



Visaj for SNIFF+J

The Visual
Application Builder
for Java™

Release 2

User's Guide





Imperial Software Technology

Berkshire House
252 Kings Road
Reading
Berkshire RG1 4HP
Tel: +44 118 958 7055
Fax: +44 118 958 9005

120 Hawthorne Avenue
Suite 101
Palo Alto
CA 94301
Tel: +1 650 688 0200
Fax: +1 650 688 1054

email: sales@ist.co.uk
support@ist.co.uk
URL: <http://www.ist.co.uk>

Trademarks and Copyrights

Visaj and the Visaj logo are trademarks of Pacific Imperial, Inc.
SwingBridge is a trademark of Pacific Imperial, Inc.
IST and the IST logo are trademarks of Imperial Software Technology Limited.
Java, JavaBeans and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc.
All other trademarks are acknowledged as the property of their respective owners.

Copyright © 1997, 1998, 1999 by Pacific Imperial, Inc.
All Rights Reserved. This manual is subject to copyright protection.
No portion may be copied without prior written consent from Imperial Software Technology Limited or Pacific Imperial, Inc.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

Visaj contains small amounts of software copyrighted by Jef Poskanzer or James R. Weeks respectively. This software is provided by the author and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Visaj Release 2.4
Reference: VJ/5 Issue 9.0 October 1999

Copyright © 1992–1999 TakeFive Software Inc.

All rights reserved. TakeFive products contain trade secrets and confidential and proprietary information of TakeFive Software Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure.

Parts of SNIFF+

Copyright 1991, 1992, 1993, 1994 by Stichting Mathematisch Centrum,
Amsterdam, The Netherlands.

Portions copyright 1991-1997 Compuware Corporation.

Trademarks

SNIFF+ is a trademark of TakeFive Software Inc.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Credits

The first version of Sniff was developed at the Informatics Laboratory of the Union Bank of Switzerland. Its development was considerably facilitated by the public domain application framework ET++.

Authors of the first version:

Walter Bischofberger (Sniff)

Erich Gamma (Sniffgdb)

Erich Gamma and André Weinand (ET++)

SNIFF+ Release 3.2

Contents

1. Overview	1
1.1 Introduction	1
1.2 Visaj	2
1.3 How To Use This Document	4
1.4 Online Help	5
1.5 Online User Guide.....	7
1.6 Prerequisites.....	7
2. Integrating SNIFF+J with Visaj	9
2.1 Introduction	9
2.2 Installation	9
2.3 Adding Visaj projects to a SNIFF+ project	10
2.4 Working with SNIFF+ in Visaj	11
2.5 Tools not supported in the SNIFF+ Visaj integration	12
3. Visaj Tutorial	13
3.1 The Tutorial	13
4. The Class Editor	37
4.1 Introduction	37

4.2	Class Structure View	38
4.3	“this”	42
4.4	Method Editing	47
4.5	Importing X-Designer Save Files	50
4.6	Applets	51
4.7	Generating Code	51
4.8	Windows Menu	52
4.9	Displaying Other Tools	53
4.10	Option Menu Items	54
5.	Beans View	55
5.1	Dynamic Display	57
5.2	Dummy Frames	58
5.3	Building Hierarchies	58
5.4	Object Palette	59
5.5	Properties	60
5.6	Customizers	68
5.7	Layout Editors	69
5.8	Using and Applying Layouts	74
5.9	Invisible Beans	74
5.10	Using Your Beans - Creating Reusable Components	76
6.	Event Bindings	79
6.1	Introduction	79
6.2	Event Binding Editor	80
6.3	Event Binding Editor: Parameters	83
6.4	Event Bindings List	86
6.5	Invalid Bindings	87

7.	Swing Component Set	89
7.1	Introduction	89
7.2	AWT to Swing Conversion	90
7.3	Loading Swing Components	90
7.4	Using Swing	93
7.5	Adding Swing Components to a Design	93
7.6	Highlighting of Non-Opaque Components	95
7.7	Swing Tips	95
8.	Image Editor	107
8.1	Description	107
8.2	Tutorial	108
8.3	Image Files	116
8.4	Help	118
8.5	Tool Palette	118
8.6	Colors	121
8.7	Selection	123
8.8	Gradients	125
8.9	Editing the Image	129
8.10	Filters	132
9.	Resource Bundle Editor	139
9.1	File Menu	140
9.2	Edit Menu	141
9.3	Generating Code	143
9.4	Using Resource Bundles	144
10.	The Project Window	147
10.1	Introduction	147

