beneath the project's working directory.

```
public String[] getEnvironment();
```

Provide a custom environment for the server runtime through this method. If it returns `null` (the default), then the environment that exists for JBuilder is used.

```
public static String[] getEnvpWithPathVariablePrefix(
    String pathPrefix);
```

Modifies the path in the current environment by prepending it with the value supplied. It returns `null` if no environment variables exist.

`pathPrefix` is a list of paths to prepend to the existing classpath.

```
public Object getFromBag(Object key);
```

Find a value identified by its key, or `null` if it cannot be found.

`key` is the value that identifies a particular setting.

```
protected static File getJavaLauncher(JBProject project);
```

Retrieve a reference to the Java runtime application.

`project` is the project whose JDK is used.

```
public String getLabel();
```

Concatenates the server short name with version and any HTTP port (see `appendHttpPort`) to return as the label for this launcher.

```
protected String getLibraryClassesRelativePath();
```

Supply the relative path to the classes for this launcher, defaulting to `../classes`.

```
public Url getLibraryDestination();
```

The purpose of this method is unclear, although it defaults to `null`.

```
public String getNecessaryArguments();
```

```
public String getNecessaryVmParameters();
```

Provide the parameters required to run the server or its JVM.

```
public JBProject getProject();
```

Find the project that this launcher is running.

```
public Url[] getProjectLibrariesForRun(JBProject project);
```

Provides the list of paths that make up the classpath for the application server. Generally this is the entire set of libraries assigned to the project. You should call the inherited version in your subclass to obtain the initial set of paths to work with.

`project` is the project whose classpath is needed. It may be `null`.

```
public Map getPropertyMap();
```

Obtain the set of properties established for this launcher.

```
public Exception[] getQueuedExceptions();
```

Return the list of queued exceptions for this server.

```
public PropertyPageFactory[] getRunConfigPropertyPages(
    Server server, ProjectServerModel model, Map
    propertyMap);
```

Provide the list of property pages for configuring this server. You can call upon the `standardRunConfigPropertyPages` method to return the default set of pages, as is done in the base implementation.

`server` is the server being configured.

`model` contains information about the project.

`propertyMap` holds the current service selections.

```
public Server getServer();
```

Access the server to which this launcher belongs, as set at creation.

```
public Service getService(Service.Type serviceType);
```

Find and return a service based on its type, or null if no match is found.
serviceType is the type of the required service.

```
public Service[] getServices();
```

Obtain a list of all the services configured for this launcher.

```
public int getShutdownWaitTime();
```

Return the time (in seconds) to wait for the server to shutdown (defaulting to 5).

```
public NjplSourceBridge getSourceBridge(String extension);
```

Locate a source bridge for a particular file extension and return it, or null if no match is found. The returned class is not covered in this book.
extension is the extension to search for.

```
public NjplSourceBridge[] getSourceBridges();
```

Access the list of source bridges for the launcher.

```
protected Stopper getStopper();
```

Obtain the stopper thread object for this server. This method is called within the stop method to assist in halting the server. By default it returns null.

```
public RunJavaProcessTracker getTracker();
```

Access the tracker for this launcher – it redirects output back to JBuilder.

```
public String getVmParameters();
```

Retrieves the user-specified parameters for the application server's JVM.

```
protected Thread getWaitForServerThread(WaitsForServer waitron);
```

Find the thread that waits for the server to be ready.
waitron provides a link to check on the server's readiness.

```
public Url getWorkingDirectory();
```

Finds out the working directory for the application server – where the server is started from.

```
public String getWorkingDirectoryFromHomeDirectory(String homeDirectory);
```

Based on a home directory, return the name of the working directory that should be used, or null (the default) if no special directory is required.
homeDirectory is the base location for the application server.

```
protected void init();
```

Override this method to perform any initialization processing. It is called by initLauncher following clearing of internal values, and before configuring the services.

```
protected void initLauncher(Service[] services);
```

This method saves the list of requested services, clears out internal settings, and then prepares to run through a call to configureLauncher for the set of services. The method is itself called by the other version of configureLauncher once the services are validated.

When invoking the services' methods, they are processed in the order of the array during their configureLauncher, preStart, and postStart calls, but in reverse order during the preStop and postStop calls.

services is the set of services to prepare for. This value should not be null, not be empty, nor have any duplicate entries.

```
public String isValidWorkingDirectory(String
    workingDirectory, String homeDirectory);
```

Verifies that a particular directory is appropriate as the working directory for this server (see `getDefaultWorkingDirectory`), returning a null (the default) if it is valid, or an error message if it is not.

`workingDirectory` is the name of the directory to check.

`homeDirectory` is the base directory for the application server.

```
public void postStart();
```

The default version of this method waits for the server to start, and then calls `postStart` on each of its services. You can override this behavior if something else is required following start up.

```
public void postStop();
```

After the server has terminated, but before the UI regains control, this method is called. The default implementation calls each service's `postStop` method. Override it to perform any other tasks necessary at this time.

```
public boolean preStart();
```

This launcher can perform any startup tasks before the server actually starts by overriding this method. Return true if the server can be started, or false if the run should be aborted.

```
protected boolean preStartServices();
```

Attempt to prestart all the services by calling their `preStart` methods. Return true if none threw a `VetoException`, or false if any of them did.

```
public void preStop();
```

Just prior to the server terminating, this method is called. The default implementation calls each service's `preStop` method. Override it to perform any other tasks necessary at this time.

```
public void putInBag(Object key, Object value);
```

Save a keyed value for this launcher.

`key` is the identifying object for the setting.

`value` is the new value of that setting.

```
protected void queueException(Exception ex);
```

Add a new exception to the list of those queued for the server.

`ex` is the new exception.

```
public void restoreFrom(Server server);
```

Copies the main settings from an application server adapter to this launcher.

`server` is the instance to copy from.

```
public void setArguments(String arguments);
```

Alters the parameters passed to the application server when it is run or debugged.

`arguments` is the new set of parameters for the server.

```
public void setNecessaryArguments(String arguments);
```

```
public void setNecessaryVmParameters(String parameters);
```

Modifies the set of parameters required for the application server or its JVM to run. These are not accessible through the GUI, and are inserted before any user-specified parameters.

`arguments` or `parameters` is the new set of required parameters.

```
public void setVmParameters(String parameters);
```

Alters the parameters passed to the application server's JVM when it is run or debugged.

`parameters` is the new set of parameters for the JVM.

```
public void setWorkingDirectory(Url workingDirectory);
```

Modifies the location from which to run the application server.

`workingDirectory` is the new directory to start it in.

```
protected List standardRunConfigPropertyPages(Server
server, ProjectServerModel model, Map propertyMap);
```

Use this method to obtain the list of standard run configuration pages for a launcher (such as from `getRunConfigPropertyPages`). These include one for the server itself, and optional ones for the Libraries and Archives topics.

`server` is the server being configured.

`model` contains information about the project.

`propertyMap` holds the current service selections.

```
public boolean stop();
```

React to the user terminating the server through this method. You can then perform any clean up tasks that are necessary. Return true if this method actually shutdown the server, or false if it did not.

```
public boolean supportsClearDeployedArchivesBeforeRun();
```

When this method returns true, a checkbox appears on the Server run page under the Archives topic, allowing the user to indicate whether they want to remove any existing archives before deploying a new set. By default the routine returns false.

```
public void trackerClosed();
```

Override this method to respond to the closure of the tracker for this server. By default it does nothing.

```
protected void updateLastModified();
```

Increments the modified counter of the server.

```
protected String useVmParameter(StringBuffer vmParams,
String name, String defaultValue);
```

Find the value of a parameter for the server's VM through this method. It searches the parameters for a given name and returns its value. If not found, the parameter is added with a default value, and that default is returned instead.

`vmParams` is the current list of VM parameters.

`name` is the name of the parameter to check for.

`defaultValue` is the value to add for the parameter if it cannot be found.

```
public void validateServices(JBProject project, Service[]
services) throws VetoException;
```

Check that this launcher can run the requested services. You should not make any permanent changes to the launcher at this time. By default the launcher ensures that the array is not empty and then calls `validate` on each entry. A `VetoException` is thrown if the services cannot be run.

`project` is the project using this server.

`services` is the list of services to check.

The following fields are also available for your descendents of `Server-Launcher`:

`protected Map bagMap;`

The map that contains settings for the launcher – accessible through the `getFromBag` and `putInBag` methods.

`protected Exception[] exceptions;`

The list of queued exceptions.

`protected JBPProject project;`

The project for this launcher, as set through `configureLauncher`.

`protected Map propertyMap;`

The map of properties for the launcher, as set through `configureLauncher`.

`protected Service[] services;`

The services provided by this launcher, as set through `initLauncher`.

`protected NjplSourceBridge[] sourceBridges;`

The array of source bridges for the launcher.

`protected RunJavaProcessTracker tracker;`

The tracker for the launcher, as set through `configureLauncher`.

`protected Url workDir;`

The current working directory.

AppServerTargeting Class

Assisting in the build process when deploying to a particular application server is the `com.borland.jbuilder.enterprise.ejb.AppServerTargeting` class. It lets you hook into the build process surrounding the JAR file for your EJBs, as well as identify additional properties that may be required by each server implementation.



WARNING

This class no longer appears in JBuilder 10.

Its methods are:

```
public void antify(AntExporter antExporter, EJBGRPFileNode
    ejbGrpNode);
```

Update an Ant build file to create the EJB JAR file for the given group. The default implementation adds an Ant `fail` task. Override this method to add tasks that generate any stub or skeleton files, and then package up the contents into a deployable JAR file.

`antExporter` is the wrapper for the Ant build file. It is not covered in this book.

`ejbGrpNode` is the EJB group being examined.



VERSION

The `antify` method is only available in JBuilder 9.

```
protected Element createAntJarTask(AntExporter antExporter,
    EJBGRPFileNode ejbGrpNode, Url targetUrl);
```

```
protected Element createAntTmpJarTask(AntExporter
    antExporter, EJBGRPFileNode ejbGrpNode);
```

Add an Ant task to the given build file that generates a temporary JAR file. The methods return the Ant target element to which the task was appended. Usually you would call this from your `antify` method and then update that JAR to make it fully deployable.

`antExporter` is the wrapper for the Ant build file.

`ejbGrpNode` is the EJB group being examined.

`targetUrl` is location of the JAR file to create.



VERSION

The `createAntJarTask` and `createAntTmpJarTask` methods are only available in JBuilder 9.

```
public EjbPropertyElement[] getEjbJarProperties(
    EJBGRPFileNode ejbGrpNode);
```

Obtain an array of vendor-specific properties for the EJB group as a whole so that the user can update them. Return `null` if the property viewer should not appear at all. This class is not discussed in this book, though it is documented in the online help.

`ejbGrpNode` is the EJB group being examined.

```
public EjbPropertyElement[] getEjbProperties(EjbReference
    ejbRef);
```

Get an array of vendor-specific properties relating to a given EJB so that the user can amend them. Return `null` if you do not want the property viewer to show at all.

`ejbRef` is a reference to an EJB. It may be `null`. This class is also not covered in this book. It gives you details about the EJB (like `getJndiName` and `isSessionBean`) and access to the Java classes that provide its various interfaces (such as `getBeanNode` and `getHomeNode`). Unfortunately this class is undocumented.

```
protected Url getJarTarget(EJBGRPFileNode ejbGrpNode);
```

Supply the location of the JAR file to create.

`ejbGrpNode` is the EJB group being examined.



VERSION

The `getJarTarget` method is only available in JBuilder 9.

```
public String getPageName(AppServer appServer);
public String getPageName(Server server);
```

Retrieve the name of the property page for a server.

`appServer` and `server` identify the application server to look for.



WARNING

The `getPageName` method that takes an `AppServer` parameter has been deprecated.

```
public static Server getServer(Project project);
```

Find the server associated with the EJB service for a project.

project is the project being examined.



VERSION

The `getServer` method is only available in JBuilder 9.

```
public boolean hasEjbJarProperties(EJBGRPFileNode
    ejbGrpNode);
    Return true if there are JAR-level properties to be shown, or false if there are
    none.
    ejbGrpNode is the EJB group being examined.
public boolean hasEjbProperties(EjbReference ejbRef);
    Return true if this EJB has any properties to display, or false if there are
    none.
    ejbRef is a reference to an EJB. It may be null.
public void postProcessBuild(BuildProcess buildProcess,
    EJBGRPFileNode ejbGrpNode, JarFileNode jarNode);
    Following the build processing, implement any extra processing required.
    buildProcess is the process that performed the build (see Chapter 25).
    ejbGrpNode is the EJB group file node being built.
    jarNode is the newly built JAR file.
public void preProcessBuild(EJBGRPFileNode ejbGrpNode);
    Perform any additional process before the main build process starts.
    ejbGrpNode is the EJB group file node being built.
public void updateBuildTask(StubsBuildTask task,
    EJBGRPFileNode ejbGrpNode, EjbReference[] ejbRefs);
    Add a task to the build process to create stubs for the given node if
    necessary. This task runs after the EJB files are compiled and before the JAR
    file is generated.
    task is the build task to update if required. This class (com.borland.
    jbuilder.enterprise package) is not covered in this book, but it extends
    BuildTask (see Chapter 25) and adds the addBuildElement method for
    your use.
    ejbGrpNode is the EJB group file node being built.
    ejbRefs is an array of references to the EJBs in this group.
public DeploymentDescriptor[]
    updateDeploymentDescriptors(DeploymentDescriptor[] dds,
    EJBGRPFileNode ejbGrpNode, BuildReport report);
public DeploymentDescriptor[]
    updateDeploymentDescriptors(DeploymentDescriptor[] dds,
    EJBGRPFileNode ejbGrpNode, boolean forceUpdate,
    BuildReport report);
public EarDeploymentDescriptor[]
    updateDeploymentDescriptors(EarDeploymentDescriptor[]
    edds, EarGrpFileNode earGrpNode, BuildReport report);
    Update the EJB or EAR group file node during the build process. The
    methods return an array of deployment descriptors to include in the JAR file,
    or null if none apply.
    dds and edds are the arrays of deployment descriptors to examine.
```

`ejbGrpNode` and `earGrpNode` are the EJB or EAR group file nodes being built.

`report` is where to send any warnings and error messages. This class is not covered in this book, but you would call its `addBuildError`, `addBuildMessage`, or `addBuildWarning` methods depending on the severity of the problem encountered. The class is undocumented.

`forceUpdate` is true to clear out any cached descriptors, or false (the default) to leave them.



WARNING

The first `updateDeploymentDescriptors` method above has been deprecated in favor of the second version.

```
public void updateVerifyReport(EJBGRPFileNode ejbGrpNode,
    JarFileNode jarNode, BuildReport report);
```

Read the contents of the JAR file and validate them. This occurs following the build processing and before `postProcessBuild` is called.

`ejbGrpNode` is the EJB group file node being built.

`jarNode` is the JAR file to examine.

`report` is where to send any warnings and error messages.

```
public void verifyDeploymentDescriptors(MessageView mv,
    MessageCategory mc, DeploymentDescriptor[] dds,
    EJBGRPFileNode ejbGrpNode);
```

```
public void verifyDeploymentDescriptors(MessageView mv,
    MessageCategory mc, EarDeploymentDescriptor[] edds,
    EarGrpFileNode earGrpNode);
```

Verify the supplied deployment descriptors and send any warnings or error messages to the Message Pane. This method is called when the Verify button is pressed in the EJB DD Editor tab. Listings 28–1 and 28–2 contain examples from the JBoss tool.

`mv` is a reference to the Message Pane to send notifications to.

`mc` is the message category to use when sending messages.

`dds` and `edds` are the arrays of deployment descriptors to verify.

`ejbGrpNode` and `earGrpNode` are the EJB or EAR group file nodes being reviewed.

Listing 28–1. JBossTargeting24.verifyDeploymentDescriptors.

```
public void verifyDeploymentDescriptors(MessageView messageview,
    MessageCategory messagecategory, DeploymentDescriptor
    adeploymentdescriptor[], EJBGRPFileNode ejbgrpfilenode) {
    try {
        JBossDescriptorConversion24 ddConversion =
            new JBossDescriptorConversion24(
                adeploymentdescriptor, ejbgrpfilenode, null);
        ddConversion.verifyDeploymentDescriptors(
            messageview, messagecategory);
    }
    catch(Exception exception) { }
}
```

Listing 28–2. *JBossDescriptorConversion24.verifyDeploymentDescriptors.*

```

public void verifyDeploymentDescriptors(MessageView messageview,
    MessageCategory messagecategory) throws Exception {
    DateFormat dateformat = DateFormat.getDateTimeInstance(2, 1);
    try {
        super.report = new BuildReport();
        messageview.addMessage(messagecategory, "");
        String s = dateformat.format(new Date());
        s = Strings.format("Maybe error??", s);
        messageview.addMessage(messagecategory, s);
        convert(false, true);
        BuildMessage abuildmessage[] = super.report.getAllMessages();
        if(abuildmessage != null) {
            for(int k = 0; k < abuildmessage.length; k++) {
                if(abuildmessage[k].isError()) {
                    s = Strings.format("Error", abuildmessage[k].getMessage(),
                        String.valueOf(abuildmessage[k].getLineNumber()));
                    continue;
                }
                if(abuildmessage[k].isWarning()) {
                    s = Strings.format("Warning", abuildmessage[k].getMessage(),
                        String.valueOf(abuildmessage[k].getLineNumber()));
                    messageview.addMessage(messagecategory, s);
                }
            }
        }
        s = Strings.format("no error", String.valueOf(0),
            String.valueOf(0));
        messageview.addMessage(messagecategory, s);
    }
    catch(Exception exception) {
        throw new AssertionError(exception.getMessage());
    }
}

```

AbstractDeploymentDescriptor Class

An individual deployment descriptor is encapsulated by the abstract `com.borland.jbuilder.enterprise.AbstractDeploymentDescriptor` class. It manages the name and location of the descriptor within the JAR file, as well as its contents.

The methods of this class are listed below:

`public byte[] getBytes();`

Supply the content of this descriptor as a byte array.

`protected abstract GrpFileAccess getDefaultAccessor();`

Define a default read/write interface for the descriptors. This class is not documented nor is it covered in this book. It only has two methods: `readDescriptors` and `writeDescriptors`, which take an `InputStream` and `OutputStream` respectively.



VERSION

The `getDefaultAccessor` method is no longer abstract in JBuilder 9.

`public String getEncoding();`

Extract the encoding from the XML content of this descriptor.

`public String getExtraLocation();`

Discover any additional directory structure for this descriptor. See `setExtraLocation` for more information.

```
public GrpFileAccess getFileAccessor();
```

Obtain a reference to the read/write interface for this EJB group. Use the default one if no other is specified.

```
public String getName();
```

Find out the name for this descriptor, such as “ejb-jar.xml”, or null if it is not yet set.

```
public long getTimestamp();
```

Retrieve the timestamp for this descriptor, or -1 if not yet set.

```
public void setBytes(byte[] newBytes);
```

Amend the content of the descriptor.
newBytes is the replacement text for the descriptor.

```
public void setExtraLocation(String extraLocation);
```

Update any additional directory structure, beneath the normal META-INF directory, required by this descriptor. For example, if this setting is “deploy” then the descriptor appears in the “META-INF/deploy” directory.
extraLocation is the extra path information. If this is null, it is ignored and any previous value is retained. It defaults to “”.

```
public void setFileAccessor(GrpFileAccess accessor);
```

Establish the read/write interface for this descriptor.
accessor is the object to use when reading or writing the descriptor.

```
public static void setFileAccessor(
    AbstractDeploymentDescriptor[] dds, GrpFileAccess
    accessor);
```

Apply a file accessor to several descriptors at once.
dds is the array of descriptors to update.
accessor is the read/write object to use for each.

```
public void setName(String newName);
```

```
public void setTimestamp(long newTimestamp);
```

Update the name or timestamp for this descriptor.
newName and newTimestamp are the new values to use.

```
public String toString();
```

```
public String toString(String encoding) throws
    UnsupportedEncodingException;
```

Convert the descriptor’s content into a string value.
encoding is the name of the encoding to use during the conversion. It defaults to the value derived from the content itself.

DeploymentDescriptor Class

As a concrete extension of AbstractDeploymentDescriptor, com.borland.jbuilder.enterprise.ejb.DeploymentDescriptor can be used in your applications.



WARNING

The DeploymentDescriptor class has been deprecated in JBuilder 10 in favor of the XMT framework, which is not covered in this book (see XmtReader and XmtWriter in the online help).

The methods of this class are as follows (omitting those that are the same as the parent class):

```
public DeploymentDescriptor();
public DeploymentDescriptor(String name);
public DeploymentDescriptor(String name, long timestamp);
public DeploymentDescriptor(String name, long timestamp,
    byte[] bytes);
public DeploymentDescriptor(String name, String
    extraLocation, long timestamp);
public DeploymentDescriptor(DeploymentDescriptor dd);
    Create a new descriptor object.
    name, timestamp, bytes, and extraLocation set the corresponding
    fields within this object.
    dd is an existing descriptor to clone.
protected GrpFileAccess getDefaultAccessor();
    Returns an accessor suitable for EJB group files (.ejbgrp extension).
public static DeploymentDescriptor[]
    getDeploymentDescriptors(EJBGRPFileNode ejbGrpNode);
    Retrieve all the descriptors from an EJB group, or null if there are none.
    ejbGrpNode is the EJB group node to examine.
```



VERSION

The `getDeploymentDescriptors` method is not available in JBuilder 10.

```
public static String getFileEncoding(EJBGRPFileNode node);
    Discover the encoding used for the XML content of a given EJB group node.
    node is the node of interest.
```



VERSION

The `getFileEncoding` method is only available in JBuilder 8.

```
public String getFullName();
    Find the full path for this descriptor within the JAR file: made up from
    "META-INF", extra location, and name.
```

AbstractDescriptorConversion Class

The `com.borland.jbuilder.enterprise.ejb.AbstractDescriptorConversion` class manages the creation and validation of deployment descriptors for a target application server. It is typically used internally by an `AppServerTargeting` class to support its functionality.



WARNING

This class no longer appears in JBuilder 10.

This class' methods are described below:

```
public AbstractDescriptorConversion(
    DeploymentDescriptor[] dds, EJBGRPFileNode ejbGrpNode,
    BuildReport report);
    Create a new conversion class.
    dds is the array of descriptors to work with from the EJB module.
    ejbGrpNode is the EJB module.
```

`report` is the destination for any errors or warnings. It may be `null`.

```
protected void addErrorMessage(String message);
protected void addWarningMessage(String message);
```

Append a message to the report object supplied during construction, or to standard error or output if none was specified.

`message` is the text to display.

```
public DeploymentDescriptor[] convert(boolean update,
    boolean msgOnError) throws Exception;
public DeploymentDescriptor[] convert(boolean update,
    boolean msgOnError, boolean ignoreTimestamps) throws
    Exception;
```

This is the main method of this class as it generates the descriptors for the application server and optionally writes them to the EJB module. It returns the array of descriptors generated and updated. Usually you would not override this method, but would just implement the ones that it calls: `generate` and `patchForAppServer`.

`update` is true to update the EJB module with the changes, or false to not update it.

`msgOnError` is true to generate messages when errors occur, or false to not produce them.

`ignoreTimestamps` is true to ignore timestamps and always write the regenerated descriptors to the EJB module, or false (the default) to check the timestamps before updating.



WARNING

The first `convert` method is deprecated in favor of the second version.

```
protected boolean createInterfaceFromDescriptors(boolean
    msgOnError);
```

Creates the internal reference to the EJB descriptor, `iEjbJar`, during the `convert` method processing. Normally you would not override nor call this method.

`msgOnError` is true to write any errors to the report object provided during construction, or false to not write them out.

```
protected DeploymentDescriptor findDescriptor(
    DeploymentDescriptor[] dds, String name);
```

Locate a particular descriptor from a list and return it, or `null` if not found.

`dds` is the array of descriptors to search.

`name` is the name of the one to find.

```
protected abstract void generate(ArrayList ddList) throws
    Exception;
```

Create any non-standard descriptors and add them to the list. The standard `ejb-jar.xml` entry is already in the list. Throw an exception if a fatal error is encountered. The `convert` method calls this one.

`ddList` is the list of descriptors to update.

```

public static IJdbc1DataSource getBeanDataSource(
    IAccessibleBean bean, IEjbJar iEjbJar);
    Find and return a datasource reference for a bean, or null if it cannot be
    located. None of the classes used here are described in this book.
    bean is the bean that uses the datasource.
    iEjbJar is the JAR descriptor.
public static String getBeanDataSourceName(
    IAccessibleBean bean);
    Discover the JNDI name for a bean's datasource from its properties, or null
    if none is found.
    bean is the bean that uses the datasource.
protected Document getDescriptorDocument(String
    fullDescriptorName, ArrayList ddList);
    Retrieve a descriptor as an XML document (org.w3c.dom.Document), or
    null if it is not located.
    fullDescriptorName is the name of the descriptor to return.
    ddList is the list of descriptors to search.
protected static JavaFileNode getEjbNode(JBProject project,
    String ejbClass);
    Find the node for a fully qualified class name if it lies on the source path and
    return it, or null if it cannot be found.
    project is the project to use the source path from.
    ejbClass is the full class name.
protected String getGenericDDName();
    Provide the name of the generic descriptor for this server. It defaults to
    "ejb-jar.xml".
protected static JotMethod[] getJotMethods(JavaFileNode
    node);
    Extract the public methods from a Java node, or an empty array if none were
    found or there is a syntax error in the file.
    node is the Java source node to examine.
protected JBProject getProject();
    Retrieve the project for this EJB module, or null if no EJB module was
    specified during construction.
public static EjbPropertyElement[]
    getPropertyNamesToSurface(EjbReference ejbRef);
    Return an array of property elements for vendor-specific properties, or null
    if you do not want the property editor to appear for the user. The returned
    class is not described in this book, but is listed in the online help.
    ejbRef is a reference to the EJB being reviewed. It may be null. This class
    is not discussed in this book.
protected Server getServer();
    Retrieve the server for the EjbService for the descriptor's project, or null
    if it cannot be found.

```

**VERSION**

The `getServer` method is only available in JBuilder 9.

```
protected boolean isEjbJarDirty();
protected boolean isModuleDirty();
```

See whether the EJB JAR contents have been modified. They return false unless changed by the corresponding setter method.



VERSION

The `isEjbJarDirty` method is deprecated in JBuilder 8 and up in favor of `isModuleDirty`, which is not available in JBuilder 7.

```
protected boolean needToUpdate(DeploymentDescriptor[]
    ddsNew);
```

Based on the descriptor names in the supplied array, determine whether any of these are missing or old (based on their timestamp) and return true in that case. Otherwise return false.

`ddsNew` is the array of descriptors to check.

```
protected void patchForAppServer(DeploymentDescriptor[]
    ddsRet);
```

Called by `convert` just before returning the vendor-specific descriptors, this method lets you perform any post-processing following a successful conversion. Any changes made here are saved with the EJB module.

`ddsRet` is the array of descriptors converted.

```
protected void setEjbJarDirty(boolean ejbJarDirty);
protected void setModuleDirty(boolean moduleDirty);
```

Note that the contents of the EJB JAR have been altered.

`ejbJarDirty` or `moduleDirty` is true to indicate that changes have been made, or false to reset the flag.



VERSION

The `setEjbJarDirty` method is deprecated in JBuilder 8 and up in favor of `setModuleDirty`, which is not available in JBuilder 7.

```
protected void updateEjbModule(DeploymentDescriptor[]
    ddsNew) throws Exception;
```

Add or replace the supplied descriptors in the list for this EJB module.

`ddsNew` is the array of descriptors to be merged.

```
public abstract void verifyDeploymentDescriptors(
    MessageView mv, MessageCategory mc) throws Exception;
```

Validate the deployment descriptors obtained during construction and send any messages to the specified area of the Message Pane. See an example from the JBoss tool in Listing 28–8.

`mv` is the Message Pane to write to.

`mc` is the category under which the messages appear.

These protected fields are available to your subclass:

```
protected DeploymentDescriptor ddGeneric;
```

A reference to the specification-compliant descriptor, usually “`ejb-jar.xml`”.

```
protected DeploymentDescriptor[] dds;
```

The descriptors extracted from the EJB module.

```
protected long descriptorDate;
```

The timestamp of the descriptors.

```
protected DescriptorPair descriptorPair;
```

The EJB specification compliance level – one of the constants from DescriptorList.

```
protected EJBGRPFileNode ejbGrpNode;
```

The EJB module that contains the descriptors.

```
protected boolean ejbJarDirty;
```

```
protected boolean moduleDirty;
```

Indicates whether anything accessible through iEjbJar below have been changed, affecting the EJB module.



VERSION

The `ejbJarDirty` field is deprecated in JBuilder 8 and up in favor of `moduleDirty`, which is not available in JBuilder 7.

```
protected IEjbJar iEjbJar;
```

Access to the live Borland descriptors.

```
protected BuildReport report;
```

The destination for errors and warnings. It may be null if no output is wanted.

SetupManager Class

Serving as a repository for all the Setup objects (see below) within JBuilder is the `com.borland.jbuilder.ide.SetupManager` class. Its methods are all static, as shown below:

```
public static void checkShowRestartWarning();
```

Determine whether the restart warning screen should be shown, and show it if necessary. Afterwards it resets the `showRestartWarning` flag to false.

```
public static synchronized Setup getSetup(String name);
```

```
public static synchronized Setup getSetup(String name,
String parentName);
```

Obtain a reference to a particular setup, or null if it cannot be found.

`name` is the name of the desired setup.

`parentName` is the name of the parent setup for which the required one is a child. If not specified, a setup at the top-most level is located.

```
public static Personality[] getSetupPersonalities();
```

Discover the personalities (see Chapter 10) applicable to Setup objects.



VERSION

The `getSetupPersonalities` method is only available in JBuilder 10.

```
public static synchronized Setup[] getSetups();
```

```
public static synchronized Setup[] getSetups(String
parentName);
```

Retrieve a list of all the registered setups, or an empty array if none are found.

`parentName` is the name of a parent setup. If specified, all the setups registered as sub-pages of this one are returned.

```
public static boolean isShowReopenWarning();
```

Find out whether a reopen warning should be shown – returning true if so, or false if not.



VERSION

The `isShowReopenWarning` method is only available in JBuilder 9 and 10.

```
public static boolean isShowRestartWarning();
```

Returns true if the restart warning should be shown, or false if it should not.

```
public static synchronized void registerSetup(Setup setup);
public static synchronized void registerSetup(Setup setup,
String parentName);
```

Register a setup with the manager.

`setup` is an instance of the new setup class.

`parentName` is the name of another setup to which this one belongs. If not specified the setup is registered at the top-most level. Otherwise it becomes a sub-page of the nominated parent. No error results if the parent does not exist, allowing for the parent and child to be registered in either order. Only one level of sub-pages is allowed.



VERSION

In JBuilder 7 and up the setup pages for application servers have been placed into a new dialog, invoked from the Tools menu as Configure Servers. To make your server configuration pages appear here instead of in the Enterprise Setup dialog, specify `AppServerSetup.APPSERVER_SETUP_NAME` (from the `com.borland.jbuilder.enterprise.ejb` package) as the parent name for your new Setup class.

```
public static void setReopenWarningMessage(String message);
public static void setRestartWarningMessage(String
message);
```

Add extra text to be displayed in a reopen or restart warning dialog.

`message` is the text to display.



VERSION

The `setReopenWarningMessage` and `setRestartWarningMessage` methods are only available in JBuilder 9 and 10.

```
public static void setShowReopenWarning(boolean show);
```

Update whether or not a reopen warning is shown when necessary. If both this and `setShowRestartWarning` are set to true, only the restart warning is shown.

`show` is true to present it, or false to hide it.



VERSION

The `setShowReopenWarning` method is only available in JBuilder 9 and 10.

```
public static void setShowRestartWarning(boolean show);
```

Alter whether the restart warning is displayed. Call this with a value of true in your setup to indicate that JBuilder should show a warning to the user to restart JBuilder to complete the setup process. It is displayed when they close the dialog.

`show` is true to present it, or false to hide it.



WARNING

Setting this value to false may suppress the warning even if other setups have requested it.

Setup Class

The abstract `com.borland.jbuilder.ide.Setup` class lets you supply custom setup pages for enterprise-level tools by extending it and registering an instance with the `SetupManager`. These pages appear in the dialog brought up by the **Tools | Configure Servers** menu item. Each one occupies its own tab, although that may be at the top-most level within the dialog, or embedded within another page. The difference comes from the choice of property page returned by this class, and by the registration process.

The methods of this class are listed here:

```
public PropertyPage createSetupPage(Browser browser);
```

Provide an instance of the setup page. This is just a public wrapper around the `getSetupPropertyPage` method.

`browser` is the current browser.



VERSION

The `createSetupPage` method is only available in JBuilder 9 and 10.

```
public abstract String getName();
```

This is the name of the setup, both for identification purposes in the `SetupManager`, and as the text on its tab in the dialog. Implement this in your subclass.

```
public PropertyPageFactory getPageFactory(Browser browser);
```

A default factory is included in this class, producing pages by calling the `getSetupPropertyPage` method.

`browser` is the current browser.

```
public Personality[] getPersonalities();
```

Return the list of personalities to which this setup applies (see Chapter 10).



VERSION

The `getPersonalities` method is only available in JBuilder 10.

```
protected abstract PropertyPage getSetupPropertyPage(
    Browser browser);
```

Override this in your subclass to supply a property page for the tool. Note that the returned page must implement `SetupPage` (see below). Using `SetupPropertyPage` for single-level pages, or `NestingSetupPropertyPage` for pages that contain other pages as a basis for your pages makes this easier.

`browser` is the current browser.

```
public PropertyPage initializeSetupPage(Browser browser,
    PropertyPage page);
```

This method updates a `SetupPage` with a description at the top, and the panel provided by the `createSetupPanel` method of the page at the

bottom. It returns `null` if the provided page is not a descendent of `SetupPage`.

`browser` is the current browser.

`page` is the `SetupPage` to initialize.



VERSION

The `initializeSetupPage` method is only available in JBuilder 9 and 10.

```
public abstract boolean isEnabled();
```

Returns true if the setup page should be displayed, or false if it should not appear at all. Supply this method in your subclass.

```
public boolean isShowDescription();
```

Determine whether or not the description of the property page should be shown, returning true to display it, or false to not show it.

```
public void setShowDescription(boolean showDescription);
```

Control whether the property page description is displayed.

`showDescription` is true to display it, or false to hide it.

Two constants also appear in this class:

```
public static final String CATEGORY;
```

The name of the global property category for this part of the API.

```
public static final GlobalProperty
```

```
LAST_SELECTED_SETUP_NAME;
```

The name of the last property page visited.

JBossSetup24 Example

The Setup class from the JBoss tool is shown in Listing 28–3.

Listing 28–3. JBossSetup24 class.

```
/*
 * Voyager JBoss OpenTool for JBuilder
 *
 * Copyright (c) 2002 Marcus Redeker, Gedoplan GmbH - Bielefeld,
 * Germany    marcus.redeker@gedoplan.de
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * Cyrille Morvan's excellent 'Kelly JOnAS OpenTool' sources provided
 * much of the inspiration and some of the code for this OpenTool.
 */
package de.gedoplan.opentools.jboss;

import com.borland.primetime.properties.PropertyPage;
import com.borland.primetime.ide.Browser;

import com.borland.jbuilder.ide.Setup;
import com.borland.jbuilder.ide.SetupManager;
```

```

import com.borland.jbuilder.enterprise.ejb.AppServerSetup;
import com.borland.jbuilder.info.JBuilderInfo;

import java.io.File;

/**
 * JBossSetup returns the property page which is displayed under menu
 * Tools/Enterprise Setup
 *
 * @author <a href="mailto:marcus.redeker@gedoplan.de">Marcus Redeker
 (Gedoplan GmbH, Germany)</a>
 * @version 1.2
 */
public class JBossSetup24 extends Setup {

    public JBossSetup24() {
    }

    /** JB init */
    public static void initOpenTool(byte byte0, byte byte1) {
        SetupManager.registerSetup(
            new JBossSetup24(), AppServerSetup.APPSERVER_SETUP_NAME);
    }

    /** get the property page. */
    protected PropertyPage getSetupPropertyPage(Browser aBrowser) {
        return new JBossSetupPropertyPage24(aBrowser);
    }

    /** display name */
    public String getName() {
        return JBossResource.getDefaultName24();
    }

    /**
     * JBoss OpenTool is enable all the time !
     * (if Enterprise is enabled !).
     */
    public boolean isEnabled() {
        return JBuilderInfo.isEntEnabled();
    }

    public String validateInstallDirectory(String s) {
        return JBossSetupPropertyPage24.validateInstallDirectory(s);
    }

    public void saveSettings(String s) {
        JBossSetupSettings24 jbossSetupSettings =
            new JBossSetupSettings24();
        jbossSetupSettings.setInstallPath(s);
        jbossSetupSettings.setConfigName(
            JBossPropertyGroup.CONFIGURATION_NAME_24.getValue());
        JBossSetupPropertyPage24.saveJBossSettings(jbossSetupSettings);
    }
}
// eof

```

SetupPage Interface

The basic abilities of pages that appear in the Configure Servers dialog are defined by the `com.borland.jbuilder.ide.SetupPage` interface. Normally you would use the `SetupPropertyPage` or `NestedSetupPropertyPage` classes (described below) that already implement this as the basis for your own property pages.

Its required methods are:

```
public JPanel createSetupPanel();
```

Provide a panel that contains the controls that interact with your tool's properties. As this interface is implemented on top of the `PropertyPage` class (see Chapter 7), you initialize and persist your settings in the `readProperties` and `writeProperties` methods.

```
public String getDescription();
```

Return a general description of the page and its options. This text is displayed at the top of the page.

SetupPropertyPage Class

When adding a custom property page for your tool, you typically subclass `com.borland.jbuilder.ide.SetupPropertyPage`. This abstract class extends `PropertyPage` (see Chapter 7) and implements `SetupPage`. Pages derived from this appear on their own tabs, though that may be at the top-most level in the dialog or within another page. For pages that contain sub-pages you should use the `NestingSetupPropertyPage` class discussed below.

Initialize your UI as part of the `readProperties` method and save any changes in the `writeProperties` one. These pages are returned from a `Setup` object through its `getSetupPropertyPage` method. The `Setup` object in turn is registered with the `SetupManager` to make it available.

Its new methods (over `PropertyPage`) are shown below:

```
public SetupPropertyPage(Browser browser);
```

Create a new property page for setting up enterprise-level tools.

`browser` is the browser to associate with this page.

```
public abstract JPanel createSetupPanel();
```

Provide a panel object that contains the UI to embed on a tab in the dialog.

Since this class derives from `JPanel`, you could return `this`.

```
protected Browser getBrowser();
```

Returns the browser set during construction of this object.

```
public abstract String getDescription();
```

Supply a description of this page to be displayed at the top of the page.

```
public JBProject getProject();
```

Returns the active project at the time of creation if it is a `JBProject` instance, or null if it is not.

```
public abstract boolean isModified();
```

Implement this method to indicate if any changes have been made that require action when the dialog is closed. Return true if changes were made, or false if nothing was altered.

NestingSetupPropertyPage Class

To allow other pages to be embedded in your property page in the `Setup` dialog, you should derive your page from `com.borland.jbuilder.ide.NestingSetupPropertyPage`. Another abstract class, this one extends `NestingPropertyPage` (see Chapter 7) and implements `SetupPage`.

You register the `Setup` object that generates your parent class with `SetupManager`, and then register the `Setup` object for each sub-page as well, supplying the name of the parent `Setup` in the process.

Its methods are as follows:

```
public NestingSetupPropertyPage(Browser browser, Setup
    setup);
```

Create a new property page that manages a set of sub-pages for setting up enterprise-level tools.

`browser` is the browser to associate with this page.

`setup` is the object creating this page.

```
public JPanel createSetupPanel();
```

Constructs a panel that contains the description at the top and a tabbed pane at the bottom, filled with the `SetupPropertyPages` registered under the `Setup` that created this page.

```
public abstract String getDescription();
```

Supply a description of this page to be displayed at the top of the page.

Summary

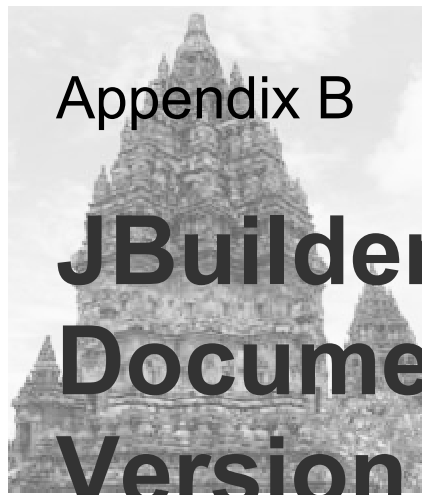
Interfacing with an application server through the OpenTools API allows you to integrate that server into JBuilder. You can then configure it, modify the deployment descriptors it needs, deploy applications to it, and run or debug them, all from within the JBuilder IDE.

There are a lot of classes and interfaces involved in this section of the API and only the main ones have been discussed here. Several of those mentioned here are documented in JBuilder's online help, although many are not.

The JBoss example used throughout this chapter demonstrates how to go about creating an application server adapter and registering it with JBuilder. Look through the full code on the accompanying Web site to gain a deeper understanding of its functions.



Appendices



Appendix B

JBuilder Documentation Version Differences

Differences in the OpenTools API arise as each new version of JBuilder is released. This is reflected in the major and minor version numbers for the OpenTools API as shown in Table B–1. Recall that JBuilder versions prior to 7 are not covered in this book.

As well as changes to the packages and classes present in the API, the documentation of these items also changes. Once a section of the API is documented it needs to remain fairly static in future versions, so that tools have a reasonable expectation of working there. Thus, much of the OpenTools API remains undocumented, due to both lack of time and to possible changes as the API evolves. To help come to grips with all these changes Table B–2 is presented.

Table B–1. OpenTools versions

JBuilder version	OpenTools version	
	Major	Minor
4	4	1
5	4	2
6	4	3
7	4	4
8	4	5
9	4	6
10	4	7

The columns under each version indicate whether a class is defined in that version (☑) and whether it is documented in that version (📖). Only the classes and interfaces (in italics) that are documented in at least one version of JBuilder are shown below. For classes that are not listed, the documentation status of that class remains the same throughout the JBuilder versions, either described in all versions or in none. Items marked “?” for the documentation have references to the appropriate HTML page, but I couldn’t reach it in my copy of JBuilder.













































































Table B–2. OpenTools API class differences

Class	JB 7	JB 8	JB 9	JB 10
com.borland.jbuilder.build				
<i>Antifiable</i>	–	–	☑📖	☑📖
ArchiveOutputEvent	–	☑📖	☑📖	☑📖
BuildMessage	☑	☑	☑📖	☑📖




















Class	JB 7	JB 8	JB 9	JB 10
EarOutputEvent	–	☑	☑	–
EjbOutputEvent	–	☑	☑	–
ExternalProcessTask	☑	☑	☑	☑
OutpathDirectoryDeletedEvent	–	☑	☑	☑
PackageDeletedEvent	–	☑	☑	☑
<i>ParseError</i>	☑	☑	☑	☑
com.borland.jbuilder.enterprise				
BasicDeploymentDescriptor	–	–	–	☑
com.borland.jbuilder.enterprise.descriptor				
Domains	–	–	–	☑
com.borland.jbuilder.enterprise.descriptor.application.wrapper				
ApplicationClientModuleWrapper	–	–	–	☑
ApplicationWrapper	–	–	–	☑
CommonModuleWrapper	–	–	–	☑
ConnectorModuleWrapper	–	–	–	☑
EjbModuleWrapper	–	–	–	☑
WebModuleWrapper	–	–	–	☑
com.borland.jbuilder.enterprise.descriptor.client.wrapper				
ApplicationClientWrapper	–	–	–	☑
com.borland.jbuilder.enterprise.descriptor.common				
BaseNodeType	–	–	–	☑
com.borland.jbuilder.enterprise.descriptor.common.wrapper				
CommonEjbRefWrapper	–	–	–	☑
DescribedWrapper	–	–	–	☑
DisplayableWrapper	–	–	–	☑
EjbLocalRefWrapper	–	–	–	☑
EjbRefWrapper	–	–	–	☑
EnvEntryWrapper	–	–	–	☑
IdentifiedWrapper	–	–	–	☑
ResourceEnvRefWrapper	–	–	–	☑
ResourceRefWrapper	–	–	–	☑
RunAsWrapper	–	–	–	☑
SecurityPermissionWrapper	–	–	–	☑
SecurityRoleRefWrapper	–	–	–	☑
SecurityRoleWrapper	–	–	–	☑

Class	JB 7	JB 8	JB 9	JB 10
com.borland.jbuilder.enterprise.descriptor.connector.wrapper				
AuthenticationMechanismWrapper	—	—	—	☑
ConfigPropertyWrapper	—	—	—	☑
ConnectorWrapper	—	—	—	☑
LicenseWrapper	—	—	—	☑
ResourceAdapterWrapper	—	—	—	☑
com.borland.jbuilder.enterprise.descriptor.ejb.wrapper				
CmpFieldWrapper	—	—	—	☑
CmrFieldWrapper	—	—	—	☑
ColumnMappingWrapper	—	—	—	☑
CommonAccessibleEjbWrapper	—	—	—	☑
CommonAssemblyMethodWrapper	—	—	—	☑
CommonAssemblyMethodWrapper.	—	—	—	☑
CommonAssemblyMethodNode				
DisplayNameResolver				
CommonEjbWrapper	—	—	—	☑
CommonFieldWrapper	—	—	—	☑
CommonMethodWrapper	—	—	—	☑
CommonMethodWrapper.	—	—	—	☑
CommonMethodNodeDisplayNameResolver				
ContainerTransactionMethodWrapper	—	—	—	☑
ContainerTransactionWrapper	—	—	—	☑
EjbJarWrapper	—	—	—	☑
EntityBeanWrapper	—	—	—	☑
ExcludeListMethodWrapper	—	—	—	☑
ExcludeListWrapper	—	—	—	☑
FinderWrapper	—	—	—	☑
MessageDrivenBeanWrapper	—	—	—	☑
MethodPermissionMethodWrapper	—	—	—	☑
MethodPermissionWrapper	—	—	—	☑
QueryMethodWrapper	—	—	—	☑
QueryWrapper	—	—	—	☑
QueryWrapper.	—	—	—	☑
QueryNodeDisplayNameResolver				
RelationshipRoleSourceWrapper	—	—	—	☑
RelationshipRoleWrapper	—	—	—	☑
RelationshipWrapper	—	—	—	☑
SecurityIdentityWrapper	—	—	—	☑
SessionBeanWrapper	—	—	—	☑

Class	JB 7	JB 8	JB 9	JB 10
TableReferenceWrapper	—	—	—	☑
com.borland.jbuilder.enterprise.descriptor. war.wrapper				
AuthConstraintWrapper	—	—	—	☑
CommonParamWrapper	—	—	—	☑
ContextParamWrapper	—	—	—	☑
ErrorPageWrapper	—	—	—	☑
FilterMappingWrapper	—	—	—	☑
FilterWrapper	—	—	—	☑
FormLoginConfigWrapper	—	—	—	☑
InitParamWrapper	—	—	—	☑
ListenerWrapper	—	—	—	☑
LoginConfigWrapper	—	—	—	☑
MimeMappingWrapper	—	—	—	☑
SecurityConstraintWrapper	—	—	—	☑
ServletMappingWrapper	—	—	—	☑
ServletWrapper	—	—	—	☑
SessionConfigWrapper	—	—	—	☑
TagLibWrapper	—	—	—	☑
UserDataConstraintWrapper	—	—	—	☑
WebAppWrapper	—	—	—	☑
WebResourceCollectionWrapper	—	—	—	☑
WelcomeFileListWrapper	—	—	—	☑
com.borland.jbuilder.enterprise.ejb				
AbstractDescriptorConversion	☑	☑	☑	—
AbstractDescriptorImporter	☑	☑	☑	—
AppServerTargeting	☑	☑	☑	—
<i>DeployConstants</i>	☑	☑	☑	—
EjbPropertyElement	☑	☑	☑	—
com.borland.jbuilder.enterprise.module				
DescriptorsManagedModelNodeReference	—	—	—	☑
ModuleDirectoryNode	—	—	☑	☑
ModuleNode	—	—	☑	☑
ModulePropertyGroup	—	—	☑	☑
ModuleType	—	—	☑	☑
ModuleWizardContext	—	—	☑	☑
com.borland.jbuilder.enterprise.module. application				
ApplicationModuleNode	—	—	☑	☑
ApplicationModuleType	—	—	☑	☑

Class	JB 7	JB 8	JB 9	JB 10
ApplicationModuleType.Feature	—	—	—	 
ApplicationModuleType.SpecFeature	—	—	—	 
ApplicationModuleUtils	—	—	—	 
com.borland.jbuilder.enterprise.module.application.legacy				
LegacyEarDeploymentDescriptor	—	—	—	 
com.borland.jbuilder.enterprise.module.client				
ApplicationClientModuleNode	—	—	—	 
ApplicationClientModuleType	—	—	—	 
ApplicationClientModuleType.Feature	—	—	—	 
ApplicationClientModuleType.SpecFeature	—	—	—	 
ApplicationClientModuleUtils	—	—	—	 
com.borland.jbuilder.enterprise.module.connector				
ConnectorModuleNode	—	—		 
ConnectorModuleType	—	—		 
ConnectorModuleUtils	—	—	—	 
com.borland.jbuilder.enterprise.module.ejb				
DataSourceWrapper	—	—	—	 
EjbModuleNode	—	—		 
EjbModuleType	—	—		 
EjbModuleUtils	—	—	—	 
EjbReference	—	—	—	 
com.borland.jbuilder.enterprise.module.ejb.legacy				
LegacyEjbDeploymentDescriptor	—	—	—	 
com.borland.jbuilder.enterprise.module.legacy				
LegacyAbstractDeploymentDescriptor	—	—	—	 
com.borland.jbuilder.enterprise.module.web				
WebModuleNode	—	—		 
WebModuleType	—	—		 
WebModuleUtils	—	—	—	 
com.borland.jbuilder.node				
BinaryResourceFileNode	 	 	—	—
CPPFileNode	 	 		
JavaScriptFileNode	 	 	 	—
SoundFileNode	 	 	—	—
com.borland.jbuilder.paths				
ProjectPathSet			 	 

Class	JB 7	JB 8	JB 9	JB 10
com.borland.jbuilder.repository				
ClassEntry	☑	☑	☑	☑☑
PackageEntry	☑	☑	☑	☑☑
SourceEntry	☑	☑	☑	☑☑
com.borland.jbuilder.runtime.servlet				
ContextDescriptor	☑☑	☑☑	☑☑	☑
ContextDescriptor.DDEXception	☑☑	☑☑	☑☑	☑
ServletDescriptor	☑☑	☑☑	☑☑	☑
WebXmlHelper	☑☑	—	—	—
com.borland.jbuilder.server				
ServerCommandLineTool	—	☑☑	☑☑	☑☑
Service.Dependency	☑☑	—	—	—
<i>ServiceDependency</i>	—	☑☑	☑☑	☑☑
com.borland.jbuilder.web				
<i>DeploymentDescriptorSupport</i>	☑☑	☑☑	☑☑	☑
<i>DescriptorReader</i>	☑☑	☑☑	☑☑	☑
<i>DescriptorWriter</i>	☑☑	☑☑	☑☑	☑
com.borland.jbuilder.web.xml				
WebXmlSupport	☑☑	☑☑	☑☑	☑
com.borland.primetime.build				
BuildInformation	—	—	☑☑	☑☑
BuildOutputEvent	—	☑☑	☑☑	☑☑
BuildProcess.Target	—	—	☑☑	☑☑
ClassOutputEvent	—	☑☑	☑☑	☑☑
com.borland.primetime.editor				
BookmarkManager	☑☑	☑☑	☑☑	—
ColorSet	—	—	☑☑	☑☑
EditorActions.AddBookmarkAction	—	—	—	☑☑
EditorActions.ClearBookmarksAction	☑☑	☑☑	☑☑	—
EditorActions.FormatFileAction	—	☑☑	☑☑	☑☑
EditorActions.ReturnKeyAction	☑☑	☑☑	—	—
EditorActions.ToggleBookmarkAction	☑☑	☑☑	☑☑	—
com.borland.primetime.ide				
<i>ProjectGroupBrowserListener</i>	—	☑☑	☑☑	☑☑
com.borland.primetime.insight.template				
TemplateActions.CreateTemplateAction	—	—	—	☑☑
TemplateManager	☑☑	☑☑	☑	☑
com.borland.primetime.node				
BinaryResourceFileNode	—	—	☑☑	☑☑
GenericResourceFileNode	—	☑☑	☑☑	☑☑

Class	JB 7	JB 8	JB 9	JB 10
ProjectGroup	—	☑ 	☑ 	☑ 
<i>ProjectGroupListener</i>	—	☑	☑ 	☑ 
SoundFileNode	—	—	☑ 	☑ 
com.borland.primetime.properties				
GlobalColorProperty	☑	☑	☑ 	☑ 
NodeArrayPropertyDefault	—	—	☑ 	☑ 
com.borland.primetime.teamdev.vcs				
VCSEventNotifier	—	—	—	☑ 
VCSRenameNotifier	—	—	☑ 	☑ 
com.borland.primetime.vfs				
CaseChangeException	—	—	☑ 	☑ 
com.borland.primetime.wizard				
<i>SummaryPage</i>	—	—	☑ 	☑ 
com.borland.primetime.xmt...				
*	—	—	☑	☑ 



Appendix C

Help Topics

The `PrimetimeHelp` class covered in Chapter 8 contains a large number of `ZipHelpTopic` fields as shown in Table C–1. They bring up topics that apply to the underlying framework, and so would be applicable to any IDE developed on top of it. Also indicated in the table is the version of JBuilder in which these fields first became available. To display any one of these you can use code like the following:

```
HelpManager.showHelp(PrimetimeHelp.TOPIC_DPrint);
```

The field names all start with “`TOPIC_`” followed by either a “`D`” or “`O`”. It appears as though the latter identify the topics as applying to a complete *dialog*, or to an embedded option page. Topics for which no page display resulted are marked as “Cannot be found”.