10.2	Creating, Saving and Opening Projects	148
10.3	Adding, Removing and Renaming Groups.	148
10.4	Adding and Removing Files	149
10.5	Creating Files	149
10.6	Editing Files.	149
10.7	The Windows Menu	150
11.	Generated Code	151
11.1	Introduction.	151
11.2	How to Generate Code.	151
11.3	What is Generated	152
11.4	Adding Your Own Code - Subclassing	153
11.5	Editing the Code.	154
11.6	Regenerating Code - Using the Update Toggle.	154
11.7	Example Code.	155
11.8	Using the Diamond Components.	159
11.9	File Types on Apple Macintoshes.	159
12.	Configuration	161
12.1	Integration with an IDE.	161
12.2	Palette File	162
12.3	Pre-defined Palette Files	166
12.4	Merging Palette Files	166
12.5	Use Swing Palette	166
12.6	Visaj Options	166
13.	Tips And Hints	171
13.1	Introduction.	171
13.2	User Interface	171
13.3	Event Bindings	178

13.4	Loading X-Designer Save Files	180
13.5	Palette Configuration	181
13.6	Layout	183
13.7	Generated Code	185
A.	Diamond Components	191
A.1	The Diamond Components	191
A.2	Using the Diamond Components	197
B.	Quick Reference	199
B.1	Class Editor Toolbar Buttons	199
B.2	Project Window Toolbar Buttons	200
B.3	Resource Bundle Editor Toolbar Buttons	200
B.4	Class Editor Menu Items	200
B.5	Project Window Menu Items	204
B.6	Resource Bundle Editor Menu Items	205
C.	Bibliography	209
C.1	Introduction	209
C.2	Books on Java	209
C.3	Books on Internationalization	209
C.4	Books on HTML	210
D.	Glossary	211
	Index	219

1.1 Introduction

This guide describes Visaj. Each area of Visaj is described in its own chapter or section and clarified with examples where needed. The major areas are:

1. The Class Editor - Visaj's editor for creating and editing classes (page 37)
2. Beans View - a description of Visaj's powerful Beans builder (page 55)
3. Object Palette - the nuts and bolts of the Beans builder (page 59)
4. The Diamond Components - what they are and how to use them (page 191)
5. Layout Editors - for laying out the components in a container (page 69)
6. Event Bindings - how to link objects using events (page 79)
7. Properties - a description of Property Sheets for components (page 60)
8. Resource Bundle Editor - how to internationalize your application (page 139)
9. The Project Window - a description of the Visaj project window (page 147). This may not be available if you are running Visaj from within an IDE - see the *Using Visaj from an IDE* section on page 2.
10. Generated Code - how to generate code and what it looks like (page 151)

11. Configuration - how to configure Visaj (page 161)
12. Quick Reference - the commands in Visaj at-a-glance (page 199)

1.1.1 Using Visaj from an IDE

If you are using Visaj from within an IDE (Integrated Development Environment), the Project Window may not be available to you. This is because the IDE may prefer to do all file handling itself. For the same reason, some of the File menu operations may also not be available. If this is the case, you will be able to perform these operations from the IDE.

1.2 Visaj

Visaj is a Java development tool which allows graphical development of the structure and interface of an application. When the graphical representation is complete, the developer can generate Java code. Not only does Visaj generate pure Java code, it is also written in Java. Both the generated code and Visaj itself are portable across all platforms which support Java 1.1 and above.