All the topics reside in the User Guide documentation, for which the following field from the same class provides a reference:

```
public static final ZipHelpBook BOOK_UserGuide;
```

Table C–1. Topic fields in *PrimetimeHelp*

Field	For Help On	Vers
<code>TOPIC_Build</code>	Build page (Preferences dialog)	10
<code>TOPIC_CodeFormattingBasic</code>	Formatting page (Project Projects dialog)	8
<code>TOPIC_DAddCustomExtension</code>	Add Custom Extension dialog	10
<code>TOPIC_DAddEditBookmark</code>	Add/Edit Bookmark dialog	10
<code>TOPIC_DAddToFavorites</code>	Add to Favorites dialog	10
<code>TOPIC_DBrowseProjectChanges</code>	Browsing project changes for version control	5
<code>TOPIC_DCC_AddFiles</code>	Add files under ClearCase	8
<code>TOPIC_DCC_ApplyVersionLabel</code>	Applying a version label under ClearCase	5
<code>TOPIC_DCC_CheckinFiles</code>	Checking files into a repository under ClearCase	5
<code>TOPIC_DCC_CheckoutFiles</code>	Checking files out of a repository under ClearCase	5

Field	For Help On	Vers
TOPIC_DCC_HijackFiles	Hijacking files under ClearCase	8
TOPIC_DCC_MergeFiles	Merging file versions under ClearCase	5
TOPIC_DCCAddEditViews	Adding and editing views under ClearCase	5
TOPIC_DCCConfigure	Viewing the ClearCase version control configuration	5
TOPIC_DCCPlaceProject	Placing a project into a repository under ClearCase	5
TOPIC_DCCProjectForClearCase	Project for ClearCase wizard	8
TOPIC_DCCPullPostJPX	Pulling and posting a project file	8
TOPIC_DCCPullProject	Pulling a project from version control under ClearCase	5-7
TOPIC_DCCTeamMenu	The ClearCase Team menu	5
TOPIC_DCommitProject	Committing a project under version control	5
TOPIC_DCompareFiles	Comparing any two files	5
TOPIC_DCreateLocalLabel	Create New Label dialog	10
TOPIC_DCVS_AddFiles	Adding files to version control under CVS	5
TOPIC_DCVS_ApplyVersionLabel	Applying a version label under CVS	5
TOPIC_DCVS_CheckinFiles	Checking files into a repository under CVS	5
TOPIC_DCVS_CheckoutFiles	Checking files out of a repository under CVS	5
TOPIC_DCVS_CreateBranch	CVS Create Branch dialog	9
TOPIC_DCVS_CVSNotFound	CVS Not Found dialog	8
TOPIC_DCVS_DeleteVersionLabel	Delete Version Label dialog	9
TOPIC_DCVS_InstallInstructions	Installing and configuring CVS	4
TOPIC_DCVS_MergeFiles	Merging file versions under CVS	5
TOPIC_DCVS_MergeWorkspace	Merge dialog	9
TOPIC_DCVS_MoveVersionLabel	Move Version Label dialog	9
TOPIC_DCVS_RemoveFiles	Removing files from version control under CVS	5
TOPIC_DCVS_UpdateSpecial	Update Special dialog	9
TOPIC_DCVSConfigure	Viewing the CVS version control configuration	5
TOPIC_DCVSCreateRepos	Creating a local repository for CVS	5
TOPIC_DCVSFileStatus	Checking a file's CVS status	5
TOPIC_DCVSPlaceProject	Placing a project into a repository under CVS	5
TOPIC_DCVSPlaceProjectStep1	Placing a project into a repository under CVS – first step	5

Field	For Help On	Vers
TOPIC_DCVSPlaceProjectStep2	Placing a project into a repository under CVS – second step	5
TOPIC_DCVSPlaceProjectStep3	Placing a project into a repository under CVS – third step	5
TOPIC_DCVSPullProject	Pulling a project from version control under CVS	5
TOPIC_DCVSPullProjectStep1	Pulling a project from version control under CVS – first step	5
TOPIC_DCVSPullProjectStep2	Pulling a project from version control under CVS – second step	5
TOPIC_DCVSPullProjectStep3	Pulling a project from version control under CVS – third step	5
TOPIC_DCVSPullProjectStep4	Pulling a project from version control under CVS – fourth step	9
TOPIC_DCVSTeamMenu	The CVS Team menu	5
TOPIC_DDirectoryView	Directory View page (Project Properties dialog)	8
TOPIC_DFileSaveAs	Save Copy As dialog	8
TOPIC_DFindText	Find Text dialog	7
TOPIC_DGotoBookmark	Bookmarks dialog	10
TOPIC_DLocalLabels	Manage local Labels dialog	10
TOPIC_DNewFile	Create New File dialog	8
		only
TOPIC_DNewFolder	Create New Folder dialog	10
TOPIC_DOrganizeFavorites	Organize Favorites dialog	10
TOPIC_DPageLayout	Page Layout page (Page Layout dialog)	7
TOPIC_DPageLayoutAdvanced	Advanced page (Page Layout dialog)	4
TOPIC_DPrint	Print dialog	4
TOPIC_DRenameFavorites	Rename Favorite dialog	10
TOPIC_DReplaceText	Find/Replace Text dialog	7
TOPIC_DRevert	Revert Confirmation dialog	7
TOPIC_DRevertToTip	Revert Confirmation dialog	9
TOPIC_DRuntimeConfigs	Run page (Project Properties dialog)	7
TOPIC_DSelectConfig	Choose Runtime Configuration dialog	7
TOPIC_DSelectDir	Select Directory dialog	7
TOPIC_DSelectFile	File Selection dialog	7
TOPIC_DSelectNode	Add Files or Packages to Project dialog	7
TOPIC_DSelectProjectVCS	Selecting a version control system for a project	5
TOPIC_DSelectReopen	Select File dialog	10
TOPIC_DSrchPath	Find In Path dialog	4
TOPIC_DSwitchBuffer	Switching buffers	7
TOPIC_DTeamMenu	<i>Cannot be found</i>	7

Field	For Help On	Vers
TOPIC_DUrlChooser	<i>Cannot be found</i>	7
TOPIC_DVersionControl	Team page (Project Properties dialog)	4
TOPIC_DVSS_AddFiles	Adding files to version control under VSS	5
TOPIC_DVSS_ApplyVersionLabel	Applying a version label under VSS	5
TOPIC_DVSS_CheckinFiles	Checking files into a repository under VSS	5
TOPIC_DVSS_CheckoutFiles	Checking files out of a repository under VSS	5
TOPIC_DVSS_RemoveFiles	Removing files from version control under VSS	5
TOPIC_DVSSConfigure	Viewing the VSS version control configuration	5
TOPIC_DVSSPlaceProject	Placing a project into a repository under VSS	5
TOPIC_DVSSPlaceProjectCheckout Files	Selecting files to keep checked out	9
TOPIC_DVSSPlaceProjectInclude	Selecting directories and files to include	9
TOPIC_DVSSPlaceProjectLocation	Select location in VSS database	9
TOPIC_DVSSPlaceProjectRunPath	Setting the path to the runtime directory	9
TOPIC_DVSSPlaceProjectSelectDir	Select Visual SourceSafe database directory	9
TOPIC_DVSSPlaceProjectUsername	Enter username and password	9
TOPIC_DVSSPullPostJPX	Pulling and posting the project file under VSS	5
TOPIC_DVSSPullProject	Pulling a project from version control under VSS	5
TOPIC_DVSSPullProjectDatabaseDir	Select Visual SourceSafe database directory	9
TOPIC_DVSSPullProjectRunPath	Setting the path to the runtime directory	9
TOPIC_DVSSPullProjectTargetDir	Select an empty target directory	9
TOPIC_DVSSPullProjectUsername	Enter username and password	9
TOPIC_DVSSPullProjectVSSProj	Select Visual SourceSafe project	9
TOPIC_DVSSTeamMenu	The VSS Team menu	5
TOPIC_DVSSUndoCheckout	Undoing a checkout under VSS	5
TOPIC_FileAssociation	Configure File Associations dialog	10
TOPIC_GObjectGallery	Object Gallery dialog	8
TOPIC_HistoryPane	History tab in the Content Pane	4
TOPIC_KeymapEditorKeyStroke	<i>Cannot be found</i>	9
TOPIC_KeymapEditorMain	Keymap Editor dialog	9
TOPIC_LookAndFeel	Look & Feel page (Preferences dialog)	10
TOPIC_OAudioFeedback	Audio page (IDE Options dialog)	5

Field	For Help On	Vers
TOPIC_OBrowser	Browser page (IDE Options dialog)	4
TOPIC_OCatalogFiles	XML: Oasis Catalog Files page (Preferences dialog)	10
TOPIC_OCodeInsightKeysSelect Keystroke	<i>Cannot be found</i>	8
TOPIC_OCodeTemplate	Templates page (Editor Options dialog)	4
TOPIC_OCodeTemplateAdd	Add Code Template dialog	4
TOPIC_OCodeTemplateCommon	Templates: Common page (Preferences dialog)	10
TOPIC_OCodeTemplateEdit	Edit Code Template dialog	4
TOPIC_OCodeTemplateHtml	Templates: HTML page (Preferences dialog)	10
TOPIC_OCodeTemplateInsertMacro	Insert Macro dialog	9
TOPIC_OColor	Color page (Editor Options dialog)	4
TOPIC_ODisplay	Display page (Editor Options dialog)	4
TOPIC_OEditor	Editor page (Editor Options dialog)	4
TOPIC_OEjbModeler	EJB Designer page (Preferences dialog)	6
TOPIC_OFileType	File Types page (IDE Options dialog)	4
TOPIC_OKeyMapping	Keymapping page (Preferences dialog)	10
TOPIC_OProjectEditor	Editor page (Project Properties dialog)	6
TOPIC_OPublicIdCatalog	XML: Public ID Catalog page (Preferences dialog)	10
TOPIC_OSchemaCatalog	XML: Schema Catalog page (Preferences dialog)	10
TOPIC_OSelectFileTypes	Select File Types dialog	10
TOPIC_OSystemIdCatalog	XML: System ID Catalog page (Preferences dialog)	10
TOPIC_OTagInsight	TagInsight page (Preferences dialog)	10
TOPIC_OUml	UML page (IDE Options dialog)	6
TOPIC_Personality	Personality page (Preferences dialog)	10
TOPIC_PersonalityAdvanced	Personality Configurations dialog	10
TOPIC_SaveWorkspaceAs	Save Workspace As dialog	10
TOPIC_ToolOptions	Configure Tools dialog	10
TOPIC_ToolProperties	Add/Edit Tool dialog	10
TOPIC_WExportToAnt	Export to Ant wizard	10
TOPIC_WExportToAnt2	Specify Generation Actions (Export to Ant wizard)	10
TOPIC_XMT_NODE_TYPE_LIST_FORM	List of Elements page	10



Appendix D

XML Tools OpenTool

The XMLTools OpenTool provides additional support for XML files within JBuilder. It adds menu items to the popup menus in the Project Pane and in the Content Pane, allowing you to validate the current XML node, or to open its DTD or stylesheet. It also adds an XSLT viewer tab for XML nodes, letting you specify an XSL transformation to apply to your XML, and view the results in both source and rendered form. This appendix describes the workings of the tool as a whole.

NOTE

Some of these features are now available in JBuilder as standard.

In Chapter 7 the properties of the XMLTools utility were discussed. There are four global properties defined: one for the list of available XML validators, one for the list of XSL transformation engines, and two others being the indexes into these lists for the items currently selected. They are grouped together and declared in the `XMLToolsPropertyGroup` class. The corresponding `XMLToolsPropertyPage` class supplies a UI wherein the user can easily alter these settings.

In Chapter 14 its additions to the Project Menu are examined. The tool adds three items to the Project Pane popup menu when activated with a single XML node selected: `Validate XML`, `Open DTD/Schema`, and `Open XSL Stylesheet`. These same items appear in the popup menu within the editor in the Content Pane as well (when it contains XML). The first item calls on the current XML validator to check that the node contains valid XML. The other two open up secondary documents referred to by the main XML: either the DTD or XML Schema defined for the document, or the XSL Transformation that it prefers to use. Their menu items are disabled if no such references can be found in the current document.

XMLValidator Interface

Since there are a number of XML parsers and validators available, the tool is designed to allow any of these to be plugged into it. It accomplishes this aim by defining the `XMLValidator` interface as shown in Listing D-1. The only requirements are a name for display purposes and a method to actually perform

the validation. The document to work with arrives as a SAX input source and an exception is thrown (preferably a `SAXParseException`) if a problem is detected.

Listing D-1. The XMLValidator interface.

```
package wood.keith.opentools.xmltools;

import org.xml.sax.InputSource;

/**
 * Interface for validating an XML document.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 16 October 2000
 * @version 2.0 15 February 2002
 */
public interface XMLValidator {

    /**
     * Validate an XML document.
     *
     * @param xmlDocument a source representing the XML document
     *                    to be validated
     * @throws Exception when an error occurs, may be a
     *                   SAXParseException showing line and column
     *                   where the error occurred
     */
    public void validate(InputSource xmlDocument) throws Exception;

    /**
     * Return a description for this validation implementation.
     *
     * @return the implementation's description
     */
    public String getValidatorName();
}
```

To add an actual validator to the XMLTools suite, all you do is implement this interface around your favorite parser and ensure that the new class appears on JBuilder's classpath. Then in the tool's property page you specify the full name of the new class and select it as the one to use for subsequent validations. Listing D-2 shows how you might wrap the Xerces (<http://xml.apache.org/xerces-j/index.html>) parser to have it perform the validation for you.

Listing D-2. Validation using Xerces 2.

```
package wood.keith.opentools.xmltools;

import org.apache.xerces.parsers.SAXParser;

import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

/**
 * Xerces 2 implementation for validating an XML document.
 * For more details see http://xml.apache.org/xerces-j/index.html.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 16 October 2000
 * @version 2.0 15 February 2002
 */
public class XMLXerces2Validator implements XMLValidator, ErrorHandler {

    // Singleton parser
    private static SAXParser _parser = null;
```

```

/**
 * Validate an XML document.
 *
 * @param xmlDocument a source representing the XML document
 *                    to be validated
 * @throws Exception when an error occurs, may be a SAXParseException
 *                    showing line and column where the error occurred
 */
public void validate(InputSource xmlDocument) throws Exception {
    // And parse it
    getParser().parse(xmlDocument);
}

/**
 * Return the singleton parser.
 *
 * @return the validating parser object
 */
private SAXParser getParser() throws SAXException {
    if (_parser == null) {
        // Create a new parser
        _parser = new SAXParser();
        // Turn on validation
        _parser.setFeature("http://xml.org/sax/features/validation", true);
        // Set error handler
        _parser.setErrorHandler(this);
    }
    return _parser;
}

/**
 * Return a description for this validation implementation.
 *
 * @return the implementation's description
 */
public String getValidatorName() {
    return "Xerces 2 (SAX) from Apache";
}

/**
 * Implement ErrorHandler warning notification.
 *
 * @param exception the generated warning
 */
public void warning(SAXParseException exception) throws SAXException {
    throw exception;
}

/**
 * Implement ErrorHandler error notification.
 *
 * @param exception the generated error
 */
public void error(SAXParseException exception) throws SAXException {
    throw exception;
}

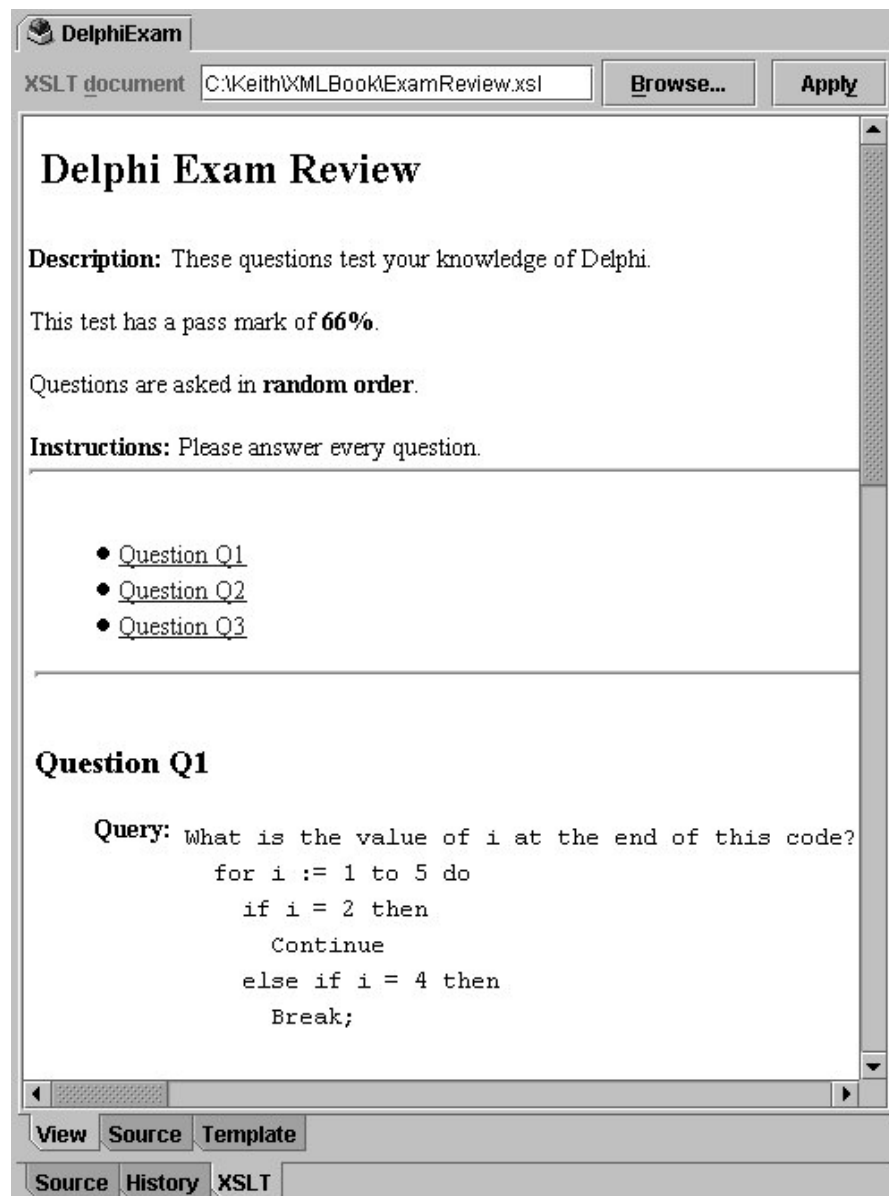
/**
 * Implement ErrorHandler fatal error notification.
 *
 * @param exception the generated fatal error
 */
public void fatalError(SAXParseException exception) throws SAXException
{
    throw exception;
}
}

```

XSLTViewerFactory Class

The other main feature of the XMLTools package is a custom node viewer that supports XSL Transformations. It appears as a viewer tab named XSLT when an XML document is opened (see Figure D-1). Within it are three sub-tabs: one to show the results of the transformation as HTML or plain text, one to display the source behind that rendered view, and the last to hold the text of the XSLT document. You may enter the name of the XSLT file to use (it defaults to the one specified in the original XML document), or browse for it through the controls along the top of the form. When you are ready you press the Apply button to invoke the currently selected transformation engine (as set through the property page).

Figure D-1. The XSLT node viewer.



As with the XML validation, there are several transformation engines available. So another interface defines the required abilities, leaving an adapter class to cater for each individual implementation. The `XSLTransformer` interface (see Listing D-3) is just like the validation one, requiring a name for display purposes and a method to invoke the transformation. The results of the processing are returned as a string value.

Listing D-3. The XSLTransformer interface.

```
package wood.keith.opentools.xmltools;

import org.xml.sax.InputSource;

/**
 * Interface for applying an XSL Transformation to a document.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 16 October 2000
 * @version 2.0 15 February 2002
 */
public interface XSLTransformer {
    /**
     * Apply an XSL transformation to a document.
     *
     * @param xmlDocument a source representing the XML document
     *                    to be transformed
     * @param xslDocument a source representing the XSL transformation
     * @return the text resulting from the transformation process
     * @throws Exception when an error occurs, may be a
     *                   SAXParseException showing line and column
     *                   where the error occurred
     */
    public String transform(InputSource xmlDocument,
                           InputSource xslDocument) throws Exception;

    /**
     * Return a description for this transformation implementation.
     *
     * @return the implementation's description
     */
    public String getTransformerName();
}
```

To integrate the node viewer into JBuilder, you start with a node viewer factory – `XSLTVIEWerFactory` in this case (see Listing D-4). Its OpenTools initialization routine registers an instance of itself with the `Browser` class. When a new node is opened the factory's `canDisplayNode` method is called and returns true if that node is XML-based. Once accepted, the `createNodeViewer` method returns a new instance of the actual node viewer.

Listing D-4. A factory for the node viewer.

```
package wood.keith.opentools.xmltools;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.text.MessageFormat;
import java.util.EventListener;
import javax.swing.JOptionPane;
import javax.swing.event.EventListenerList;

import com.borland.primetime.PrimeTime;
import com.borland.primetime.ide.Browser;
import com.borland.primetime.ide.Context;
import com.borland.primetime.ide.NodeViewer;
import com.borland.primetime.ide.NodeViewerFactory;
```

```

import com.borland.primetime.node.Node;
import com.borland.primetime.node.TextFileNode;

/**
 * Create a page for applying XSLT transformations to XML documents.
 *
 * @author Keith Wood (kbwood@iprimus.com.au)
 * @version 1.0 16 October 2000
 * @version 2.0 15 February 2002
 */
public class XSLTViewerFactory implements NodeViewerFactory {

    /** The name of the transformer property. */
    public static final String XSL_TRANSFORMER_PROP = "xslTransformer";

    /** Messages. */
    private static final String CAPTION = "XML Tools";
    private static final String TRANSFORMER_MISSING =
        "XSLTransformer class not found\n{0}";
    private static final String TRANSFORMER_BUILD =
        "XSLTransformer class not created\n{0}";

    /** Listeners for transformer changes. */
    private static EventListenerList _listenerList =
        new EventListenerList();

    /** The XSL transformation object. */
    private static XSLTransformer _xslTransformer = null;

    /** The viewer factory object. */
    private static XSLTViewerFactory _xsltViewerFactory =
        new XSLTViewerFactory();

    /** Static initialisation. */
    static {
        // Find the name of the XMLValidator class
        String[] transformerList =
            XMLToolsPropertyGroup.TRANSFORMER_LIST.getValues();
        int index = XMLToolsPropertyGroup.TRANSFORMER.getInteger();
        String transformerClass = (index < transformerList.length ?
            transformerList[index] : XMLToolsPropertyGroup.DEFAULT_TRANSFORMER);
        // And create an instance of it
        setTransformer(transformerClass);
    }

    /**
     * Register this factory as a NodeViewer factory.
     *
     * @param majorVersion the major version of the current OpenTools API
     * @param minorVersion the minor version of the current OpenTools API
     */
    public static void initOpenTool(byte majorVersion, byte minorVersion) {
        if (majorVersion != PrimeTime.CURRENT_MAJOR_VERSION) {
            return;
        }
        Browser.registerNodeViewerFactory(_xsltViewerFactory);
        if (PrimeTime.isVerbose()) {
            System.out.println("Loaded XSLT Viewer v" + XMLTools.VERSION);
            System.out.println("Written by Keith Wood (kbwood@iprimus.com.au)");
        }
    }

    /**
     * One of the functions needed to implement NodeViewerFactory.
     * Use this function to tell the Browser if this factory can
     * view a certain node.
     *
     * @param node the node the Browser will open
     * @return true if this factory can create a viewer for this node,
     *         false otherwise.
     */

```

```

*/
public boolean canDisplayNode(Node node) {
    return XMLTools.isXmlNode(node);
}

/**
 * One of the functions needed to implement NodeViewerFactory.
 * Used to create the viewer for the node that is described
 * by a context. The viewer created is of type XSLTNodeViewer.
 *
 * @param context the context that describes the node.
 * @return a NodeViewer capable of viewing this node, or
 *         null if the viewer creation fails.
 */
public NodeViewer createNodeViewer(Context context) {
    Node node = context.getNode();
    if (canDisplayNode(node)) {
        return new XSLTNodeViewer(context, (TextFileNode)node);
    }
    return null;
}

/**
 * Instantiate the transformer class.
 *
 * @param transformerClass the name of the XSLTransformer class
 */
public static void setTransformer(String transformerClass) {
    try {
        XSLTransformer oldXslt = _xsltTransformer;
        _xsltTransformer = null;
        _xsltTransformer =
            (XSLTransformer)Class.forName(transformerClass).newInstance();
        firePropertyChange(XSL_TRANSFORMER_PROP, oldXslt, _xsltTransformer);
        oldXslt = null;
    }
    catch (ClassNotFoundException cnfe) {
        JOptionPane.showMessageDialog(Browser.getActiveBrowser(),
            MessageFormat.format(TRANSFORMER_MISSING, new Object[]
                {cnfe.getMessage()}), CAPTION, JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(Browser.getActiveBrowser(),
            MessageFormat.format(TRANSFORMER_BUILD, new Object[]
                {ex.getMessage()}), CAPTION, JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Retrieve the XSL Transformation object.
 *
 * @return the current XSLTransformer object
 */
public static XSLTransformer getTransformer() {
    return _xsltTransformer;
}

/**
 * Add a PropertyChangeListener to the listener list.
 * The listener is registered for all properties.
 *
 * @param listener the PropertyChangeListener to be added
 */
public static void addPropertyChangeListener(
    PropertyChangeListener listener) {
    _listenerList.add(PropertyChangeListener.class, listener);
}

/**

```

```

    * Remove a PropertyChangeListener from the listener list.
    * This removes a PropertyChangeListener that was registered
    * for all properties.
    *
    * @param listener the PropertyChangeListener to be removed
    */
    public static void removePropertyChangeListener(
        PropertyChangeListener listener) {
        _listenerList.remove(PropertyChangeListener.class, listener);
    }

    /**
    * Report a bound property update to any registered listeners.
    * No event is fired if old and new are equal and non-null.
    *
    * @param propertyName the programmatic name of the property
    *                    that was changed
    * @param oldValue the old value of the property
    * @param newValue the new value of the property
    */
    public static void firePropertyChange(String propertyName,
        Object oldValue, Object newValue) {
        if (oldValue != null && newValue != null &&
            oldValue.equals(newValue)) {
            return;
        }

        EventListener[] listeners =
            _listenerList.getListeners(PropertyChangeListener.class);
        PropertyChangeEvent event = null;
        for (int index = 0; index < listeners.length; index++) {
            if (event == null) {
                event = new PropertyChangeEvent(
                    _xsltViewerFactory, propertyName, oldValue, newValue);
            }
            ((PropertyChangeListener) listeners[index]).propertyChange(event);
        }
    }
}

```

The remaining methods keep track of the currently selected transformation engine, and inform registered listeners when that value changes. Transformation implementations are identified by class name, so problems may arise when attempting to convert this into an actual class instance. Interested parties can retrieve the current setting through the `getTransformer` method.

XSLTNodeViewer Class

The factory returns an instance of the `XSLTNodeViewer` class when appropriate. It extends `AbstractNodeViewer` and supplies the name and tooltip for the new viewer tab, along with a component to place in the Content Pane, and a `null` for the Structure Pane component to indicate that nothing appears there.

Listing D-5. The XSLT node viewer.

```

package wood.keith.opentools.xmltools;

import javax.swing.JComponent;
import javax.swing.JOptionPane;

import com.borland.primetime.ide.Browser;
import com.borland.primetime.ide.Context;
import com.borland.primetime.node.TextFileNode;
import com.borland.primetime.viewer.AbstractNodeViewer;

/**

```

```

* XSLTNodeViewer is a wrapper around XSLTViewer, which is
* the class that implements the low level details of managing
* a viewer, in this case a tabbedPane and several editorpanes.
*
* @author Keith Wood (kbwood@iprimus.com.au)
* @version 1.0 16 October 2000
* @version 2.0 15 February 2002
*/
public class XSLTNodeViewer extends AbstractNodeViewer {

    /**
     * Title and longer description for the viewer tab.
     */
    private static final String TAB_TITLE = "XSLT";
    private static final String TAB_DESC = "XSLT processing";

    /**
     * Messages.
     */
    private static final String TRANSFORMER_MISSING =
        "XSLTransformer class not found";

    /** The implementation of the lower level viewer. */
    private XSLTViewer xsltViewer = null;

    /** The node which holds the XML document to be transformed. */
    private TextFileNode _editorNode;

    /**
     * Create a XSLTNodeViewer object, based on a Context and a Node object,
     * both of which we will get from the XSLTNodeViewerFactory.
     *
     * @param context the context of the editor buffer
     * @param editorNode the node which holds the XML document
     */
    public XSLTNodeViewer(Context context, TextFileNode editorNode) {
        super(context);
        editorNode = editorNode;
    }

    /**
     * This title will show on the tab of the viewer.
     */
    public String getViewerTitle() { return TAB_TITLE; }

    /**
     * This is a longer description of the viewer.
     */
    public String getViewerDescription() { return TAB_DESC; }

    /**
     * Creates the UI component for the XSLT pane.
     *
     * @return the XSLTViewer to display on the XSLT pane
     */
    public JComponent createViewerComponent() {
        if (XSLTViewerFactory.getTransformer() == null) {
            JOptionPane.showMessageDialog(Browser.getActiveBrowser(),
                TRANSFORMER_MISSING, TAB_DESC, JOptionPane.ERROR_MESSAGE);
            return null;
        }
        if (xsltViewer == null) {
            xsltViewer = new XSLTViewer(_editorNode);
        }
        return xsltViewer;
    }

    /**
     * Creates the UI component to display in the Structure Pane.
     * Not yet implemented.
     */
}

```

```
*  
* @return null  
*/  
public JComponent createStructureComponent() { return null; }  
  
/**  
* Forces the viewer to a certain line number position.  
*  
* @param lineNo the line number the viewer should go to.  
*/  
public void setPosition(int lineNo) {  
    xsltViewer.setPosition(lineNo);  
}  
}
```

Index

•

.pme, 79

A

AbstractDeploymentDescriptor
class, 259

getBytes, 259

getDefaultAccessor, 259

getEncoding, 259

getExtraLocation, 259

getFileAccessor, 260

getName, 260

getTimestamp, 260

setBytes, 260

setExtraLocation, 260

setFileAccessor, 260

setName, 260

setTimestamp, 260

toString, 260

AbstractDescriptorConversion
class, 261

addErrorMessage, 262

addWarningMessage, 262

convert, 262

createInterfaceFrom-

Descriptors, 262

findDescriptor, 262

generate, 262

getBeanDataSource, 263

getBeanDataSourceName,
263

getDescriptorDocument, 263

getEjbNode, 263

getGenericDDName, 263

getJotMethods, 263

getProject, 263

getPropertyNamesToSurface,
263

getServer, 263

isEjbJarDirty, 264

isModuleDirty, 264

needToUpdate, 264

patchForAppServer, 264

setEjbJarDirty, 264

setModuleDirty, 264

updateEjbModule, 264

verifyDeployment-
Descriptors, 264

AbstractNodeViewer class

example, 295

AbstractRevisionNumber class,
203

compareTo, 203

doComparison, 203

getPrecedence, 203

getRevisionNumberInstance,
203

getRevisionString, 203

ACTION_Customizer field

BasicLayoutAssistant, 142

ACTION_MoveToFirst field

BasicLayoutAssistant, 142

ACTION_MoveToLast field

BasicLayoutAssistant, 142

ACTION_Serialize field

BasicLayoutAssistant, 142

ActionGroup class

example, 216

actions

for VCS, 200, 201, 211

actionVerify method

JBuilderInfo, 26

activate method

Designer, 111

example, 116

add method

CmtModel, 96

example, 120

addActionListener method

ColorPanel, 14

addAddition method

Diff, 21

addAssignment method

example, 187

JotCodeBlock, 171

addBlankLine method

example, 187

JotCommentable, 174

addBufferListener method

example, 125

addChange method

Diff, 21

addChangeListener method

ListPanel, 35

addClass method

example, 186

JotSourceFile, 160

addComment method

example, 187

JotCommentable, 175

addComponentSourceListener
method

CmtComponentSource, 85

addConstructor method

JotClassSource, 167

addDeletion method

Diff, 22

addDesignerListener method

DesignerManager, 107

addDesignerReleaseListener
method

DesignerManager, 107

addDoStatement method

JotCodeBlock, 171

addEntries method

PathSet, 44

addErrorMessage method

AbstractDescriptor-

Conversion, 262

addField method

example, 187

JotClassSource, 167

addForStatement method

JotCodeBlock, 171

addIfStatement method

example, 188

JotCodeBlock, 172

addImport method

example, 146, 185

JotSourceFile, 160

addInitBlock method

JotClassSource, 167

addInnerClass method

JotClassSource, 167

addInnerClass method

JotCodeBlock, 172

addInterface method

example, 186

JotClassSource, 168

addJotFileListener method

JotSourceFile, 160

addListElement method

ListPanel, 35

addMessage method

example, 259

addMethod method

CmtComponentSource, 85

example, 187

JotClassSource, 168

addMethodCall method

JotCodeBlock, 172

addMethodDeclaration method

JotClassSource, 168

addModel method

CmtComponentSource, 85

- addMouseListenerToHeaderIn-
Table method
TableSorter, 63
- addParameter method
example, 187
JotMethodSource, 170
- addPersonalIgnoreFiles method
VCSUtils, 205
- addProjectLibrary method
ProjectPathSet, 50
- addProperty method
CmtComponentSource, 86
- addPropertyChangeListener
method
CmtSubcomponent, 88, 91
example, 122
- addPropertyState method
CmtSubcomponent, 89
- addReturnStatement method
example, 190
JotCodeBlock, 172
- addSourceBridge method
ServerLauncher, 248
- addStatement method
example, 188
JotCodeBlock, 172
- addSubcomponent method
CmtComponentSource, 86
example, 120
- addTeamIgnoreFiles method
VCSUtils, 206
- addThrowSpecifier method
example, 188
JotMethodSource, 170
- addToIgnoreList method
VCS, 199
- addTraceCategory method
Debug, 16
- addTryStatement method
example, 189
JotCodeBlock, 172
- addUniquePath method
PathSet, 44
Server, 237
- addUniquePaths method
PathSet, 44
- addUniquePathsIfEnabled
method
PathSet, 44
- addUserData method
JotClass, 163
JotMarker, 161
- addVariableDeclaration method
example, 188
JotCodeBlock, 172
- addVCS method
example, 214
VCSFactory, 199
- addWarningMessage method
AbstractDescriptor-
Conversion, 262
- addWhileStatement method

- example, 189
JotCodeBlock, 172
- addWizardPage method
example, 181
- adjustPositionForNib method
DesignView, 136
example, 148
- annotate method
Designer, 112
example, 116
- antify method
AppServerTargeting, 255
- appendHttpPort method
ServerLauncher, 248
- application servers, 224
client libraries, 239
deployment descriptors, 255,
259, 260
features, 232
getting a server, 227
getting a service, 226
JDKs, 239
legacy server registration,
228
manager, 225
property pages, 270
server configuration
registration, 228
server JDK registration, 228
server registration, 229
server targeting registration,
229
servers, 236
service registration, 229, 245
service type icons, 231
service type registration, 229
service types, 230
services, 232
setup, 265, 267
setup registration, 266
- AppServerTargeting class, 255
antify, 255
createAntJarTask, 256
createAntTmpJarTask, 256
getEjbJarProperties, 256
getEjbProperties, 256
getJarTarget, 256
getPageName, 256
getServer, 256
hasEjbJarProperties, 257
hasEjbProperties, 257
postProcessBuild, 257
preProcessBuild, 257
updateBuildTask, 257
updateDeployment-
Descriptors, 257
updateVerifyReport, 258
verifyDeployment-
Descriptors, 258
- assureNibs method
DesignView, 136
example, 148

- attemptDefaultConfiguration
method
Server, 237
- AuralImage class, 5
createAuralImage, 6
dumpPixelMaps, 6
getAlphaThreshold, 6
getAuralImage, 6
getAuraRGB, 6
getBlendedImage, 6
getSourceImage, 6
setAlphaThreshold, 6
setAuraRGB, 6

B

- BasicLayoutAssistant class, 140
ACTION_Customizer, 142
ACTION_MoveToFirst, 142
ACTION_MoveToLast, 142
ACTION_Serialize, 142
calcBestZ, 140, 141, 142
example, 144
- BasicWizard class
example, 180
- BorderCornerLayoutAssistant
example, 142
- browseClass method
PackageBrowserTree, 41
- browsePackageOrClass method
PackageBrowserTree, 41
- Browser class
example, 292
- BrowserAction class
example, 98
- Buffer class
example, 125
- BUFFER_REVISION field
RevisionInfo, 203
- bufferChanged method
example, 126
- BufferListener interface
example, 124
- BufferUpdater interface
example, 124
- buildFeatureSet method
Service, 232
- buttons
on a strip, 6
- ButtonStrip class, 6
createButton, 7
createCancelButton, 7
createHelpButton, 7
createNoButton, 7
createOkButton, 7
createYesButton, 7
setOrientation, 7

C

- calcBestZ method

- BasicLayoutAssistant, 140, 141, 142
- canAdd method
 - ListPanel, 35
- cancelOperation method
 - CommitAction, 211
- canConvertToInteger method
 - Strings, 59
- canDisplayNode method
 - example, 293
- canEdit method
 - ListPanel, 35
- canMoveDown method
 - ListPanel, 36
- canMoveUp method
 - ListPanel, 36
- canRemove method
 - ListPanel, 36
- canStop method
 - ServerLauncher, 248
- capitalize method
 - Strings, 59
- CATEGORY field
 - Setup, 268
- centerOnScreen method
 - DefaultDialog, 19
- changeDirectoryReferencesInString method
 - Server, 237
- CHAR_ANY field
 - RegularExpression, 55
- CHAR_ESCAPE field
 - RegularExpression, 55
- CHAR_WILDCARD field
 - RegularExpression, 55
- checkChildNodes method
 - Service.Type, 231
- checkModel method
 - TableSorter, 63
- checkProjectLocal method
 - VCSUtils, 206
- checkReread method
 - CmtComponentSource, 86
 - JotPackages, 157
- checkSetup method
 - Server, 237
- checkShowRestartWarning method
 - SetupManager, 265
- CheckTree class, 7
- CheckTreeNode class, 8
 - getExpandedIcon, 9
 - getIcon, 9
 - getText, 9
 - isAffectedByParentEnabled, 9
 - isCheckedable, 9
 - isChecked, 9
 - isEnabled, 9
 - isEnabledmentAffectedByParent, 9
 - isLocked, 9
 - setAffectChildrenEnabled, 9
 - setAffectedByParent, 9
 - setChecked, 9
 - setEnabled, 9
 - setEnabledmentAffectedByParent, 10
 - setExpandedIcon, 10
 - setIcon, 10
 - setLocked, 10
 - setText, 10
 - toString, 10
- CLASS_SCOPE field
 - CmtSubcomponent, 91
- classes
 - selecting, 38, 41
- Classes class, 10
 - findPathUrl, 10
 - getRootEntryFromClasspath, 11
 - getShortName, 11
 - pathContainsClass, 11
 - toPath, 11
- classpath
 - contains, 11
 - find, 10
 - path sets, 43
- cleanupRemovedComponent method
 - LayoutAssistant, 132
- clear method
 - Server, 237
- clearExceptionQueue method
 - ServerLauncher, 249
- clearProjectSettings method
 - Server, 237
- clearSourceBridges method
 - ServerLauncher, 249
- ClientJarService class, 232
- clipCenter method
 - ClipPath, 11
- clipMenuItemPath method
 - ClipPath, 12
- ClipPath class, 11
 - clipCenter, 11
 - clipMenuItemPath, 12
- ClipPathRenderer class, 12
- close method
 - CmtModel, 96
 - Designer, 112
 - example, 117
 - ZipIndex, 68
- closeDialog method
 - PackageBrowserDialog, 39
- CMT, 79
 - component events, 85, 86, 87, 88
 - component factory
 - registration, 83
 - subcomponent events, 89, 91
- CmtComponent interface, 83
 - getContainerDelegate, 83
 - getCustomizerClass, 83
 - getDefaultEventIndex, 84
 - getDefaultPropertyIndex, 84
 - getEvent, 84
 - getEvents, 84
 - getException, 84
 - getFile, 84
 - getIcon, 84
 - getLiveClazz, 84
 - getLiveType, 84
 - getManager, 84
 - getMethod, 84
 - getMethods, 84
 - getProperties, 84
 - getProperty, 84
 - getPropertyFromSetter, 84
 - getType, 84
 - isBean, 85
 - isContainer, 85
 - isHiddenState, 85
 - isReadOnly, 85
 - release, 85
- CmtComponentListener interface, 88
 - componentChanged, 88
 - eventChanged, 88
 - methodChanged, 88
 - propertyChanged, 88
 - subcomponentChanged, 88
- CmtComponentManager class, 82
 - createComponent, 83
 - findEditor, 83
 - registerComponentFactory, 83
- CmtComponents interface, 82
 - getComponent, 82
 - getPackages, 82
 - getProject, 82
 - release, 82
 - shutdown, 82
- CmtComponentSource interface, 85
 - addComponentSourceListener, 85
 - addMethod, 85
 - addModel, 85
 - addProperty, 86
 - addSubcomponent, 86
 - checkReread, 86
 - commit, 86
 - example, 116, 120
 - fireComponentChanged, 86
 - fireSubcomponentChanged, 86
 - getInitMethod, 86
 - getLastDesignedNode, 86
 - getModel, 86
 - getModels, 86
 - getModelTree, 87
 - getName, 87
 - getSourceFile, 87
 - getSubcomponent, 87

- getSubcomponents, 87
- INIT_METHOD_NAME, 87
- INIT_METHOD_PARAMS, 87
- JBOWNER_METHOD_NAME, 87
- removeComponentSource-Listener, 87
- removeMethod, 87
- removeModel, 87
- removeProperty, 87
- removeSubcomponent, 87
- renameSubcomponent, 87
- setLastDesignedNode, 87
- VA_INIT_METHOD_NAME, 87
- CMTDump example, 97
- CmtEvent interface, 93
- CmtEventSource interface, 93
- CmtEventState interface, 95
 - example, 99
 - getDefaultHandlerText, 95
- CmtFeature interface, 92
 - getComponent, 92
 - getDisplayName, 92
 - getName, 92
 - getShortDescription, 92
 - isExpert, 92
 - isHidden, 92
- CmtModel interface, 95
 - add, 96
 - close, 96
 - example, 119
 - getChildren, 96
 - getComponent, 96
 - getGraph, 96
 - getName, 96
 - getRoot, 96
 - isMultiInstance, 96
 - isSubcomponentOwned, 96
 - move, 96
 - remove, 97
- CmtModelNode interface, 97
 - example, 116, 122
 - getModel, 97
 - getSubcomponent, 97
 - getTag, 97
 - isDesignable, 97
- CmtProperty interface, 92
 - getEditor, 92
 - getReadMethod, 92
 - getType, 93
 - getWriteMethod, 93
 - isBound, 93
 - isConstrained, 93
 - isReadable, 93
 - isWritable, 93
- CmtPropertySetting interface, 95
 - getMethodCall, 95
 - getProperty, 95
 - getSubcomponent, 95
 - getValue, 95
 - getValueSource, 95
 - setValueSource, 95
- CmtPropertySource interface, 93
 - setName, 93
 - setReadable, 93
 - setType, 93
 - setWritable, 93
- CmtPropertyState interface, 93
 - example, 99
 - getProperty, 94
 - getPropertySetting, 94
 - getSubcomponent, 94
 - getValue, 94
 - getValueSource, 94
 - getValueText, 94
 - isDefault, 94
 - isPseudoPropertyState, 94
 - isReadOnly, 94
 - reset, 94
 - setDefaultValue, 94
 - setValue, 94
 - setValueSource, 94
 - setValueText, 95
 - triggerPropertyChange, 95
- CmtSubcomponent class
 - example, 98
- CmtSubcomponent interface, 88
 - addPropertyChangeListener, 88, 91
 - addPropertyState, 89
 - CLASS_SCOPE, 91
 - copy, 89
 - example, 116, 120
 - firePropertyChange, 89
 - getAsContainer, 89
 - getAssignment, 89
 - getComponent, 89
 - getComponentType, 89
 - getCustomizerDialog, 89
 - getDeclaredClass, 89
 - getDefaultEventState, 89
 - getDefaultPropertyState, 89
 - getEventState, 89
 - getEventStates, 89
 - getInitializer, 89
 - getInitMethod, 90
 - getLiveClass, 90
 - getLiveInstance, 90
 - getMethodCalls, 90
 - getName, 90
 - getOuterComponent, 90
 - getPropertyState, 90
 - getPropertyStates, 90
 - getScope, 90
 - getSourceName, 90
 - isNeedsSerialize, 90
 - METHOD_SCOPE, 91
 - release, 90
 - releaseLiveInstance, 91
 - removePropertyChangeListener, 91
 - serialize, 91
 - setAssignment, 91
 - setCustomizerDialog, 91
 - setInitializer, 91
 - setLiveClass, 91
 - setLiveInstance, 91
 - setNeedsSerialize, 91
 - setScope, 91
- code
 - download, vi
- code templates
 - todo, 191
- ColorCombo class, 12
 - decodeColor, 13
 - decodeColors, 13
 - encodeColor, 13
 - encodeColors, 13
 - getCustomColors, 13
 - getPopupAlignment, 13
 - getPopupGridHeight, 13
 - getSelectedColor, 13
 - setCustomColors, 13
 - setPopupAlignment, 13
 - setPopupGridHeight, 13
 - setSelectedColor, 13
- ColorPanel class, 14
 - addActionListener, 14
 - color constants, 15
 - fixedColors, 15
 - getActionCommand, 14
 - getCustomColors, 14
 - getSelectedColor, 14
 - removeActionListener, 14
 - setActionCommand, 14
 - setCustomColor, 14
 - setCustomColors, 15
 - setPanelGridHeight, 15
 - setSelectedColor, 15
- colors
 - constants, 15
 - selecting, 12, 14
- commit method
 - CmtComponentSource, 86
 - example, 121, 185
 - JotPackages, 157
- CommitAction class, 211
 - cancelOperation, 211
 - getErrorMessage, 211
 - getPropertyPage, 211
 - isCancellable, 212
 - performAction, 212
 - setRunnerListener, 212
 - wasCommitSuccessful, 212
- compare method
 - TableSorter, 63
- compareRowsByColumn method
 - TableSorter, 63
- compareTo method
 - AbstractRevisionNumber, 203
- Component Modeling Tool. *See* CMT