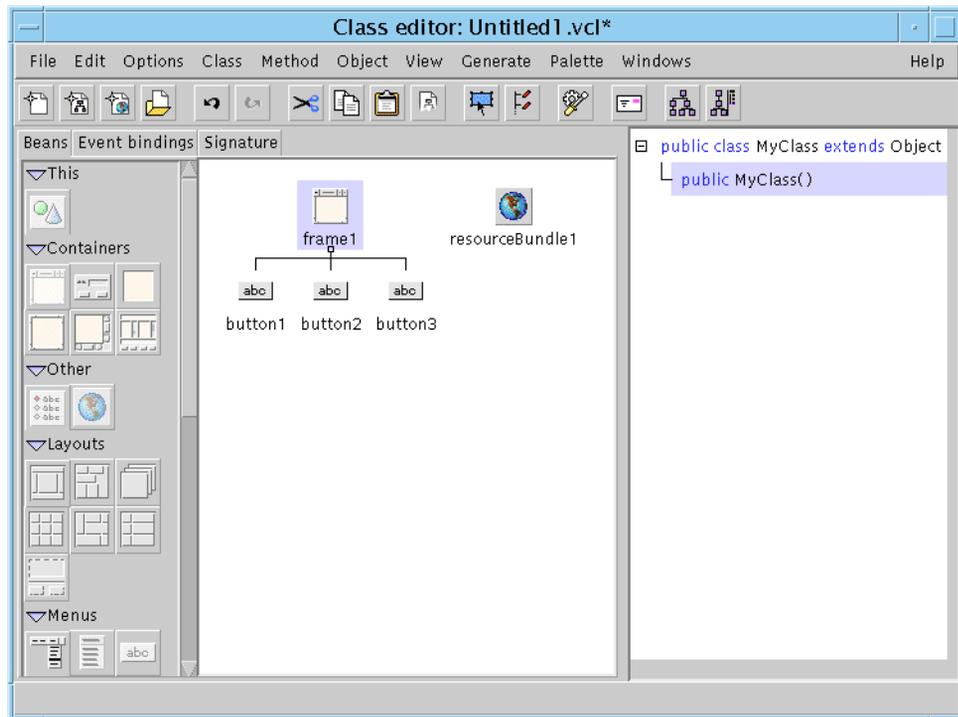


Figure 1-1 Class Editor

Development of an application with Visaj is quick and simple. The powerful Class Editor is displayed first. This is where the substance of the user interface is defined and source code generated. Apart from designing and editing your class file, buttons on the toolbar and in the menus lead you to the project window for organizing the files within your project, and to the Resource Bundle Editor for the internationalization of your application.

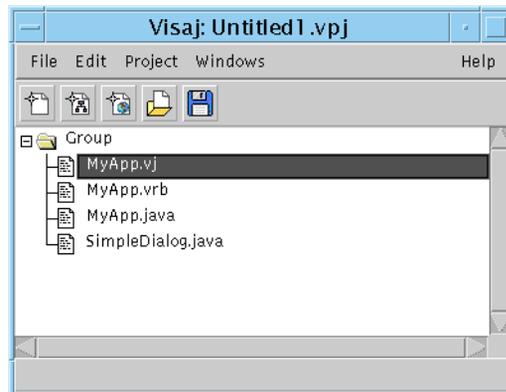


Figure 1-2 Visaj Project Window



Figure 1-3 Resource Bundle Editor

1.3 How To Use This Document

This document describes all the functions of Visaj. Each major feature of Visaj has its own chapter.

Chapter 3, “Visaj Tutorial”, starting on page 13, leads you through the major areas of Visaj to create a simple yet functional application. Starting with this tutorial will give you a thorough grounding in the principles of Java application development with Visaj.

Other chapters serve as both a guide and reference for each aspect of Visaj. Appendix B, “Quick Reference”, starting on page 199, helps you quickly find your way around Visaj.

The list of books in Appendix C, “Bibliography”, starting on page 209, suggests some titles for further reading on particular topics relevant to application development with Visaj.

Use the table of contents and index pages to go straight to information on a particular area of Visaj.

1.4 *Online Help*

Online help is available from the Class Editor, the Resource Bundle Editor and the Project window in Visaj. To view the list of help topics, select “Index” from the Help menu in any of these windows. A dialog containing a list of hypertext links to help topics relevant to the current window is displayed, as shown in Figure 1-4. Double-clicking over a topic displays a separate window containing help on the selected topic, as shown in Figure 1-5. This window has three buttons at the top which allow you to move forwards and backwards through the list of help topics and to change the font of the text in the window. References to subjects covered elsewhere in the help are shown as highlighted, underlined hypertext links. Clicking over one of these displays the relevant help information.