- componentAbsLocation method
 - DesignView, 136
 - example, 147
 - componentChanged method
 - CmtComponentListener, 88
 - CompositeIcon class, 15
 - findHit, 16
 - configureLauncher method
 - ServerLauncher, 249
 - Service, 232
 - configureServices method
 - ServerLauncher, 249
 - ConnectorService class, 232
 - constraintDialog field
 - DesignView, 137
 - constraintEditorSelection-
 - Changing method
 - LayoutAssistant, 132
 - contains method
 - ZipIndex, 69
 - convert method
 - AbstractDescriptor-
 - Conversion, 262
 - convertLineEndings method
 - Strings, 59
 - convertToInteger method
 - Strings, 59
 - convertToPlatformLineEndings
 - method
 - Strings, 60
 - convertToUnixLineEndings
 - method
 - Strings, 60
 - copy method
 - CmtSubcomponent, 89
 - Streams, 58
 - createAntJarTask method
 - AppServerTargeting, 256
 - createAntTmpJarTask method
 - AppServerTargeting, 256
 - createAuralImage method
 - AuralImage, 6
 - createBackupAndOutputDirs
 - method
 - VCSUtils, 206
 - createButton method
 - ButtonStrip, 7
 - createCancelButton method
 - ButtonStrip, 7
 - createClientJar method
 - Server, 238
 - createClientLibrary method
 - Server, 238
 - createComponent method
 - CmtComponentManager, 83
 - createHelpButton method
 - ButtonStrip, 7
 - createInterfaceFromDescriptors
 - method
 - AbstractDescriptor-
 - Conversion, 262
 - createLibrariesFromSetup
 - method
 - Server, 238
 - createLibrary method
 - Server, 238
 - createNoButton method
 - ButtonStrip, 7
 - createNodeViewer method
 - example, 293
 - createOkButton method
 - ButtonStrip, 7
 - createSetupPage method
 - Setup, 267
 - createSetupPanel method
 - NestingSetupPropertyPage,
 - 271
 - SetupPage, 270
 - SetupPropertyPage, 270
 - createViewComponent method
 - example, 295
 - createWizard method
 - example, 181
 - createYesButton method
 - ButtonStrip, 7
 - customizeArguments method
 - ServerLauncher, 249
 - customizeClassPath method
 - ServerLauncher, 249
 - customizeLibraries method
 - ServerLauncher, 249
 - customizeTransportAddress
 - method
 - ServerLauncher, 249
 - customizeVmParameters method
 - ServerLauncher, 250
- ## D
- David Brouse, 212
 - bio, 212
 - Debug class, 16
 - addTraceCategory, 16
 - debugRect, 16
 - enableAssert, 16
 - enableOutput, 17
 - ensure, 17
 - flush, 17
 - print, 17
 - println, 17
 - printInc, 17
 - printProfiler, 17
 - printStackTrace, 17
 - removeTraceCategory, 17
 - setLogStream, 17
 - startProfiler, 17
 - stopProfiler, 18
 - trace, 18
 - warn, 18
 - debugging, 16
 - debugRect method
 - Debug, 16
 - decapitalize method
 - Strings, 60
 - decode method
 - Strings, 60
 - Strings.StringEncoding, 62
 - decodeArray method
 - Strings, 60
 - decodeColor method
 - ColorCombo, 13
 - decodeColors method
 - ColorCombo, 13
 - decodeKeyStroke method
 - KeyStrokeEditorPanel, 32
 - DEFAULT_WEIGHT field
 - Server, 247
 - DefaultDialog class, 19
 - centerOnScreen, 19
 - doDefaultClick, 20
 - findFrame, 20
 - getBoundsAsString, 20
 - isAutoCenter, 20
 - setAutoCenter, 20
 - setBoundsAsString, 20
 - setCancelButton, 20
 - setDefaultButton, 20
 - setHelpButton, 20
 - show, 20
 - showModalDialog, 20
 - showSimpleModalDialog, 20
 - showSimpleNonModal-
 - Dialog, 20
 - delete method
 - PathSet, 44
 - deployLibraries method
 - ServerLauncher, 250
 - deployLibrary method
 - ServerLauncher, 250
 - deployLibraryEntry method
 - ServerLauncher, 250
 - deployment descriptors. *See*
 - application
 - servers:deployment
 - descriptors
 - DeploymentDescriptor class,
 - 260
 - getDefaultAccessor, 261
 - getDeploymentDescriptors,
 - 261
 - getFileEncoding, 261
 - getFullName, 261
 - DeployService class, 232
 - Designer interface, 111
 - activate, 111
 - annotate, 112
 - close, 112
 - example, 115
 - getModelName, 112
 - open, 112
 - Designer sample, 129
 - designerClosed method
 - DesignerListener, 108
 - designerClosing method

- DesignerListener, 108
- DesignerEvent class, 109
 - dispatch, 110
 - getComponent, 110
 - getContext, 110
 - getDesigner, 110
 - getShow, 110
 - getToolName, 110
 - isReleaseEvent, 110
- DesignerListener interface, 108
 - designerClosed, 108
 - designerClosing, 108
 - designerOpened, 108
 - designerOpening, 108
 - designerShow, 109
 - designerToolSelected, 109
- DesignerManager class, 107
 - addDesignerListener, 107
 - addDesignerReleaseListener, 107
 - example, 115
 - getDesigner, 107
 - getDesignerViewers, 107
 - getInstance, 107
 - lookupHelp, 107
 - registerDesigner, 107
 - removeDesignerListener, 108
 - removeDesignerReleaseListener, 108
 - designerOpened method
 - DesignerListener, 108
 - designerOpening method
 - DesignerListener, 108
 - designerReleased method
 - DesignerReleaseListener, 109
 - DesignerReleaseListener
 - interface, 108
 - designerReleased, 109
 - designerReleasing, 109
 - designerReleasing method
 - DesignerReleaseListener, 109
- designers, 105
 - events, 107, 108, 109
 - for InternetBeans, 113
 - registration, 107
- designerShow method
 - DesignerListener, 109
- designerToolSelected method
 - DesignerListener, 109
- DesignerViewer class
 - example, 98
- DesignView class, 136
 - adjustPositionForNib, 136
 - assureNibs, 136
 - componentAbsLocation, 136
 - constraintDialog, 137
 - example, 147, 148
 - getDesignerView, 136
 - getModel, 136
 - getTempComponent, 136
 - isConstraintEditorShowing, 137

- setModel, 137
- dialogs
 - default, 19
 - validating, 21
- DialogValidator interface, 21
 - validateDialog, 21
- Diff class, 21
 - addAddition, 21
 - addChange, 21
 - addDeletion, 22
 - diff, 22
 - editScriptToDiff, 22
 - iterator, 22
 - reset, 22
 - reverseIterator, 22
 - size, 23
 - toEditScript, 23
- diff method
 - Diff, 22
- DiffEntry class, 23
- directories
 - backup, 50, 206
 - class path, 45
 - documentation, 45
 - output, 51, 206
 - project base, 51
 - relative to project, 207
 - source files, 45, 46, 50
 - testing, 51
 - working, 51
- dispatch method
 - DesignerEvent, 110
 - JotFileEvent, 162
- doComparison method
 - AbstractRevisionNumber, 203
- doDefaultClick method
 - DefaultDialog, 20
- doesProjectTreeNeedRefreshed method
 - VCSUtils, 206
- doubleClickElement method
 - ListPanel, 36
- DummyPrintStream class, 23
- dumpPixelMaps method
 - AuraImage, 6

E

- editConstraints method
 - LayoutAssistant, 132
- editElement method
 - ListPanel, 36
- editor kits
 - example, 125
- editScriptToDiff method
 - Diff, 22
- editSelectedListElement method
 - ListPanel, 36
- EJB. *See* application servers
- EJBGRPFileNode class, 233, 239
- EjbService class, 232
- EMPTY_ARRAY field
 - PathSet, 48
 - Strings, 61
- enableAssert method
 - Debug, 16
- enableControls method
 - ListPanel, 36
- enableDragDrop method
 - SearchTree, 57
- enableOutput method
 - Debug, 17
- encode method
 - Strings, 60
 - Strings.StringEncoding, 62
- encodeArray method
 - Strings, 60
- encodeColor method
 - ColorCombo, 13
- encodeColors method
 - ColorCombo, 13
- encodeKeyStroke method
 - KeyStrokeEditorPanel, 32
- ensure method
 - Debug, 17
- ensureNonNullValue method
 - Server, 238
- ensureProjectContainsServer-ClientLibrary method
 - Server, 238
- ensureProjectContainsServer-Library method
 - Server, 238
- environment
 - JBuilder information, 26
 - platform information, 49
- escapeParameter method
 - ServerLauncher, 250
- eventChanged method
 - CmtComponentListener, 88
- events
 - for color panel, 14
 - for components, 85, 86, 87, 88
 - for designers, 107, 108, 109
 - for JOT, 160, 162
 - for list panels, 35, 37
 - for subcomponents, 88, 89, 91
- exactMatch method
 - RegularExpression, 53
- examples
 - BorderCornerLayout-Assistant, 142
 - CMTDump, 97
 - code download, vi
 - InternetBeansDesigner, 113
 - InternetBeansModel, 119
 - InternetBeansModelNode, 119

- InternetBeansViewer, 124
 - JBossSetup24, 268
 - JSPTagWizard, 179
 - SourceSafeVCS, 212
 - UISampler, 72
 - XMLValidator, 287
 - XSLTNodeViewer, 294
 - XSLTViewerFactory, 290
- F**
- features. *See* application servers:features
 - FILE_REVISION field
 - RevisionInfo, 203
 - fileClassChanged method
 - JotFileListener, 162
 - fileImportChanged method
 - JotFileListener, 162
 - fileMiscChanged method
 - JotFileListener, 162
 - filePackageChanged method
 - JotFileListener, 162
 - files
 - differences, 21
 - findBrowser method
 - example, 181
 - findDescriptor method
 - AbstractDescriptor-Conversion, 262
 - findEditor method
 - CmtComponentManager, 83
 - findFrame method
 - DefaultDialog, 20
 - findHit method
 - Compositelcon, 16
 - findPathUrl method
 - Classes, 10
 - findServerPathSet method
 - ServerManager, 226
 - findService method
 - ServerManager, 226
 - findSubstringMatch method
 - RegularExpression, 54
 - finish method
 - example, 182
 - fireComponentChanged method
 - CmtComponentSource, 86
 - firePropertyChange method
 - CmtSubcomponent, 89
 - fireSubcomponentChanged method
 - CmtComponentSource, 86
 - fixConflictForEjbGrpXmlSource method
 - VCSUtils, 206
 - fixConflictsForJavaSource method
 - VCSUtils, 206
 - fixedColors field
 - ColorPanel, 15
 - flush method
 - Debug, 17
 - format method
 - Strings, 61
 - formatJarFileParameter method
 - Server, 239
- G**
- generate method
 - AbstractDescriptor-Conversion, 262
 - generating Java, 155
 - getActionCommand method
 - ColorPanel, 14
 - getActions method
 - VCSFileStatus, 204
 - getActiveVCS method
 - VCSUtils, 207
 - getActiveVCSName method
 - VCSUtils, 207
 - getAddButton method
 - ListPanel, 36
 - getAllAvailableFeatures method
 - Service, 232
 - getAllAvailableSpecFeatures method
 - Service, 232
 - getAllChildren method
 - ZipIndex, 69
 - getAllowPackages method
 - PackageBrowserDialog, 39
 - getAlphaThreshold method
 - AuralImage, 6
 - getArchivesToDeployOnRun method
 - ServerLauncher, 250
 - getArguments method
 - ServerLauncher, 250
 - getAsContainer method
 - CmtSubcomponent, 89
 - example, 147
 - getAssignment method
 - CmtSubcomponent, 89
 - example, 99
 - JotExpression, 176
 - getAssignments method
 - JotCodeBlock, 173
 - getAssociatedJdk method
 - Server, 239
 - getAssociatedModuleType method
 - Service, 232
 - getAuralImage method
 - AuralImage, 6
 - Images, 25
 - getAuraRGB method
 - AuralImage, 6
 - getAuthor method
 - RevisionInfo, 202
 - getAutoProperty method
 - example, 214
 - getAuxPath method
 - ProjectPathSet, 50
 - getAuxPaths method
 - ProjectPathSet, 50
 - getAvailableSpecFeaturesForAssociatedModuleType method
 - Service, 233
 - getAvailableTypes method
 - ServerManager, 226
 - getBackupUrl method
 - VCSUtils, 207
 - getBakPath method
 - ProjectPathSet, 50
 - getBeanDataSource method
 - AbstractDescriptor-Conversion, 263
 - getBeanDataSourceName method
 - AbstractDescriptor-Conversion, 263
 - getBlankIcon method
 - Icons, 24
 - getBlendedImage method
 - AuralImage, 6
 - getBoundsAsString method
 - DefaultDialog, 20
 - getBrowser method
 - example, 182
 - SetupPropertyPage, 270
 - getBrowserActiveNode method
 - VCSUtils, 207
 - getBuffer method
 - example, 125
 - getBufferContent method
 - example, 127
 - getBuildNumber method
 - JBuilderInfo, 26
 - getBytes method
 - AbstractDeployment-Descriptor, 259
 - getCatches method
 - example, 190
 - getChildren method
 - CmtModel, 96
 - example, 120
 - ZipIndex, 69
 - getClass method
 - JotFile, 159
 - JotPackages, 157
 - getClasses method
 - JotFile, 159
 - getClassPath method
 - PathSet, 44
 - Server, 239
 - getClazz method
 - JotFileEvent, 162
 - getClientJarService method
 - Server, 239
 - getClientLibraryClassPath method
 - method

- Server, 239
- getClientLibraryName method
 - Server, 239
- getClientVmParameters method
 - Service, 233
- getCmtModel method
 - example, 122, 123
- getCodeBlock method
 - example, 187
 - JotMethodSource, 170
 - JotStatement, 177
- getCollection method
 - PathSet, 44
- getCommand method
 - ServerLauncher, 250
- getComment method
 - JotCommentable, 175
 - RevisionInfo, 202
 - VCSFileInfo, 204
- getCommentText method
 - JotComment, 175
- getCompanionNode method
 - Server, 239
 - Service, 233
- getCompanyName method
 - JBuilderInfo, 26
- getComparableLocation method
 - JotClassSource, 168
 - JotCodeBlock, 173
 - JotSourceFile, 160
- getComponent method
 - CmtComponents, 82
 - CmtFeature, 92
 - CmtModel, 96
 - CmtSubcomponent, 89
 - DesignerEvent, 110
 - example, 120
- getComponentSource method
 - example, 146
- getComponentType method
 - CmtSubcomponent, 89
 - JotClass, 163
- getCondition method
 - JotExpression, 176
- getConstraints method
 - example, 146
- getConstraintsType method
 - example, 145
 - LayoutAssistant, 132
- getConstructor method
 - JotClass, 163
- getConstructors method
 - JotClass, 164
- getContainerDelegate method
 - CmtComponent, 83
- getContent method
 - example, 125
- getContents method
 - TextFile, 66
- getContext method
 - DesignerEvent, 110
- getCopy method
 - PathSet, 45
 - Server, 239
- getCurrentWorkingDirectory method
 - ServerLauncher, 250
- getCustomColors method
 - ColorCombo, 13
 - ColorPanel, 14
- getCustomConfigurationPage-Factory method
 - Server, 239
- getCustomizedRunDebugClass-Path method
 - Service, 233
- getCustomizerClass method
 - CmtComponent, 83
- getCustomizerDialog method
 - CmtSubcomponent, 89
- getDate method
 - RevisionInfo, 202
- getDaysLeft method
 - JBuilderInfo, 27
- getDeclaredClass method
 - CmtSubcomponent, 89
 - example, 99
- getDeclaredConstructor method
 - JotClass, 164
- getDeclaredConstructors method
 - JotClass, 164
- getDeclaredField method
 - JotClass, 164
- getDeclaredFields method
 - JotClass, 164
- getDeclaredInnerClasses method
 - JotClass, 164
 - JotCodeBlock, 173
- getDeclaredMethod method
 - JotClass, 164
- getDeclaredMethods method
 - JotClass, 164
- getDeclaredModifiers method
 - JotClassSource, 168
 - JotMethodSource, 171
- getDeclaringClass method
 - JotMethod, 169
- getDeclaringFile method
 - JotClassSource, 168
- getDefaultAccessor method
 - AbstractDeployment-Descriptor, 259
 - DeploymentDescriptor, 261
- getDefaultArguments method
 - ServerLauncher, 250
- getDefaultClassPath method
 - Server, 239
- getDefaultEventIndex method
 - CmtComponent, 84
- getDefaultEventState method
 - CmtSubcomponent, 89
 - example, 99
- getDefaultHandlerText method
 - CmtEventState, 95
 - example, 99
- getDefaultHomeDirectory method
 - Server, 239
- getDefaultName method
 - Server, 239
- getDefaultNecessaryArguments method
 - ServerLauncher, 250
- getDefaultNecessaryVm-Parameters method
 - ServerLauncher, 250
- getDefaultPropertyIndex method
 - CmtComponent, 84
- getDefaultPropertyState method
 - CmtSubcomponent, 89
 - example, 99
- getDefaultServerName method
 - Server, 239
- getDefaultSourcePath method
 - ProjectPathSet, 50
- getDefaultSupportedSpec-Features method
 - Service, 233
- getDefaultVersion method
 - Server, 239
- getDefaultVmParameters method
 - ServerLauncher, 250
- getDefaultWorkingDirectory method
 - ServerLauncher, 250
- getDependencies method
 - Service, 233
- getDeploymentDescriptors method
 - DeploymentDescriptor, 261
- getDeployService method
 - Server, 240
- getDescription method
 - example, 216
 - JBuilderInfo, 27
 - NestingSetupPropertyPage, 271
 - SetupPage, 270
 - SetupPropertyPage, 270
 - VCS, 199
- getDescriptorDocument method
 - AbstractDescriptor-Conversion, 263
- getDesigner method
 - DesignerEvent, 110
 - DesignerManager, 107
- getDesignerView method
 - DesignView, 136
- getDesignerViewers method
 - DesignerManager, 107
- getDirectory method
 - ZipIndexEntry, 71
- getDisabledImage method
 - Images, 25
- getDisableIcon method

- Icons, 24
- getDisplayName method
 - CmtFeature, 92
- getDocPath method
 - PathSet, 45
- getEditButton method
 - ListPanel, 36
- getEditor method
 - CmtProperty, 92
- getEjbJarProperties method
 - AppServerTargeting, 256
- getEjbNode method
 - AbstractDescriptor-
 - Conversion, 263
- getEjbProperties method
 - AppServerTargeting, 256
- getEjbService method
 - Server, 240
- getElementName method
 - ListPanel, 36
- getElse method
 - example, 190
- getEmptyDescription method
 - PathSet, 45
- getEnabledServers method
 - ServerManager, 226
- getEncoding method
 - AbstractDeployment-
 - Descriptor, 259
 - JotPackages, 157
- getEndPosition method
 - JotMarker, 161
- getEnvironment method
 - ServerLauncher, 251
- getEnvpWithPathVariablePrefix method
 - ServerLauncher, 251
- getErrorMessage method
 - CommitAction, 211
- getEvent method
 - CmtComponent, 84
- getEvents method
 - CmtComponent, 84
- getEventState method
 - CmtSubcomponent, 89
- getEventStates method
 - CmtSubcomponent, 89
 - example, 99
- getException method
 - CmtComponent, 84
- getExcludedPaths method
 - VCSUtils, 207
- getExistingZipFile method
 - ZipIndex, 69
- getExpandedIcon method
 - CheckTreeNode, 9
- getExpansionPackNames method
 - JBuilderInfo, 27
- getExpansionState method
 - SearchTree, 57
- getExtraDescriptions method
 - JBuilderInfo, 27
- getExtraLocation method
 - AbstractDeployment-
 - Descriptor, 259
- getExtraVisiBrokerTool-
 - Parameters method
 - Server, 240
- getFarthestPath method
 - PackageBrowserTree, 42
 - SearchTree, 57
- getFeatureDefinition method
 - Service.Type, 231
- getFeatures method
 - Service, 233
- getField method
 - JotClass, 165
- getFields method
 - JotClass, 165
- getFile method
 - CmtComponent, 84
 - JotClass, 165
 - JotFileEvent, 163
 - JotPackages, 158
 - VCSFileInfo, 204
- getFileAccessor method
 - AbstractDeployment-
 - Descriptor, 260
- getFileEncoding method
 - DeploymentDescriptor, 261
- getFileNameBasedOnProtocol method
 - Server, 240
- getFilenames method
 - ZipIndex, 69
- getFileObject method
 - example, 215
- getFiles method
 - JotPackages, 158
- getFilesArray method
 - JotPackages, 158
- getFilesNeededByVCS method
 - VCSUtils, 207
- getFirstParameter method
 - JotMethod, 169
- getFromBag method
 - ServerLauncher, 251
- getFullClassName method
 - JotSourceFile, 160
- getFullClassPath method
 - PathSet, 45
 - Server, 240
- getFullDocPath method
 - PathSet, 45
- getFullLibPath method
 - ProjectPathSet, 50
- getFullLibraryClassPath method
 - Server, 240
- getFullName method
 - DeploymentDescriptor, 261
 - JotType, 169
 - PathSet, 45
 - Server, 240
- getFullPath method
 - PackageBrowserTree, 42
- getFullSourcePath method
 - PathSet, 45
- getGenericDDName method
 - AbstractDescriptor-
 - Conversion, 263
- getGraph method
 - CmtModel, 96
 - example, 120, 123
- getHeadingSpace method
 - Text, 65
- getHomeDirectory method
 - Server, 240
- getIcon method
 - CheckTreeNode, 9
 - CmtComponent, 84
 - Icon.IconFactory, 25
 - Icons, 24
 - PathSet, 45
 - Service.Type, 231
- getIconFactory method
 - Icons, 24
- getImage method
 - Images, 25, 26
- getImport method
 - JotFileEvent, 163
 - JotSourceFile, 160
- getImports method
 - JotSourceFile, 160
- getIncludeTestPath method
 - ProjectPathSet, 50
- getIncompleteDescription method
 - PathSet, 45
 - Server, 240
- getIndentColumn method
 - Text, 65
- getIndentLevel method
 - JotSourceElement, 174
- getInitializer method
 - CmtSubcomponent, 89
 - example, 99
- getInitMethod method
 - CmtComponentSource, 86
 - CmtSubcomponent, 90
 - example, 99
- getInnerClasses method
 - JotClass, 165
- getInstance method
 - DesignerManager, 107
- getInterfaces method
 - JotClass, 165
- getJarTarget method
 - AppServerTargeting, 256
- getJavaInitializationString method
 - KeyStrokeEditorPanel, 32
- getJavaLauncher method
 - ServerLauncher, 251
- getJDK method
 - ProjectPathSet, 50

- getJDKPathSet method
 - ProjectPathSet, 50
- getJDKs method
 - ProjectPathSet, 50
- getJdkSupportProvider method
 - Server, 240
- getJotClass method
 - JotType, 169
- getJotClassSource method
 - JotType, 169
- getJotMethods method
 - AbstractDescriptor-
Conversion, 263
- getJotPackages method
 - example, 185
- getJspServletService method
 - Server, 240
- getKeyName method
 - KeyStrokeEditorPanel, 32
- getKeyStroke method
 - KeyStrokeDialog, 31
 - KeyStrokeEditorPanel, 32
 - KeyStrokeEditorTextField, 33
- getKeyStrokeName method
 - KeyStrokeEditorPanel, 32
- getKeyStrokeText method
 - KeyStrokeEditorPanel, 32
- getKeyText method
 - KeyStrokeEditorPanel, 33
- getLabel method
 - RevisionInfo, 202
 - ServerLauncher, 251
- getLabels method
 - RevisionInfo, 202
- getLastDesignedNode method
 - CmtComponentSource, 86
- getLastModificationSaved method
 - PathSet, 45
- getLastModified method
 - PathSet, 45
 - Server, 240
 - ZipIndex, 69
 - ZipIndexEntry, 71
- getLastRegisteredServer method
 - ServerManager, 226
- getLayoutAssistant method
 - SelectNib, 138
- getLegacyFullName method
 - Server, 240
- getLength method
 - RegularExpression, 54
- getLibKit method
 - PathSet, 45
- getLibKits method
 - ProjectPathSet, 51
- getLibPath method
 - ProjectPathSet, 51
- getLibraries method
 - ProjectPathSet, 51
- getLibrary method
 - ProjectPathSet, 51
- getLibraryClassesRelativePath method
 - ServerLauncher, 251
- getLibraryDestination method
 - ServerLauncher, 251
- getList method
 - ListPanel, 36
- getListCellRendererComponent method
 - ListPanel, 36
- getListScrollPane method
 - ListPanel, 36
- getLiveClass method
 - CmtSubcomponent, 90
 - example, 99
- getLiveClazz method
 - CmtComponent, 84
 - example, 99
- getLiveInstance method
 - CmtSubcomponent, 90
 - example, 116, 120, 123
- getLiveType method
 - CmtComponent, 84
- getLocalRevisions method
 - VCSUtils, 207
- getManager method
 - CmtComponent, 84
- getMergeConflictDividerMarker method
 - VCS, 199
- getMergeConflictEndMarker method
 - VCS, 199
- getMergeConflictStartMarker method
 - VCS, 199
- getMethod method
 - CmtComponent, 84
 - JotClass, 165
- getMethodCall method
 - CmtPropertySetting, 95
 - example, 99
 - JotCodeBlock, 173
 - JotExpression, 176
- getMethodCalls method
 - example, 99
- getMethodCalls method
 - CmtSubcomponent, 90
 - JotCodeBlock, 173
- getMethods method
 - CmtComponent, 84
 - JotClass, 165
- getMode method
 - PackageBrowserTree, 42
- getModel method
 - CmtComponentSource, 86
 - CmtModelNode, 97
 - DesignView, 136
- getModelName method
 - Designer, 112
 - example, 117
- getModels method
 - CmtComponentSource, 86
- getModelTree method
 - CmtComponentSource, 87
- getModifiers method
 - JotClass, 166
 - JotMethod, 169
- getMoveDownButton method
 - ListPanel, 36
- getMoveUpButton method
 - ListPanel, 36
- getName method
 - AbstractDeployment-
Descriptor, 260
 - CmtComponentSource, 87
 - CmtFeature, 92
 - CmtModel, 96
 - CmtSubcomponent, 90
 - example, 100, 121, 216, 269
 - JotClass, 166
 - JotFile, 159
 - JotMethod, 169
 - JotType, 169
 - PathSet, 46
 - ProjectPathSet, 51
 - Server, 241
 - Service.Type, 231
 - Setup, 267
 - VCS, 199
 - VCSFileInfo, 204
 - ZipIndexEntry, 71
- getNameFromFullName method
 - Server, 241
- getNames method
 - VCSFactory, 199
- getNearestPath method
 - SearchTree, 57
- getNecessaryArguments method
 - ServerLauncher, 251
- getNecessaryVmParameters method
 - ServerLauncher, 251
- getNew method
 - JotExpression, 176
- getNewPathsBasedOnNew-
HomeDirectory method
 - Server, 241
- getNewProjectFromVCS-
WizardAction method
 - VCS, 199
- getNibBounds method
 - SelectNib, 138
- getNode method
 - example, 182, 293
- getNodeValue method
 - Server, 241
- getNoneServerItem method
 - ServerManager, 226
- getNormalText method
 - JotComment, 175
- getOpenZipIndexes method
 - ZipIndex, 69

- getOperation method
 - JotExpression, 176
- getOptimizerPackages method
 - Server, 241
- getOuterComponent method
 - CmtSubcomponent, 90
 - example, 100
- getOutPath method
 - ProjectPathSet, 51
- getPackage method
 - JotFile, 159
 - JotPackages, 158
- getPackageBrowserFilter method
 - PackageBrowserTree, 42
- getPackageManager method
 - JotFile, 159
- getPackages method
 - CmtComponents, 82
 - JotPackages, 158
 - Server, 241
- getPackagesArray method
 - JotPackages, 158
- getPageComponent method
 - example, 183
- getPageFactory method
 - Setup, 267
- getPageName method
 - AppServerTargeting, 256
- getParameter method
 - JotMethod, 169
- getParameters method
 - JotMethod, 170
- getParameterTypes method
 - JotMethod, 170
- getParent method
 - JotSourceElement, 174
- getPathNode method
 - PackageBrowserTree, 42
- getPathNodes method
 - PackageBrowserTree, 42
- getPathRelativeToProject-Directory method
 - VCSUtils, 207
- getPaths method
 - example, 181
- getPathSet method
 - Server, 241
- getPathSetReferenceClass method
 - PathSet, 46
- getPathTime method
 - ProjectPathSet, 51
- getPersonalExcludedPaths method
 - VCSUtils, 208
- getPersonalities method
 - example, 181
 - Setup, 267
- getPopupAlignment method
 - ColorCombo, 13
- getPopupGridHeight method
 - ColorCombo, 13
- getPrecedence method
 - AbstractRevisionNumber, 203
- getPreserveMode method
 - SearchTree, 57
- getPrimaryServer method
 - ServerManager, 227
- getProject method
 - AbstractDescriptor-Conversion, 263
 - CmtComponents, 82
 - PackageBrowserTree, 42
 - ServerLauncher, 251
 - SetupPropertyPage, 270
- getProjectConfigPage method
 - example, 214
 - VCS, 199
- getProjectConfigPageNew method
 - VCS, 199
- getProjectLibraries method
 - ProjectPathSet, 51
- getProjectLibrariesForRun method
 - ServerLauncher, 251
- getProjectPath method
 - example, 219
- getProjectPropertiesPage method
 - Service, 234
- getProjectStatus method
 - example, 216
 - VCS, 200
- getProperties method
 - CmtComponent, 84
 - PathSet, 46
- getProperty method
 - CmtComponent, 84
 - CmtPropertySetting, 95
 - CmtPropertyState, 94
 - example, 99
 - PathSet, 46
- getPropertyEditor method
 - example, 145
 - LayoutAssistant, 133
- getPropertyFromSetter method
 - CmtComponent, 84
- getPropertyKey method
 - Service.Type, 231
- getPropertyMap method
 - ServerLauncher, 251
- getPropertyNamesToSurface method
 - AbstractDescriptor-Conversion, 263
- getPropertyPage method
 - CommitAction, 211
- getPropertySetting method
 - CmtPropertyState, 94
 - example, 99
- getPropertyState method
 - CmtSubcomponent, 90
- getPropertyStates method
 - CmtSubcomponent, 90
 - example, 99
- getQueuedExceptions method
 - ServerLauncher, 251
- getRawBuildNumber method
 - JBuilderInfo, 27
- getRawLastModified method
 - ZipIndex, 69
 - ZipIndexEntry, 71
- getReadMethod method
 - CmtProperty, 92
- getRectangleDimension method
 - example, 147, 148
 - SelectNib, 138
- getRectangleLocation method
 - example, 148
 - SelectNib, 138
- getRefactorCheckoutAction method
 - VCS, 200
- getReferenceName method
 - PathSet, 46
- getRelativePath method
 - VCSUtils, 208
- getRemoveButton method
 - ListPanel, 36
- getRequire method
 - PathSet, 46
- getRequiredName method
 - PathSet, 46
- getResolver method
 - PathSet, 46
- getResourcePath method
 - ProjectPathSet, 51
- getResult method
 - KeyStrokeEditorPanel, 33
- getReturnType method
 - JotMethod, 170
- getRevisionNumber method
 - RevisionInfo, 202
- getRevisionNumberInstance method
 - AbstractRevisionNumber, 203
- getRevisions method
 - example, 217
 - VCS, 200
- getRevisionString method
 - AbstractRevisionNumber, 203
- getRoot method
 - CmtModel, 96
 - example, 120, 123
- getRootEntryFromClasspath method
 - Classes, 11
- getRunConfigPropertyPage method
 - Service, 234
- getRunConfigPropertyPages method
 - method

- ServerLauncher, 251
- getScope method
 - CmtSubcomponent, 90
 - example, 100
- getSelectedColor method
 - ColorCombo, 13
 - ColorPanel, 14
- getSelectedIndex method
 - ListPanel, 36
- getSelectedIndices method
 - ListPanel, 36
- getSelectedListElement method
 - ListPanel, 37
- getSelectedListElements method
 - ListPanel, 37
- getSelectedNode method
 - PackageBrowserTree, 42
- getSelectedNodes method
 - PackageBrowserTree, 42
- getSelectedNodesInProjectPane method
 - VCSUtils, 208
- getSelectedPath method
 - PackageBrowserTree, 43
- getSelectedPaths method
 - PackageBrowserTree, 43
- getSelectionModel method
 - ListPanel, 37
- getSelectionState method
 - SearchTree, 57
- getServer method
 - AbstractDescriptor-
Conversion, 263
 - AppServerTargeting, 256
 - ServerLauncher, 252
 - ServerManager, 227
 - Service, 235
- getServers method
 - ServerManager, 227
- getServerTypeId method
 - Server, 241
- getService method
 - ServerLauncher, 252
 - ServerManager, 227
 - Service, 235
- getServices method
 - Server, 242
 - ServerLauncher, 252
 - ServerManager, 227
 - Service, 235
- getServiceType method
 - ServerManager, 227
 - Service, 235
- getServiceTypeKeys method
 - Service, 235
- getServiceTypes method
 - ServerManager, 227
- getSetup method
 - SetupManager, 265
- getSetupLauncher method
 - Server, 242
- getSetupPersonalities method
 - SetupManager, 265
- getSetupPropertyPage method
 - example, 269
 - Setup, 267
- getSetups method
 - SetupManager, 265
- getShortDescription method
 - CmtFeature, 92
- getShortName method
 - Classes, 11
 - Server, 242
- getShortNameWithVersion method
 - method
 - Server, 242
- getShow method
 - DesignerEvent, 110
- getShowEventType method
 - KeyStrokeEditorTextField, 34
- getShowKeyReleased method
 - KeyStrokeEditorTextField, 34
- getShowKeyTyped method
 - KeyStrokeEditorTextField, 34
- getShutdownWaitTime method
 - ServerLauncher, 252
- getSKU method
 - JBuilderInfo, 27
- getSKUDescription method
 - JBuilderInfo, 27
- getSKUName method
 - JBuilderInfo, 27
- getSkuVersion method
 - Service.Type, 231
- getSortedRowIndex method
 - TableSorter, 63
- getSource method
 - example, 215
 - VCS, 200
- getSourceBridge method
 - ServerLauncher, 252
- getSourceBridges method
 - ServerLauncher, 252
- getSourceFile method
 - CmtComponentSource, 87
 - example, 146, 185
 - JotPackages, 158
- getSourceImage method
 - AuraImage, 6
- getSourceName method
 - CmtSubcomponent, 90
 - example, 100
- getSourcePath method
 - example, 181
 - PathSet, 46
- getSourceVersion method
 - JotPackages, 158
- getStartPosition method
 - JotMarker, 161
- getStatements method
 - JotCodeBlock, 173
- JotMethodSource, 171
- JotStatement, 177
- getStatus method
 - VCSFileInfo, 204
 - VCSFileStatus, 204
- getStatusIcon method
 - VCSFileStatus, 204
- getStopper method
 - ServerLauncher, 252
- getStringPath method
 - SearchTree, 57
- getSubcomponent method
 - CmtComponentSource, 87
 - CmtModelNode, 97
 - CmtPropertySetting, 95
 - CmtPropertyState, 94
 - example, 116, 121, 122, 123, 147
- getSubcomponents method
 - CmtComponentSource, 87
 - example, 98, 116
- getSubStrings method
 - Strings, 61
- getSummaryText method
 - JotComment, 175
- getSuperclass method
 - JotClass, 166
- getTag method
 - CmtModelNode, 97
- getTempComponent method
 - DesignView, 136
 - example, 148
- getTestPath method
 - ProjectPathSet, 51
- getText method
 - CheckTreeNode, 9
 - JotSourceElement, 174
- getTexture method
 - TexturePanel, 67
- getThen method
 - example, 188
- getThrowSpecifiers method
 - JotMethod, 170
- getTimestamp method
 - AbstractDeployment-
Descriptor, 260
 - JotFile, 159
- getToolName method
 - DesignerEvent, 110
- getTracker method
 - ServerLauncher, 252
- getType method
 - CmtComponent, 84
 - CmtProperty, 93
 - JotClass, 166
 - JotComment, 175
- getUniqueRunDebugClassPath method
 - Server, 242
- getUnsortedRowIndex method
 - TableSorter, 63
- getUrl method

- JotFile, 159
- JotPackages, 158
- PathSet, 46
- Server, 242
- VCSFileInfo, 204
- getUserData method
 - JotClass, 166
 - JotMarker, 161
- getUserName method
 - JBuilderInfo, 27
- getValue method
 - CmtPropertySetting, 95
 - CmtPropertyState, 94
 - example, 99
 - JotExpression, 176
- getValueAt method
 - TableSorter, 63
- getValues method
 - example, 292
- getValueSource method
 - CmtPropertySetting, 95
 - CmtPropertyState, 94
 - example, 99, 146
- getValueText method
 - CmtPropertyState, 94
 - example, 99, 146
 - KeyStrokeEditorPanel, 33
- getVariable method
 - JotExpression, 176
- getVariableDeclaration method
 - JotCodeBlock, 173
- getVariableDeclarations method
 - JotCodeBlock, 173
- getVCS method
 - VCSFactory, 199
- getVCSContextMenuGroup
 - method
 - example, 217
 - VCS, 200
- getVCSFileActions method
 - VCSFileStatus, 205
- getVCSFileMenuGroup method
 - example, 216
 - VCS, 200
- getVCSGlobalMenuGroup
 - method
 - VCS, 201
- getVCSIcon method
 - example, 216
 - VCS, 201
- getVCSProjectMenuGroup
 - method
 - example, 216
 - VCS, 201
- getVersion method
 - Server, 242
- getViewerDescription method
 - example, 295
- getViewerTitle method
 - example, 295
- getVKText method
 - KeyStrokeEditorPanel, 33

- getVmParameters method
 - ServerLauncher, 252
- getWaitForServerThread method
 - ServerLauncher, 252
- getWeight method
 - Server, 242
- getWorkingDirectory method
 - ProjectPathSet, 51
 - ServerLauncher, 252
- getWorkingDirectoryFrom-
HomeDirectory method
 - ServerLauncher, 252
- getWriteMethod method
 - CmtProperty, 93
- getZipEntries method
 - ZipIndex, 70
- getZipIndex method
 - ZipIndex, 69
- getZipIndexEntry method
 - ZipIndex, 70

H

- handleOldStyleProjects method
 - VCSUtils, 208
- hasClientJarCreator method
 - Server, 242
- hasEjbDeployer method
 - Server, 242
- hasEjbJarProperties method
 - AppServerTargeting, 257
- hasEjbProperties method
 - AppServerTargeting, 257
- hasSetup method
 - Server, 242
- help method
 - example, 183
- HelpManager class
 - example, 183
- hide method
 - SelectBoxes, 137
 - ZipIndex, 70
- hideAll method
 - SelectBoxes, 137
 - ZipIndex, 70

I

- IBM_LINUX field
 - Platform, 49
- IBM_VENDOR field
 - Platform, 49
- Icon.IconFactory class, 25
 - getIcon, 25
- icons
 - adding auras, 5
 - composites, 15
 - disabled, 24
 - factory, 25
 - for check tree nodes, 9
 - for components, 84
 - for path sets, 45
 - for service types, 231
 - for VCS, 201
 - for VCS status, 204
 - utilities, 24
- Icons class, 24
 - getBlankIcon, 24
 - getDisabledIcon, 24
 - getIcon, 24
 - getIconFactory, 24
- images
 - utilities, 25
- Images class, 25
 - getAuralImage, 25
 - getDisabledImage, 25
 - getImage, 25, 26
 - waitForImage, 26
- inInstance method
 - JotClass, 166
- init method
 - ServerLauncher, 252
- INIT_METHOD_NAME field
 - CmtComponentSource, 87
- INIT_METHOD_PARAMS field
 - CmtComponentSource, 87
- initialize method
 - Server, 242
- initializeSetupPage method
 - Setup, 267
- initLauncher method
 - ServerLauncher, 252
- inMultiInstance method
 - CmtModel, 96
- InternetBeans
 - designer, 113
- InternetBeansDesigner example, 113
- InternetBeansModel example, 119
- InternetBeansModelNode
 - example, 119
- InternetBeansViewer example, 124
- INVALID_SERVER field
 - Server, 247
- invokeWizard method
 - example, 181
- isAddToProjectEnabled method
 - PackageBrowserDialog, 39
- isAddToProjectSelected method
 - PackageBrowserDialog, 39
- isAddToProjectVisible method
 - PackageBrowserDialog, 39
- isAffectedByParentEnabled method
 - CheckTreeNode, 9
- isArray method
 - JotClass, 166
- isAssignableFrom method
 - JotClass, 166
- isAutoCenter method

- DefaultDialog, 20
- isBeaEnabled method
 - JBuilderInfo, 28
- isBean method
 - CmtComponent, 85
- isBinary method
 - example, 216
 - VCS, 201
- isBinaryFileNode method
 - VCSUtils, 208
- isBound method
 - CmtProperty, 93
- isBreakOnNewLine method
 - RegularExpression, 54
- isCancellable method
 - CommitAction, 212
- isCaseSensitive method
 - RegularExpression, 54
- isCheckable method
 - CheckTreeNode, 9
- isChecked method
 - CheckTreeNode, 9
- isClassComboVisible method
 - PackageBrowserDialog, 39
- isClassNameInTree method
 - PackageBrowserTree, 43
- isCommentRequired method
 - VCSFileStatus, 205
- isComponentEnabled method
 - JBuilderInfo, 28
- isConfigureVCSMenuEnabled method
 - example, 217
 - VCS, 201
- isConstant method
 - JotExpression, 176
- isConstrained method
 - CmtProperty, 93
- isConstraintEditorShowing method
 - DesignView, 137
- isContainer method
 - CmtComponent, 85
- isCopy method
 - Server, 243
- isDefault method
 - CmtPropertyState, 94
 - example, 99
- isDefaultEnabled method
 - Service.Type, 231
- isDesignable method
 - CmtModelNode, 97
 - example, 123
- isDirectory method
 - ZipIndex, 70
 - ZipIndexEntry, 72
- isEjbJarDirty method
 - AbstractDescriptor-
Conversion, 264
- isEmpty method
 - PathSet, 46
 - Strings, 61
- isEnabled method
 - CheckTreeNode, 9
 - example, 269
 - PathSet, 46
 - Setup, 268
- isEnabledForSku method
 - Service.Type, 231
- isEnabledmentAffectedByParent method
 - CheckTreeNode, 9
- isEntEnabled method
 - JBuilderInfo, 28
- isEnterpriseEnabled method
 - JBuilderInfo, 28
- isExpert method
 - CmtFeature, 92
- isFileType method
 - VCSUtils, 208
- isFoundationOnlyEnabled method
 - JBuilderInfo, 28
- isGenericPremiumEnabled method
 - JBuilderInfo, 28
- isGenericValueEnabled method
 - JBuilderInfo, 28
- isGranula method
 - Service, 235
- isGranular method
 - Server, 243
- isHidden method
 - CmtFeature, 92
 - ZipIndex, 70
- isHiddenState method
 - CmtComponent, 85
- isIncomplete method
 - PathSet, 47
 - Server, 243
- isInitiallySetup method
 - Server, 243
- isInProjectDirectory method
 - VCSUtils, 208
- isInterface method
 - JotClass, 166
- isLibraryEnabled method
 - JBuilderInfo, 28
- isLicensed method
 - JBuilderInfo, 28
- isLocked method
 - CheckTreeNode, 9
- isMacLAF method
 - Platform, 49
- isModified method
 - JotSourceElement, 174
 - SetupPropertyPage, 270
- isModifiedInVCS method
 - VCSFileStatus, 205
- isModifiedLocally method
 - VCSFileStatus, 205
- isModuleDirty method
 - AbstractDescriptor-
Conversion, 264
- isMultiInstance method
 - example, 120
- isNeedsSerialize method
 - CmtSubcomponent, 90
- isNew method
 - VCSFileStatus, 205
- isNull method
 - JotExpression, 176
- isOpen method
 - ZipIndex, 70
- isPackageName method
 - PackageBrowserTree, 43
- isPageValid method
 - Server, 243
- isPatternMatch method
 - RegularExpression, 54
- isPrimitive method
 - JotClass, 166
- isProEnabled method
 - JBuilderInfo, 28
- isPseudoPropertyState method
 - CmtPropertyState, 94
 - example, 99
- isReadable method
 - CmtProperty, 93
- isReadOnly method
 - CmtComponent, 85
 - CmtPropertyState, 94
 - JotSourceFile, 161
 - PathSet, 47
- isRegExpMatch method
 - RegularExpression, 54
- isReleaseEvent method
 - DesignerEvent, 110
- isRuntime method
 - Service.Type, 232
- isSeEnabled method
 - JBuilderInfo, 28
- isSelectable method
 - SelectNib, 138
- isSelectVCSMenuEnabled method
 - VCS, 201
- isServerEnabled method
 - Server, 243
- isSetup method
 - Server, 243
- isSetupCompleted method
 - Server, 243
- isShowDescription method
 - Setup, 268
- isShowReopenWarning method
 - SetupManager, 266
- isShowRestartWarning method
 - SetupManager, 266
- isSpecialChar method
 - RegularExpression, 54
- isSpecialDown method
 - Platform, 49
- isStudio method
 - JBuilderInfo, 29
- isSubcomponentOwned method

- CmtModel, 96
 - example, 120
- isSybaseEnabled method
 - JBuilderInfo, 29
- isTermLicense method
 - JBuilderInfo, 29
- isTrial method
 - JBuilderInfo, 29
- isUnderVCS method
 - example, 214
 - RevisionInfo, 202
 - VCS, 201
- isValidSetupDirectory method
 - Server, 243
- isValidWorkingDirectory
 - method
 - ServerLauncher, 253
- isVCSFileOrDir method
 - VCSUtils, 208
- isVisible method
 - SelectBoxes, 137
- isWorkingRevision method
 - RevisionInfo, 202
- isWritable method
 - CmtProperty, 93
- iterator method
 - Diff, 22

J

- JAR files. *See* Zip files
- JarFileNode class, 233, 239
- Java
 - generating, 155
 - parsing, 155
- Java Object Toolkit. *See* JOT
- JavaBeans, 79
 - customizers, 83
 - icons, 84
- jbInit method, 79
- JBoss
 - JBuilder integration, 225
- JBossSetup24 example, 268
- JBOWNER_METHOD_NAME
 - field
 - CmtComponentSource, 87
- JBuilderInfo class, 26
 - actionVerify, 26
 - getBuildNumber, 26
 - getCompanyName, 26
 - getDaysLeft, 27
 - getDescription, 27
 - getExpansionPackNames, 27
 - getExtraDescriptions, 27
 - getRawBuildNumber, 27
 - getSKU, 27
 - getSKUDescription, 27
 - getSKUName, 27
 - getUserName, 27
 - isBeaEnabled, 28
 - isComponentEnabled, 28
 - isEntEnabled, 28
 - isEnterpriseEnabled, 28
 - isFoundationOnlyEnabled, 28
 - isGenericPremiumEnabled, 28
 - isGenericValueEnabled, 28
 - isLibraryEnabled, 28
 - isLicensed, 28
 - isProEnabled, 28
 - isSeEnabled, 28
 - isStudio, 29
 - isSybaseEnabled, 29
 - isTermLicense, 29
 - isTrial, 29
 - launchWizard, 29
 - setStatusMsg, 29
 - showInfoDialog, 29
- JDataStoreService class, 232
- JDKPathSet class, 48
 - retrieving, 50
- JOT, 79, 155
 - adding a class, 160
 - blank lines, 174
 - class diagram, 156
 - class modifiers, 166, 168
 - classes, 163, 167
 - comments, 174, 175
 - constructors, 163, 164, 167, 168
 - creating objects, 167
 - deleting a, 161
 - events, 160, 161, 162
 - example, 179
 - expressions, 176
 - fields, 164, 165, 167, 168
 - generating classes, 167
 - get Url, 159
 - import statement, 160, 161
 - inner classes, 164, 165, 167, 168, 172, 173
 - interfaces, 165, 166, 168
 - locations, 160, 161
 - methods, 164, 165, 168, 169, 170, 172
 - open a file, 158
 - package statement, 161
 - parsing classes, 163
 - save changes, 157
 - starting point, 157
 - statements, 170, 171, 173, 177
 - static initializers, 167, 168
 - super, 166
 - superclass, 169
 - tidy up, 158
 - types, 169
 - variables, 173
- JotAnonymousClass interface, 167
- JotAssignment interface, 176
- JotBinaryExpression interface, 176
- JotBreak interface, 177
- JotCase interface, 177
- JotCatch interface, 177
- JotClass interface, 163
 - addUserData, 163
 - getComponentType, 163
 - getConstructor, 163
 - getConstructors, 164
 - getDeclaredConstructor, 164
 - getDeclaredConstructors, 164
 - getDeclaredField, 164
 - getDeclaredFields, 164
 - getDeclaredInnerClasses, 164
 - getDeclaredMethod, 164
 - getDeclaredMethods, 164
 - getField, 165
 - getFields, 165
 - getFile, 165
 - getInnerClasses, 165
 - getInterfaces, 165
 - getMethod, 165
 - getMethods, 165
 - getModifiers, 166
 - getName, 166
 - getSuperclass, 166
 - getType, 166
 - getUserData, 166
 - isArray, 166
 - isAssignableFrom, 166
 - isInstance, 166
 - isInterface, 166
 - isPrimitive, 166
 - newInstance, 167
- JotClassSource interface, 167
 - addConstructor, 167
 - addField, 167
 - addInitBlock, 167
 - addInnerClass, 167
 - addInterface, 168
 - addMethod, 168
 - addMethodDeclaration, 168
 - example, 186
 - getComparableLocation, 168
 - getDeclaredModifiers, 168
 - getDeclaringFile, 168
 - removeConstructor, 168
 - removeField, 168
 - removeInitBlock, 168
 - removeInnerClass, 168
 - removeInterface, 168
 - removeMethod, 168
 - setModifiers, 168
 - setName, 168
 - setSuperclass, 169
- JotCodeBlock interface, 171, 177
 - addAssignment, 171
 - addDoStatement, 171
 - addForStatement, 171
 - addIfStatement, 172
 - addInnerClass, 172
 - addMethodCall, 172

- addReturnStatement, 172
- addStatement, 172
- addTryStatement, 172
- addVariableDeclaration, 172
- addWhileStatement, 172
- getAssignments, 173
- getComparableLocation, 173
- getDeclaredInnerClasses, 173
- getMethodCall, 173
- getMethodCalls, 173
- getStatements, 173
- getVariableDeclaration, 173
- getVariableDeclarations, 173
- removeAssignment, 173
- removeInnerClass, 173
- removeMethodCall, 173
- removeStatement, 173
- removeVariableDeclaration, 173
- JotComment interface, 175
 - example, 187
 - getCommentText, 175
 - getNormalText, 175
 - getSummaryText, 175
 - getType, 175
- JotCommentable interface, 174
 - addBlankLine, 174
 - addComment, 175
 - getComment, 175
 - removeComment, 175
- JotCondition interface, 176
- JotConstructor interface, 170
- JotConstructorSource interface, 171
- JotContinue interface, 177
- JotDefault interface, 177
- JotDo interface, 177
- JotExpression interface, 176
 - getAssignment, 176
 - getCondition, 176
 - getMethodCall, 176
 - getNew, 176
 - getOperation, 176
 - getValue, 176
 - getVariable, 176
 - isConstant, 176
 - isNull, 176
- JotExpressionStatement interface, 178
- JotFieldDeclaration interface, 178
 - example, 187
- JotFile interface, 159
 - getClass, 159
 - getClasses, 159
 - getName, 159
 - getPackage, 159
 - getPackageManager, 159
 - getTimestamp, 159
 - getUrl, 159
- JotFileEvent class, 162
 - dispatch, 162
- getClass, 162
- getFile, 163
- getImport, 163
- JotFileListener interface, 162
 - fileClassChanged, 162
 - fileImportChanged, 162
 - fileMiscChanged, 162
 - filePackageChanged, 162
- JotFinally interface, 178
- JotFor interface, 178
- JotIf interface, 178
 - example, 188
- JotInitBlock interface, 174, 178
- JotInitializer interface, 176
- JotInnerClass interface, 167
- JotLabelled interface, 178
- JotMarker interface, 161
 - addUserData, 161
 - getEndPosition, 161
 - getStartPosition, 161
 - getUserData, 161
- JotMethod interface, 169
 - getDeclaringClass, 169
 - getFirstParameter, 169
 - getModifiers, 169
 - getName, 169
 - getParameter, 169
 - getParameters, 170
 - getParameterTypes, 170
 - getReturnType, 170
 - getThrowSpecifiers, 170
- JotMethodCall interface, 176
- JotMethodSource interface, 170
 - addParameter, 170
 - addThrowSpecifier, 170
 - example, 187
 - getCodeBlock, 170
 - getDeclaredModifiers, 171
 - getStatements, 171
 - removeParameter, 171
 - removeThrowSpecifier, 171
 - setModifiers, 171
 - setName, 171
 - setParameterText, 171
 - setReturnType, 171
- JotNew interface, 176
- JotPackages interface, 157
 - checkReread, 157
 - commit, 157
 - example, 185
 - file types, 159
 - getClass, 157
 - getEncoding, 157
 - getFile, 158
 - getFiles, 158
 - getFilesArray, 158
 - getPackage, 158
 - getPackages, 158
 - getPackagesArray, 158
 - getSourceFile, 158
 - getSourceVersion, 158
 - getUrl, 158
- loadClass, 158
- release, 158
- releaseAll, 159
- shutdown, 159
- JotReturn interface, 178
- JotSourceElement interface, 174
 - getIndentLevel, 174
 - getParent, 174
 - getText, 174
 - isModified, 174
 - setModified, 174
 - setText, 174
- JotSourceFile interface, 159
 - addClass, 160
 - addImport, 160
 - addJotFileListener, 160
 - example, 185
 - getComparableLocation, 160
 - getFullClassName, 160
 - getImport, 160
 - getImports, 160
 - isReadOnly, 161
 - out, 161
 - removeClass, 161
 - removeImport, 161
 - removeJotFileListener, 161
 - reRead, 161
 - setPackage, 161
 - setTimestamp, 161
- JotStatement interface, 177
 - getCodeBlock, 177
 - getStatements, 177
- JotSubscript interface, 176
- JotSwitch interface, 178
- JotSynchronized interface, 178
- JotThrow interface, 178
- JotTry interface, 178
 - example, 189
- JotType interface, 169
 - getFullName, 169
 - getJotClass, 169
 - getJotClassSource, 169
 - getName, 169
 - setName, 169
- JotTypeop interface, 177
- JotUnaryExpression interface, 177
- JotVariableDeclaration interface, 178
 - example, 188
- JotWhile interface, 178
 - example, 189
- JSortedTable class, 64
- JSP, 113, 179
- JspServletService class, 232
- JSPTagWizard example, 179

K

- KeyStrokeDialog class, 30
- getKeyStroke, 31

- setDisplayOptions, 31
- setKeyStroke, 31
- KeyStrokeEditorPanel class, 31
 - decodeKeyStroke, 32
 - encodeKeyStroke, 32
 - getJavaInitializationString, 32
 - getKeyName, 32
 - getKeyStroke, 32
 - getKeyStrokeName, 32
 - getKeyStrokeText, 32
 - getKeyText, 33
 - getResult, 33
 - getValueText, 33
 - getVKText, 33
 - resetFocus, 33
 - setDisplayOptions, 33
 - setKeyStroke, 33
 - stopEditing, 33
- KeyStrokeEditorTextField class, 33
 - getKeyStroke, 33
 - getShowEventType, 34
 - getShowKeyReleased, 34
 - getShowKeyTyped, 34
 - setDisplayOptions, 34
 - setKeyStroke, 34
 - setShowEventType, 34
 - setShowKeyReleased, 34
 - setShowKeyTyped, 34
- keystrokes
 - selecting, 30, 31, 33

L

- LAST_SELECTED_SETUP_NAME field
 - Setup, 268
- lastColumnSorted method
 - TableSorter, 63
- lastSortAscending method
 - TableSorter, 63
- launchWizard method
 - JBuilderInfo, 29
- layout assistants
 - registration, 131
 - resizing nibs, 139
- layout managers, 130
- LayoutAssistant interface, 131
 - cleanupRemovedComponent, 132
 - constraintEditorSelectionChanging, 132
 - editConstraints, 132
 - example, 144
 - getConstraintsType, 132
 - getPropertyEditor, 133
 - layoutChanged, 133
 - prepareActionGroup, 133
 - prepareAddComponent, 133
 - prepareAddStatus, 133

- prepareChangeLayout, 134
- prepareCloneComponent, 134
- prepareCloneStatus, 134
- prepareMouseMoveStatus, 134
- prepareMoveComponent, 135
- prepareMoveStatus, 135
- prepareResizeComponent, 135
- prepareResizeStatus, 135
- prepareSelectComponent, 135
- resizeAction, 136
- LayoutAssistant sample, 151
- layoutChanged method
 - LayoutAssistant, 133
- length method
 - ZipIndex, 70
- libraryExists method
 - Server, 244
- LINUX field
 - Platform, 49
- ListPanel class, 34
 - addChangeListener, 35
 - addListElement, 35
 - canAdd, 35
 - canEdit, 35
 - canMoveDown, 36
 - canMoveUp, 36
 - canRemove, 36
 - doubleClickElement, 36
 - editElement, 36
 - editSelectedListElement, 36
 - enableControls, 36
 - getAddButton, 36
 - getEditButton, 36
 - getElementName, 36
 - getList, 36
 - getListCellRendererComponent, 36
 - getListScrollPane, 36
 - getMoveDownButton, 36
 - getMoveUpButton, 36
 - getRemoveButton, 36
 - getSelectedIndex, 36
 - getSelectedIndices, 36
 - getSelectedListElement, 37
 - getSelectedListElements, 37
 - getSelectionModel, 37
 - moveSelectedListElement, 37
 - promptForElement, 37
 - removeChangeListener, 37
 - removeSelectedListElements, 37
 - selectValue, 37
 - setAddButtonVisible, 37
 - setEditButtonVisible, 37
 - setEnabled, 37
 - setList, 37
 - setMoveButtonsVisible, 37
- lists

- selecting, 34
- loadClass method
 - JotPackages, 158
- lookupHelp method
 - DesignerManager, 107
- ltrim method
 - Strings, 61

M

- MAC field
 - Platform, 49
- makeFullName method
 - Server, 244
- makeIntoArray method
 - Text, 65
- makeLocalBackup method
 - VCSUtils, 209
- makeServerLibraryName method
 - Server, 244
- makeServerToolName method
 - Server, 244
- Marcus Redeker, 225
 - bio, 225
- MessageService class, 232
- MessageView class
 - example, 259
- METHOD_SCOPE field
 - CmtSubcomponent, 91
- methodChanged method
 - CmtComponentListener, 88
- ModelNode class
 - example, 145
- modifyProjectLibraryList method
 - Server, 244
- move method
 - CmtModel, 96
 - example, 121
- moveSelectedListElement method
 - ListPanel, 37

N

- n2sort method
 - TableSorter, 64
- NamingService class, 232
- needToUpdate method
 - AbstractDescriptorConversion, 264
- NestingSetupPropertyPage class, 270
 - createSetupPanel, 271
 - getDescription, 271
- NEW_SERVER_WEIGHT field
 - Server, 247
- newInstance method
 - JotClass, 167
- newLauncher method

- Server, 244
- NO_DEPENDENCIES field
 - Service, 236
- NO_FEATURES field
 - Service, 236
- NO_MATCH field
 - RegularExpression, 55
- NO_NAME field
 - Server, 248
- nodes
 - binary files, 208
 - getting selected nodes, 208
- NodeViewerFactory interface
 - example, 292
- NONE_NAME field
 - ServerManager, 230
- notifyProjectTreeHasBeenRefreshed method
 - VCSUtils, 209
- notifyVCSSelected method
 - VCS, 201

O

- open method
 - Designer, 112
 - example, 117
 - ZipIndex, 70
- OpenTools-Designer, 131
 - example, 129, 150
- OpenTools-ServerServices, 230
- OpenTools-UI
 - example, 222
- OpenTools-Wizard
 - example, 191
- OS_NAME field
 - Platform, 49
- out method
 - JotSourceFile, 161

P

- PackageBrowserDialog class, 38
 - closeDialog, 39
 - getAllowPackages, 39
 - isAddToProjectEnabled, 39
 - isAddToProjectSelected, 39
 - isAddToProjectVisible, 39
 - isClassComboVisible, 39
 - setAddToProjectEnabled, 39
 - setAddToProjectSelected, 39
 - setAddToProjectVisible, 39
 - setAllowPackages, 39
 - setClassComboVisible, 39
 - showClassBrowserDialog, 40
 - showHelp, 40
 - showPackageBrowserDialog, 40
- PackageBrowserFilter interface, 40
 - packageFilter, 40

- PackageBrowserTree class, 41
 - browseClass, 41
 - browsePackageOrClass, 41
 - getFarthestPath, 42
 - getFullPath, 42
 - getMode, 42
 - getPackageBrowserFilter, 42
 - getPathNode, 42
 - getPathNodes, 42
 - getProject, 42
 - getSelectedNode, 42
 - getSelectedNodes, 42
 - getSelectedPath, 43
 - getSelectedPaths, 43
 - isClassNameInTree, 43
 - isPackageName, 43
 - setMode, 43
 - setPackageBrowserFilter, 43
 - setProject, 43
 - setSelectedPath, 43
- packageFilter method
 - PackageBrowserFilter, 40
- packages
 - for a project, 158
 - selecting, 38, 41
- PackageTree sample, 193
- parentLocation field
 - SelectNib, 138
- parsing Java, 155
- patchForAppServer method
 - AbstractDescriptor-Conversion, 264
- pathContainsClass method
 - Classes, 11
- pathFromString method
 - PathSet, 47
- pathFromArray method
 - PathSet, 47
- paths, 43
- PathSet class, 43
 - addEntries, 44
 - addUniquePath, 44
 - addUniquePaths, 44
 - addUniquePathsIfEnabled, 44
 - delete, 44
 - EMPTY_ARRAY, 48
 - getClassPath, 44
 - getCollection, 44
 - getCopy, 45
 - getDocPath, 45
 - getEmptyDescription, 45
 - getFullClassPath, 45
 - getFullDocPath, 45
 - getFullName, 45
 - getFullSourcePath, 45
 - getIcon, 45
 - getIncompleteDescription, 45
 - getLastModificationSaved, 45
 - getLastModified, 45
 - getLibKit, 45
 - getName, 46

- getPathSetReferenceClass, 46
- getProperties, 46
- getProperty, 46
- getReferenceName, 46
- getRequired, 46
- getRequiredNames, 46
- getResolver, 46
- getSourcePath, 46
- getUrl, 46
- isEmpty, 46
- isEnabled, 46
- isIncomplete, 47
- isReadOnly, 47
- pathFromString, 47
- pathFromArray, 47
- pathToString, 47
- pathToStringArray, 47
- resetFullPaths, 47
- save, 47
- setClassPath, 47
- setCollection, 48
- setDocPath, 48
- setFromCopy, 48
- setLibKit, 48
- setName, 48
- setProperty, 48
- setRequired, 48
- setSourcePath, 48
- setUrl, 48
- pathToString method
 - PathSet, 47
- pathToStringArray method
 - PathSet, 47
- pattern matching, 53
- performAction method
 - CommitAction, 212
- Platform class, 49
 - IBM_LINUX, 49
 - IBM_VENDOR, 49
 - isMacLAF, 49
 - isSpecialDown, 49
 - LINUX, 49
 - MAC, 49
 - OS_NAME, 49
 - SOLARIS, 49
 - STANDALONE_DDEDITOR, 49
 - UNIX, 49
 - WIN32, 49
- postProcessBuild method
 - AppServerTargeting, 257
- postStart method
 - ServerLauncher, 253
 - Service, 235
- postStop method
 - ServerLauncher, 253
 - Service, 235
- prefixMatch method
 - RegularExpression, 54
- prepareActionGroup method
 - example, 145
 - LayoutAssistant, 133

- prepareAddComponent method
 - example, 145
 - LayoutAssistant, 133
 - prepareAddStatus method
 - example, 146
 - LayoutAssistant, 133
 - prepareChangeLayout method
 - example, 146
 - LayoutAssistant, 134
 - prepareCloneComponent method
 - LayoutAssistant, 134
 - prepareCloneStatus method
 - LayoutAssistant, 134
 - prepareMouseMoveStatus method
 - example, 146
 - LayoutAssistant, 134
 - prepareMoveComponent method
 - LayoutAssistant, 135
 - prepareMoveStatus method
 - example, 147
 - LayoutAssistant, 135
 - prepareResizeComponent method
 - example, 147
 - LayoutAssistant, 135
 - prepareResizeStatus method
 - example, 147
 - LayoutAssistant, 135
 - prepareSelectComponent method
 - example, 148
 - LayoutAssistant, 135
 - preProcessBuild method
 - AppServerTargeting, 257
 - preServiceSelectionChanged-AndSaved method
 - Server, 244
 - preStart method
 - ServerLauncher, 253
 - Service, 235
 - preStartServices method
 - ServerLauncher, 253
 - preStop method
 - ServerLauncher, 253
 - Service, 235
 - print method
 - Debug, 17
 - println method
 - Debug, 17
 - printlnc method
 - Debug, 17
 - printProfiler method
 - Debug, 17
 - printStackTrace method
 - Debug, 17
 - Project Pane
 - refreshing, 206, 209
 - ProjectPathSet class, 49
 - addProjectLibrary, 50
 - getAuxPath, 50
 - getAuxPaths, 50
 - getBakPath, 50
 - getDefaultSourcePath, 50
 - getFullLibPath, 50
 - getIncludeTestPath, 50
 - getJDK, 50
 - getJDKPathSet, 50
 - getJDKs, 50
 - getLibKits, 51
 - getLibPath, 51
 - getLibraries, 51
 - getLibrary, 51
 - getName, 51
 - getOutPath, 51
 - getTime, 51
 - getProjectLibraries, 51
 - getResourcePath, 51
 - getTestPath, 51
 - getWorkingDirectory, 51
 - putClassOnFullPath, 52
 - reloadLibraries, 52
 - setAuxPath, 52
 - setBakPath, 52
 - setDefaultSourcePath, 52
 - setFullLibPath, 52
 - setIncludeTestPath, 52
 - setJDKPathSet, 52
 - setJDKs, 52
 - setLibPath, 52
 - setLibraries, 53
 - setOutPath, 53
 - setTestPath, 53
 - setWorkingDirectory, 53
 - projects
 - find packages, 158
 - projectUsesService method
 - Service, 235
 - promptForElement method
 - ListPanel, 37
 - property editors, 83
 - property pages
 - for application servers, 270
 - for VCS, 199
 - for VCS commit, 211
 - propertyChanged method
 - CmtComponentListener, 88
 - putClassOnFullPath method
 - ProjectPathSet, 52
 - putInBag method
 - ServerLauncher, 253
- ## Q
- queueException method
 - ServerLauncher, 253
- ## R
- read method
 - Streams, 58
 - ZipIndex, 70, 71
 - readChars method
 - Streams, 59
 - ReadingSource sample, 193
 - reallocateIndexes method
 - TableSorter, 64
 - redoLastColumnSort method
 - TableSorter, 64
 - refreshHistoryPane method
 - VCSUtils, 209
 - registerAssistant method
 - example, 144
 - UIDesigner, 131
 - registerComponentFactory method
 - CmtComponentManager, 83
 - registerCustomConfiguration-PageFactory method
 - ServerManager, 228
 - registerDesigner method
 - DesignerManager, 107
 - example, 115
 - registerFileOrDirNeededByVCS method
 - VCSUtils, 209
 - registerJdkSupportProvider method
 - ServerManager, 228
 - registerLegacyName method
 - ServerManager, 228
 - registerNodeViewerFactory method
 - example, 292
 - registerServer method
 - ServerManager, 229
 - registerService method
 - ServerManager, 229
 - registerServices method
 - Server, 245
 - registerServiceType method
 - ServerManager, 229
 - registerSetup method
 - example, 269
 - SetupManager, 266
 - registerSourceClass method
 - VCSUtils, 209
 - registerTargeting method
 - ServerManager, 229
 - registration
 - for application servers, 229
 - for component factories, 83
 - for designers, 107
 - for layout assistants, 131
 - for legacy servers, 228
 - for server configuration, 228
 - for server JDKs, 228
 - for server service types, 229
 - for server services, 229, 245
 - for server setups, 266
 - for server targeting, 229
 - for VCS, 199
 - for VCS required files, 209
 - for VCS source classes, 209
 - RegularExpression class, 53

- CHAR_ANY, 55
- CHAR_ESCAPE, 55
- CHAR_WILDCARD, 55
- exactMatch, 53
- findSubstringMatch, 54
- getLength, 54
- isBreakOnNewLine, 54
- isCaseSensitive, 54
- isPatternMatch, 54
- isRegExpMatch, 54
- isSpecialChar, 54
- NO_MATCH, 55
- prefixMatch, 54
- setBreakOnNewLine, 55
- setPatternMatch, 55
- setRegExpMatch, 55
- substringMatch, 55
- RegularExpression.MatchResult class, 55
- release method
 - CmtComponent, 85
 - CmtComponents, 82
 - CmtSubcomponent, 90
 - example, 186
 - JotPackages, 158
- releaseAll method
 - JotPackages, 159
- releaseLiveInstanc method
 - CmtSubcomponent, 91
- reloadLibraries method
 - ProjectPathSet, 52
- remove method
 - CmtModel, 97
 - example, 121
- removeActionListener method
 - ColorPanel, 14
- removeAllWhitespaceFrom method
 - Text, 65
- removeAssignment method
 - JotCodeBlock, 173
- removeBufferListener method
 - example, 125
- removeChangeListener method
 - ListPanel, 37
- removeClass method
 - JotSourceFile, 161
- removeComment method
 - JotCommentable, 175
- removeComponentSource-Listener method
 - CmtComponentSource, 87
- removeConstructor method
 - JotClassSource, 168
- removeDesignerListener method
 - DesignerManager, 108
- removeDesignerReleaseListener method
 - DesignerManager, 108
- removeField method
 - JotClassSource, 168
- removeFile method

- VCSUtils, 210
- removeFromIgnoreList method
 - VCS, 201
- removeIgnoreTeamFiles method
 - VCSUtils, 210
- removeImport method
 - JotSourceFile, 161
- removeInitBlock method
 - JotClassSource, 168
- removeInnerClass method
 - JotClassSource, 168
 - JotCodeBlock, 173
- removeInterface method
 - JotClassSource, 168
- removeJotFileListener method
 - JotSourceFile, 161
- removeMethod method
 - CmtComponentSource, 87
 - JotClassSource, 168
- removeMethodCall method
 - JotCodeBlock, 173
- removeModel method
 - CmtComponentSource, 87
- removeParameter method
 - JotMethodSource, 171
- removePersonalIgnoreFiles method
 - VCSUtils, 210
- removeProperty method
 - CmtComponentSource, 87
- removePropertyChangeListener method
 - CmtSubcomponent, 91
- removeSelectedListElements method
 - ListPanel, 37
- removeStatement method
 - JotCodeBlock, 173
- removeSubcomponent method
 - CmtComponentSource, 87
 - example, 121
- removeThrowSpecifier method
 - JotMethodSource, 171
- removeTraceCategory method
 - Debug, 17
- removeTrailingBlankLines method
 - Text, 65
- removeVariableDeclaration method
 - JotCodeBlock, 173
- renameSubcomponent method
 - CmtComponentSource, 87
- replaceTabs method
 - Text, 66
- reRead method
 - JotSourceFile, 161
- reset method
 - CmtPropertyState, 94
 - Diff, 22
- resetFocus method
 - KeyStrokeEditorPanel, 33

- resetFullPaths method
 - PathSet, 47
- resizeAction method
 - example, 148
 - LayoutAssistant, 136
 - SelectNib, 138
- restoreFrom method
 - ServerLauncher, 253
- reverseliterator method
 - Diff, 22
- RevisionInfo class, 202
 - BUFFER_REVISION, 203
 - FILE_REVISION, 203
 - getAuthor, 202
 - getComment, 202
 - getDate, 202
 - getLabel, 202
 - getLabels, 202
 - getRevisionNumber, 202
 - isUnderVCS, 202
 - isWorkingRevision, 202
 - setAuthor, 202
 - setComment, 202
 - setDate, 202
 - setLabel, 202
 - setLabels, 202
 - setRevisionNumber, 202
 - setVCSFlag, 202
 - setWorkingRevision, 202
- rtrim method
 - Strings, 61
- run configurations
 - updating for application servers, 247

S

- samples
 - Designer, 129
 - LayoutAssistant, 151
 - PackageTree, 193
 - ReadingSource, 193
 - samplevcs, 223
 - WritingSource, 193
- samplevcs sample, 223
- save method
 - example, 185
 - PathSet, 47
 - Server, 245
 - ServerManager, 229
- SearchTree class, 56
 - enableDragDrop, 57
 - getExpansionState, 57
 - getFarthestPath, 57
 - getNearestPath, 57
 - getPreserveMode, 57
 - getSelectionState, 57
 - getStringPath, 57
 - selectNearestPath, 57
 - setDefaultTooltipEnabled, 58
 - setExpansionState, 58

- setPreserveMode, 58
 - setSelectionMode, 58
- SelectBoxes interface, 137
 - example, 147
 - hide, 137
 - hideAll, 137
 - isVisible, 137
 - setContainer, 137
 - show, 137
- selectNearestPath method
 - SearchTree, 57
- SelectNib class, 137
 - example, 147, 148
 - getLayoutAssistant, 138
 - getNibBounds, 138
 - getRectangleDimension, 138
 - getRectangleLocation, 138
 - isSelectable, 138
 - parentLocation, 138
 - resizeAction, 138
 - setLayoutAssistant, 138
 - setRectangleDimension, 138
 - setRectangleLocation, 138
 - setSelectable, 138
 - target, 138
 - type, 139
 - use, 139
- selectValue method
 - ListPanel, 37
- serialize method
 - CmtSubcomponent, 91
- Server class, 236
 - addUniquePath, 237
 - attemptDefaultConfiguration, 237
 - changeDirectoryReferences-InString, 237
 - checkSetup, 237
 - clear, 237
 - clearProjectSettings, 237
 - createClientJar, 238
 - createClientLibrary, 238
 - createLibrariesFromSetup, 238
 - createLibrary, 238
 - DEFAULT_WEIGHT, 247
 - ensureNonNullValue, 238
 - ensureProjectContainsServer-ClientLibrary, 238
 - ensureProjectContainsServer-Library, 238
 - formatJarFileParameter, 239
 - getAssociatedJdk, 239
 - getClassPath, 239
 - getClientJarService, 239
 - getClientLibraryClassPath, 239
 - getClientLibraryName, 239
 - getCompanionNode, 239
 - getCopy, 239
 - getCustomConfigurationPageFactory, 239
 - getDefaultClassPath, 239
 - getDefaultHomeDirectory, 239
 - getDefaultName, 239
 - getDefaultServerName, 239
 - getDefaultVersion, 239
 - getDeployService, 240
 - getEjbService, 240
 - getExtraVisiBrokerTool-Parameters, 240
 - getFileNameBasedOn-Protocol, 240
 - getFullClassPath, 240
 - getFullLibraryClassPath, 240
 - getFullName, 240
 - getHomeDirectory, 240
 - getIncompleteDescription, 240
 - getJdkSupportProvider, 240
 - getJspServletService, 240
 - getLastModified, 240
 - getLegacyFullName, 240
 - getName, 241
 - getNameFromFullName, 241
 - getNewPathsBasedOnNew-HomeDirectory, 241
 - getNodeValue, 241
 - getOptimizerPackages, 241
 - getPackages, 241
 - getPathSet, 241
 - getServerTypeId, 241
 - getServices, 242
 - getSetupLauncher, 242
 - getShortName, 242
 - getShortNameWithVersion, 242
 - getUniqueRunDebugClass-Path, 242
 - getUrl, 242
 - getVersion, 242
 - getWeight, 242
 - hasClientJarCreator, 242
 - hasEjbDeployer, 242
 - hasSetup, 242
 - initialize, 242
 - INVALID_SERVER, 247
 - isCopy, 243
 - isGranular, 243
 - isIncomplete, 243
 - isInitiallySetup, 243
 - isPageValid, 243
 - isEnabled, 243
 - isSetup, 243
 - isSetupCompleted, 243
 - isValidSetupDirectory, 243
 - libraryExists, 244
 - makeFullName, 244
 - makeServerLibraryName, 244
 - makeServerToolName, 244
 - modifyProjectLibraryList, 244
- NEW_SERVER_WEIGHT, 247
- newLauncher, 244
- NO_NAME, 248
- preServiceSelectionChangedAndSaved, 244
- registerServices, 245
- save, 245
- serverModified, 245
- serverRelativePath, 245
- serviceSelectionChanged, 245
- serviceSelectionChangedAndSaved, 245
- setClassPath, 245
- setCustomConfiguration-PageFactory, 245
- setHomeDirectory, 245
- setJdkSupportProvider, 245
- setPathSet, 246
- setPathSetFromCopy, 246
- setServerEnabled, 246
- setSetupCompleted, 246
- setVersion, 246
- setWeight, 246
- supportsCopy, 246
- supportsCreateClientJar, 246
- supportsJavaRunnableEjb-Container, 246
- updateClassPathWithNew-HomeDirectory, 246
- updateLastModified, 247
- updateProjectClientSettings, 247
- updateProjectSettings, 247
- updateRunConfigurations, 247
- usesVisiBrokerOrb, 247
- WEIGHTED_COMPARATOR, 248
- SERVER_COMPARATOR
 - field
 - ServerManager, 230
- ServerLauncher class, 248
 - addSourceBridge, 248
 - appendHttpPort, 248
 - canStop, 248
 - clearExceptionQueue, 249
 - clearSourceBridges, 249
 - configureLauncher, 249
 - configureServices, 249
 - customizeArguments, 249
 - customizeClassPath, 249
 - customizeLibraries, 249
 - customizeTransportAddress, 249
 - customizeVmParameters, 250
 - deployLibraries, 250
 - deployLibrary, 250
 - deployLibraryEntry, 250
 - escapeParameter, 250

- getArchivesToDeployOnRun, 250
- getArguments, 250
- getCommand, 250
- getCurrentWorkingDirectory, 250
- getDefaultArguments, 250
- getDefaultNecessaryArguments, 250
- getDefaultNecessaryVmParameters, 250
- getDefaultVmParameters, 250
- getDefaultWorkingDirectory, 250
- getEnvironment, 251
- getEnvpWithPathVariablePrefix, 251
- getFromBag, 251
- getJavaLauncher, 251
- getLabel, 251
- getLibraryClassesRelativePath, 251
- getLibraryDestination, 251
- getNecessaryArguments, 251
- getNecessaryVmParameters, 251
- getProject, 251
- getProjectLibrariesForRun, 251
- getPropertyMap, 251
- getQueuedExceptions, 251
- getRunConfigPropertyPages, 251
- getServer, 252
- getService, 252
- getServices, 252
- getShutdownWaitTime, 252
- getSourceBridge, 252
- getSourceBridges, 252
- getStopper, 252
- getTracker, 252
- getVmParameters, 252
- getWaitForServerThread, 252
- getWorkingDirectory, 252
- getWorkingDirectoryFromHomeDirectory, 252
- init, 252
- initLauncher, 252
- isValidWorkingDirectory, 253
- postStart, 253
- postStop, 253
- preStart, 253
- preStartServices, 253
- preStop, 253
- putInBag, 253
- queueException, 253
- restoreFrom, 253
- setArguments, 253
- setNecessaryArguments, 253
- setNecessaryVmParameters, 253
- setVmParameters, 254
- setWorkingDirectory, 254
- standardRunConfigPropertyPages, 254
- stop, 254
- supportsClearDeployedArchivesBeforeRun, 254
- trackerClosed, 254
- updateLastModified, 254
- useVmParameters, 254
- validateServices, 254
- ServerManager class, 225
 - findServerPathSet, 226
 - findService, 226
 - getAvailableTypes, 226
 - getEnabledServers, 226
 - getLastRegisteredServer, 226
 - getNoneServerItem, 226
 - getPrimaryServer, 227
 - getServer, 227
 - getServers, 227
 - getService, 227
 - getServices, 227
 - getServiceType, 227
 - getServiceTypes, 227
 - NONE_NAME, 230
 - registerCustomConfigurationPageFactory, 228
 - registerJdkSupportProvider, 228
 - registerLegacyName, 228
 - registerServer, 229
 - registerService, 229
 - registerServiceType, 229
 - registerTargeting, 229
 - save, 229
 - SERVER_COMPARATOR, 230
 - SERVICE_COMPARATOR, 230
 - SERVICE_TYPE_COMPARATOR, 230
 - serviceAvailable, 229
 - serviceSupported, 230
- serverModified method
 - Server, 245
- serverRelativePath method
 - Server, 245
- servers. *See* application
 - servers:servers
- Service class, 232
 - buildFeatureSet, 232
 - configureLauncher, 232
 - getAllAvailableFeatures, 232
 - getAllAvailableSpecFeatures, 232
 - getAssociatedModuleType, 232
- getAvailableSpecFeaturesForAssociatedModuleType, 233
- getClientVmParameters, 233
- getCompanionNode, 233
- getCustomizedRunDebugClassPath, 233
- getDefaultSupportedSpecFeatures, 233
- getDependencies, 233
- getFeatures, 233
- getProjectPropertiesPage, 234
- getRunConfigPropertyPage, 234
- getServer, 235
- getService, 235
- getServices, 235
- getServiceType, 235
- getServiceTypeKeys, 235
- isGranular, 235
- NO_DEPENDENCIES, 236
- NO_FEATURES, 236
- postStart, 235
- postStop, 235
- preStart, 235
- preStop, 235
- projectUsesService, 235
- supportsFeature, 236
- validate, 236
- service types. *See* application
 - servers:service types
- Service.Type class, 230
 - checkChildNodes, 231
 - getFeatureDefinition, 231
 - getIcon, 231
 - getName, 231
 - getPropertyKey, 231
 - getSkuVersion, 231
 - isDefaultEnabled, 231
 - isEnabledForSku, 231
 - isRuntime, 232
- SERVICE_COMPARATOR field
 - ServerManager, 230
- SERVICE_TYPE_COMPARATOR field
 - ServerManager, 230
- serviceAvailable method
 - ServerManager, 229
- services. *See* application
 - servers:services
- serviceSelectionChanged method
 - Server, 245
- serviceSelectionChangedAndSaved method
 - Server, 245
- serviceSupported method
 - ServerManager, 230
- SessionService class, 232
- setActionCommand method

- ColorPanel, 14
- setActiveNode method
 - example, 183
- setAddButtonVisible method
 - ListPanel, 37
- setAddToProjectEnabled method
 - PackageBrowserDialog, 39
- setAddToProjectSelected method
 - PackageBrowserDialog, 39
- setAddToProjectVisible method
 - PackageBrowserDialog, 39
- setAffectChildrenEnabled method
 - CheckTreeNode, 9
- setAffectedByParent method
 - CheckTreeNode, 9
- setAllowPackages method
 - PackageBrowserDialog, 39
- setAlphaThreshold method
 - AuraImage, 6
- setArguments method
 - ServerLauncher, 253
- setAssignment method
 - CmtSubcomponent, 91
- setAuraRGB method
 - AuraImage, 6
- setAuthor method
 - RevisionInfo, 202
- setAutoCenter method
 - DefaultDialog, 20
- setAuxPath method
 - ProjectPathSet, 52
- setBackground method
 - example, 148
- setBakPath method
 - ProjectPathSet, 52
- setBoundsAsString method
 - DefaultDialog, 20
- setBreakOnNewLine method
 - RegularExpression, 55
- setBytes method
 - AbstractDeployment-Descriptor, 260
- setCancelButton method
 - DefaultDialog, 20
- setChecked method
 - CheckTreeNode, 9
- setClassComboVisible method
 - PackageBrowserDialog, 39
- setClassPath method
 - PathSet, 47
 - Server, 245
- setCollection method
 - PathSet, 48
- setComment method
 - RevisionInfo, 202
 - VCSFileInfo, 204
- setContainer method
 - SelectBoxes, 137
- setContent method
 - example, 127
- setContent method
 - TextFile, 66
- setCustomColor method
 - ColorPanel, 14
- setCustomColors method
 - ColorCombo, 13
 - ColorPanel, 15
- setCustomConfigurationPage-Factory method
 - Server, 245
- setCustomizerDialog method
 - CmtSubcomponent, 91
- setDate method
 - RevisionInfo, 202
- setDefaultButton method
 - DefaultDialog, 20
- setDefaultSourcePath method
 - ProjectPathSet, 52
- setDefaultTooltipEnabled method
 - SearchTree, 58
- setDefaultValue method
 - CmtPropertyState, 94
- setDisplayOptions method
 - KeyStrokeDialog, 31
 - KeyStrokeEditorPanel, 33
 - KeyStrokeEditorTextField, 34
- setDocPat method
 - PathSet, 48
- setEditButtonVisible method
 - ListPanel, 37
- setEjbJarDirty method
 - AbstractDescriptor-Conversion, 264
- setEnabled method
 - CheckTreeNode, 9
 - example, 181
 - ListPanel, 37
- setEnabledAffectedByParent method
 - CheckTreeNode, 10
- setExpandedIcon method
 - CheckTreeNode, 10
- setExpansionState method
 - SearchTree, 58
- setExtraLocation method
 - AbstractDeployment-Descriptor, 260
- setFileAccessor method
 - AbstractDeployment-Descriptor, 260
- setFromCopy method
 - PathSet, 48
- setFullLibPath method
 - ProjectPathSet, 52
- setHelpButton method
 - DefaultDialog, 20
- setHomeDirectory method
 - Server, 245
- setIcon method
 - CheckTreeNode, 10
- setIncludeTestPath method
 - ProjectPathSet, 52
- setInitializer method
 - CmtSubcomponent, 91
- example, 187, 188
- setJDKPathSet method
 - ProjectPathSet, 52
- setJDKs method
 - ProjectPathSet, 52
- setJdkSupportProvider method
 - Server, 245
- setKeyStroke method
 - KeyStrokeDialog, 31
 - KeyStrokeEditorPanel, 33
 - KeyStrokeEditorTextField, 34
- setLabel method
 - RevisionInfo, 202
- setLabels method
 - RevisionInfo, 202
- setLastDesignedNode method
 - CmtComponentSource, 87
- setLayoutAssistant method
 - example, 148
 - SelectNib, 138
- setLibKit method
 - PathSet, 48
- setLibPath method
 - ProjectPathSet, 52
- setLibraries method
 - ProjectPathSet, 53
- setList method
 - ListPanel, 37
- setLiveClass method
 - CmtSubcomponent, 91
- setLiveInstance method
 - CmtSubcomponent, 91
- setLocked method
 - CheckTreeNode, 10
- setLogStream method
 - Debug, 17
- setMode method
 - PackageBrowserTree, 43
- setModel method
 - DesignView, 137
 - TableSorter, 64
- setModified method
 - JotSourceElement, 174
- setModifiers method
 - example, 186, 187
 - JotClassSource, 168
 - JotMethodSource, 171
- setModuleDirty method
 - AbstractDescriptor-Conversion, 264
- setMoveButtonsVisible method
 - ListPanel, 37
- setName method
 - AbstractDeployment-Descriptor, 260
 - CmtPropertySource, 93
 - JotClassSource, 168

- JotMethodSource, 171
- JotType, 169
- PathSet, 48
- setNecessaryArguments method
 - ServerLauncher, 253
- setNecessaryVmParameters method
 - ServerLauncher, 253
- setNeedsSerialize method
 - CmtSubcomponent, 91
- setOrientation method
 - ButtonStrip, 7
- setOutPath method
 - ProjectPathSet, 53
- setPackage method
 - example, 185
 - JotSourceFile, 161
- setPackageBrowserFilter method
 - PackageBrowserTree, 43
- setPanelGridHeight method
 - ColorPanel, 15
- setParameterText method
 - JotMethodSource, 171
- setParent method
 - example, 182
- setPathSet method
 - Server, 246
- setPathSetFromCopy method
 - Server, 246
- setPatternMatch method
 - RegularExpression, 55
- setPopupAlignment method
 - ColorCombo, 13
- setPopupGridHeight method
 - ColorCombo, 13
- setPreserveMode method
 - SearchTree, 58
- setProject method
 - PackageBrowserTree, 43
- setProperty method
 - PathSet, 48
- setReadable method
 - CmtPropertySource, 93
- setRectangleDimension method
 - SelectNib, 138
- setRectangleLocation method
 - SelectNib, 138
- setRegExpMatch method
 - RegularExpression, 55
- setReopenWarningMessage method
 - SetupManager, 266
- setRequired method
 - PathSet, 48
- setRestartWarningMessage method
 - SetupManager, 266
- setReturnType method
 - JotMethodSource, 171
- setRevisionNumber method
 - RevisionInfo, 202
- setRunnerListener method
 - CommitAction, 212
- setScope method
 - CmtSubcomponent, 91
- setSelectable method
 - example, 148
 - SelectNib, 138
- setSelectedColor method
 - ColorCombo, 13
 - ColorPanel, 15
- setSelectedPath method
 - PackageBrowserTree, 43
- setSelectionState method
 - SearchTree, 58
- setSelectTopAfterSort method
 - TableSorter, 64
- setServerEnabled method
 - Server, 246
- setSetupCompleted method
 - Server, 246
- setShowDescription method
 - Setup, 268
- setShowEventType method
 - KeyStrokeEditorTextField, 34
- setShowKeyReleased method
 - KeyStrokeEditorTextField, 34
- setShowKeyTyped method
 - KeyStrokeEditorTextField, 34
- setShowReopenWarning method
 - SetupManager, 266
- setShowRestartWarning method
 - SetupManager, 266
- setSourcePath method
 - PathSet, 48
- setStatus method
 - VCSFileInfo, 204
 - VCSFileStatus, 205
- setStatusMsg method
 - JBuilderInfo, 29
- setStatusText method
 - VCSUtils, 210
- setSuperclass method
 - example, 186
 - JotClassSource, 169
- setTestPath method
 - ProjectPathSet, 53
- setText method
 - CheckTreeNode, 10
 - JotSourceElement, 174
- setTexture method
 - TexturePanel, 67
- setTimestamp method
 - AbstractDeployment-Descriptor, 260
 - JotSourceFile, 161
- setType method
 - CmtPropertySource, 93
- Setup class, 267
 - CATEGORY, 268
 - createSetupPage, 267
 - example, 269
 - getName, 267
 - getPageFactory, 267
 - getPersonalities, 267
 - getSetupPropertyPage, 267
 - initializeSetupPage, 267
 - isEnabled, 268
 - isShowDescription, 268
 - LAST_SELECTED_SETUP_NAME field, 268
 - setShowDescription, 268
- SetupManager class, 265
 - checkShowRestartWarning, 265
 - example, 269
 - getSetup, 265
 - getSetupPersonalities, 265
 - getSetups, 265
 - isShowReopenWarning, 266
 - isShowRestartWarning, 266
 - registerSetup, 266
 - setReopenWarningMessage, 266
 - setRestartWarningMessage, 266
 - setShowReopenWarning, 266
 - setShowRestartWarning, 266
- SetupPage interface, 269
 - createSetupPanel, 270
 - getDescription, 270
- SetupPropertyPage class, 270
 - createSetupPanel, 270
 - getBrowser, 270
 - getDescription, 270
 - getProject, 270
 - isModified, 270
- setUrl method
 - PathSet, 48
 - VCSFileInfo, 204
- setValue method
 - CmtPropertyState, 94
 - example, 182
- setValueAt method
 - TableSorter, 64
- setValueSource method
 - CmtPropertySetting, 95
 - CmtPropertyState, 94
 - example, 146
- setValueText method
 - CmtPropertyState, 95
- setVCSFlag method
 - RevisionInfo, 202
- setVersion method
 - Server, 246
- setVmParameters method
 - ServerLauncher, 254
- setWeight method
 - Server, 246
- setWizardTitle method
 - example, 181
- setWorkingDirectory method
 - ProjectPathSet, 53

- ServerLauncher, 254
 - setWorkingRevision method
 - RevisionInfo, 202
 - setWritable method
 - CmtPropertySource, 93
 - show method
 - DefaultDialog, 20
 - example, 147, 148
 - SelectBoxes, 137
 - ZipIndex, 71
 - showAll method
 - ZipIndex, 71
 - showClassBrowserDialog method
 - PackageBrowserDialog, 40
 - showHelp method
 - example, 183
 - PackageBrowserDialog, 40
 - showInfoDialog method
 - JBuilderInfo, 29
 - showModalDialog method
 - DefaultDialog, 20
 - showPackageBrowserDialog method
 - PackageBrowserDialog, 40
 - showProjectStatus method
 - VCSUtils, 210
 - showSimpleModalDialog method
 - DefaultDialog, 20
 - showSimpleNonModalDialog method
 - DefaultDialog, 20
 - showVcsConfigurationDialog method
 - VCSUtils, 211
 - shutdown method
 - CmtComponents, 82
 - JotPackages, 159
 - shuttlesort method
 - TableSorter, 64
 - size method
 - Diff, 23
 - SOLARIS field
 - Platform, 49
 - sort method
 - TableSorter, 64
 - sortByColumn method
 - TableSorter, 64
 - SourceSafeVCS example, 212
 - STANDALONE_DDEDITOR field
 - Platform, 49
 - STANDARD_ENCODING field
 - Strings, 62
 - STANDARD_ENCODING_DESCRIPTION field
 - Strings, 62
 - standardRunConfigPropertyPage
 - s method
 - ServerLauncher, 254
 - startProfiler method
 - Debug, 17
 - startsWithIgnoreCase method
 - Strings, 61
 - stop method
 - ServerLauncher, 254
 - stopEditing method
 - KeyStrokeEditorPanel, 33
 - stopProfiler method
 - Debug, 18
 - Streams class, 58
 - copy, 58
 - read, 58
 - readChars, 59
 - strings
 - escaping characters, 62
 - Strings class, 59
 - canConvertToInteger, 59
 - capitalize, 59
 - convertLineEndings, 59
 - convertToInteger, 59
 - convertToPlatformLineEndings, 60
 - convertToUnixLineEndings, 60
 - decapitalize, 60
 - decode, 60
 - decodeArray, 60
 - EMPTY_ARRAY, 61
 - encode, 60
 - encodeArray, 60
 - format, 61
 - getSubStrings, 61
 - isEmpty, 61
 - ltrim, 61
 - rtrim, 61
 - STANDARD_ENCODING, 62
 - STANDARD_ENCODING_DESCRIPTION, 62
 - startsWithIgnoreCase, 61
 - Strings.StringEncoding class, 62
 - decode, 62
 - encode, 62
 - subcomponentChanged method
 - CmtComponentListener, 88
 - substringMatch method
 - RegularExpression, 55
 - supportsClearDeployedArchivesBeforeRun method
 - ServerLauncher, 254
 - supportsCopy method
 - Server, 246
 - supportsCreateClientJar method
 - Server, 246
 - supportsFeature method
 - Service, 236
 - supportsJavaRunnableEjbContainer method
 - Server, 246
 - swap method
 - TableSorter, 64
- ## T
- tableChanged method
 - TableSorter, 64
 - tables
 - sorted, 62
 - TableSorter class, 62
 - addMouseListenerToHeaderInTable, 63
 - checkModel, 63
 - compare, 63
 - compareRowsByColumn, 63
 - getSortedRowIndex, 63
 - getUnsortedRowIndex, 63
 - getValueAt, 63
 - lastColumnSorted, 63
 - lastSortAscending, 63
 - n2sort, 64
 - reallocateIndexes, 64
 - redoLastColumnSort, 64
 - setModel, 64
 - setSelectTopAfterSort, 64
 - setValueAt, 64
 - shuttlesort, 64
 - sort, 64
 - sortByColumn, 64
 - swap, 64
 - tableChanged, 64
 - target field
 - example, 148
 - SelectNib, 138
 - Text class, 65
 - getHeadingSpace, 65
 - getIndentColumn, 65
 - makeIntoArray, 65
 - removeAllWhitespaceFrom, 65
 - removeTrailingBlankLines, 65
 - replaceTabs, 66
 - TextFile class, 66
 - getContents, 66
 - setContents, 66
 - TexturePanel class, 66
 - getTexture, 67
 - setTexture, 67
 - todo template, 191
 - toEditScript method
 - Diff, 23
 - toPath method
 - Classes, 11
 - toString method
 - AbstractDeploymentDescriptor, 260
 - CheckTreeNode, 10
 - trace method
 - Debug, 18
 - trackerClosed method
 - ServerLauncher, 254
 - TransactionService class, 232
 - trees
 - searching in, 56

- with checkboxes, 7
- triggerPropertyChange method
 - CmtPropertyState, 95
- type field
 - example, 148
 - SelectNib, 139

U

- UIDesigner class, 131
 - example, 144
 - registerAssistant, 131
- UISampler example, 72
- UNIX field
 - Platform, 49
- update method
 - example, 181
- updateBuildTask method
 - AppServerTargeting, 257
- updateClassPathWithNewHome
 - Directory method
 - Server, 246
- updateDeploymentDescriptors
 - method
 - AppServerTargeting, 257
- updateEjbModule method
 - AbstractDescriptor-
 - Conversion, 264
- updateLastModified method
 - Server, 247
 - ServerLauncher, 254
- updateProjectClientSettings
 - method
 - Server, 247
- updateProjectSettings method
 - Server, 247
- updateRunConfigurations
 - method
 - Server, 247
- updateVerifyReport method
 - AppServerTargeting, 258
- Url class
 - backup Urls, 207
 - example, 181
 - for a class, 158
 - for a JOT file, 159
 - make backup, 209
- use field
 - SelectNib, 139
- usesVisiBrokerOrb method
 - Server, 247
- useVmParameters method
 - ServerLauncher, 254
- utilities
 - for streams, 58
 - for strings, 59, 65
 - for text files, 66
 - images, 24, 25

V

- VA_INIT_METHOD_NAME
 - field
 - CmtComponentSource, 87
- validate method
 - Service, 236
- validateDialog method
 - DialogValidator, 21
- validateServices method
 - ServerLauncher, 254
- VCS, 197. *See* VCS class
 - binary files, 201, 208, 209
 - comments, 205
 - committing, 211
 - configuration, 211
 - current VCS, 207
 - deleting files, 210
 - file status, 204
 - icons, 201, 204
 - ignoring files, 199, 201, 205, 206, 210
 - labels, 202
 - make backup, 209
 - menus, 200, 201
 - project status, 200, 210
 - property page, 199
 - registration, 199
 - required files, 207, 208, 209
 - retrieve project, 199
 - retrieve source, 200
 - revision history, 200
 - revision numbers, 202, 203
 - revisions, 202
 - utilities, 205
 - working revision, 202
- VCS class, 199. *See* VCS
 - addToIgnoreList, 199
 - example, 213
 - getDescription, 199
 - getMergeConflictDivider-
 - Marker, 199
 - getMergeConflictEndMarker, 199
 - getMergeStartDividerMarker, 199
 - getName, 199
 - getNewProjectFromVCS-
 - WizardAction, 199
 - getProjectConfigPage, 199
 - getProjectConfigPageNew, 199
 - getProjectStatus, 200
 - getRefactorCheckoutAction, 200
 - getRevisions, 200
 - getSource, 200
 - getVCSContextMenuGroup, 200
 - getVCSFileMenuGroup, 200
 - getVCSGlobalMenuGroup, 201

- getVCSIcon, 201
- getVCSProjectMenuGroup, 201
- isBinary, 201
- isConfigureVCSMenu-
 - Enabled, 201
- isSelectVCSMenuEnabled, 201
- isUnderVCS, 201
- notifyVCSSelected, 201
- removeFromIgnoreList, 201
- VCSFactory class, 198
 - addVCS, 199
 - example, 214
 - getNames, 199
 - getVCS, 199
- VCSFileInfo class, 203
 - getComment, 204
 - getFile, 204
 - getName, 204
 - getStatus, 204
 - getUrl, 204
 - setComment, 204
 - setStatus, 204
 - setUrl, 204
- VCSFileStatus class, 204
 - getActions, 204
 - getStatus, 204
 - getStatusIcon, 204
 - getVCSFileActions, 205
 - isCommentRequired, 205
 - isModifiedInVCS, 205
 - isModifiedLocally, 205
 - isNew, 205
 - setStatus, 205
- VCSUtils class, 205
 - addPersonalIgnoreFiles, 205
 - addTeamIgnoreFiles, 206
 - checkProjectLocal, 206
 - createBackupAndOutputDirs, 206
 - doesProjectTreeNeed-
 - Refreshed, 206
 - fixConflictForEjbGrpXml-
 - Source, 206
 - fixConflictsForJavaSource, 206
 - getActiveVCS, 207
 - getActiveVCSName, 207
 - getBackupUrl, 207
 - getBrowserActiveNode, 207
 - getExcludedPaths, 207
 - getFilesNeededByVCS, 207
 - getLocalRevisions, 207
 - getPathRelativeToProject-
 - Directory, 207
 - getPersonalExcludedPaths, 208
 - getRelativePath, 208
 - getSelectedNodesInProject-
 - Pane, 208
 - handleOldStyleProjects, 208

- isBinaryFileNode, 208
- isFileType, 208
- isInProjectDirectory, 208
- isVCSFileOrDir, 208
- makeLocalBackup, 209
- notifyProjectTreeHasBeenRe-
freshed, 209
- refreshHistoryPane, 209
- registerFileOrDirNeededByV-
CS, 209
- registerSourceClass, 209
- removeFile, 210
- removePersonalIgnoreFiles,
210
- removeTeamIgnoreFiles, 210
- setStatusText, 210
- showProjectStatus, 210
- showVcsConfigurationDialog
, 211
- verifyDeploymentDescriptors
method
 - AbstractDescriptor-
Conversion, 264
 - AppServerTargeting, 258
 - example, 258, 259
- version control system. *See* VCS
- VetoException class, 67
- VFS, 128
- Virtual File System. *See* VFS
- VisiBroker, 247
- Visual SourceSafe, 212

W

- waitForImage method
 - Images, 26
- warn method
 - Debug, 18
- wasCommitSuccessful method
 - CommitAction, 212
- Web application, 113
- WebAppPersonality class
 - example, 181
- WEIGHTED_COMPARATOR
field
 - Server, 248
- WIN32 field
 - Platform, 49
- Wizard interface
 - example, 180
- WizardAction class
 - example, 180
 - for VCS, 199
- WritingSource sample, 193

X

- XMLValidator example, 287
- XSLTNodeViewer example, 294
- XSLTViewerFactory example,
290

Z

- Zip files, 67

- ZipHelpTopic class
 - example, 183
- ZipIndex class, 67
 - close, 68
 - contains, 69
 - getAllChildren, 69
 - getChildren, 69
 - getExistingZipFile, 69
 - getFilenames, 69
 - getLastModified, 69
 - getOpenZipIndexes, 69
 - getRawLastModified, 69
 - getZipEntries, 70
 - getZipIndex, 69
 - getZipIndexEntry, 70
 - hide, 70
 - hideAll, 70
 - isDirectory, 70
 - isHidden, 70
 - isOpen, 70
 - length, 70
 - open, 70
 - read, 70, 71
 - show, 71
 - showAll, 71
- ZipIndexEntry class, 71
 - getDirectory, 71
 - getLastModified, 71
 - getName, 71
 - getRawLastModified, 71
 - isDirectory, 72