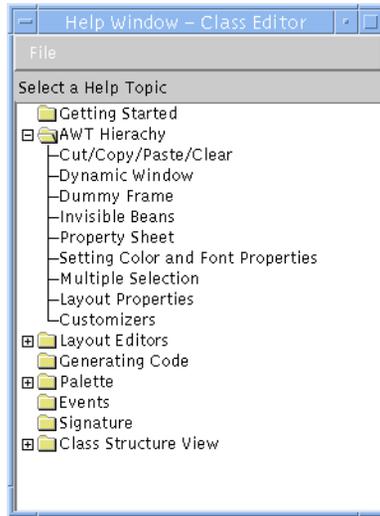


Figure 1-4 Online Help Index Window

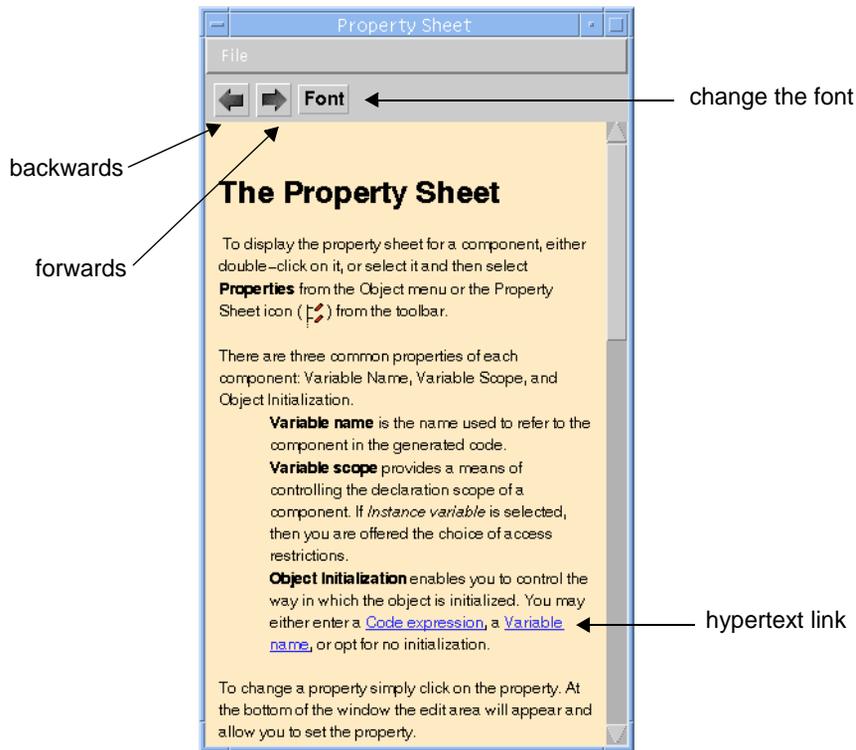


Figure 1-5 Help on an Individual Topic

1.5 Online User Guide

This user guide is available from Visaj in html format. Select "User Guide" from the Help menu.

1.6 Prerequisites

Visaj will run on any platform with JDK 1.1. Although Visaj is intuitive and easy to use, you will need to understand the Java language in order to be able to create a class structure for your application, to design and customize your user interface and to use the code generated by Visaj.

Integrating SNIFF+J with Visaj

2

2.1 Introduction

The Visaj integration with SNIFF+ allows you to combine Visaj's graphical GUI design features with SNIFF+'s source code engineering functionality. The symbol information and inheritance relationships of Visaj generated source code can therefore be directly browsed, version controlled, and edited in SNIFF+.

2.2 Installation

Note – *The Visaj Resource Bundle Editor, Image Editor and Project Window are currently not part of the SNIFF+ Visaj integration.*

Note – *Once Visaj is selected as part of the SNIFF+ installation it is automatically installed on your computer, you need not install Visaj separately.*

2.2.1 Requirements

- Integrating SNIFF+ with Visaj is only possible since SNIFF+ 3.1.
- You need JDK 1.1.3 or higher installed on your computer.
- JDK must be in your path.

Note – *You can download the JDK from <http://java.sun.com>*

For JDK 1.2 users

We suggest that you do the following to improve overall performance.

In the SNIFF_DIR/bin/runvisaj.sh script, add the following option:

- `-Djava2d.font.usePlatformFont=true`

after `javaw`

2.2.2 *Selecting Visaj as part of your SNIFF+ Installation*

- **On Windows**, Visaj is part of the Java package. To integrate SNIFF+ with Visaj, make sure that you select the **Java package** as part of your SNIFF+ installation.
- **On Unix**, To integrate SNIFF+ with Visaj, make sure that you select the **Visaj package** as part of your SNIFF+ installation.
- For more information on how to install SNIFF+, please refer to the SNIFF+ Installation Guide for Windows/Unix.

2.3 *Adding Visaj projects to a SNIFF+ project*

2.3.1 *Adding a new Visaj project*

In the SNIFF+ Project Editor:

1. Make sure that the relevant SNIFF+ project is highlighted in the Project Tree.
2. From the menu, choose either
Project → **Add Visaj Project to** *<Projectname>*, or choose
Tools → **Visaj**
3. In the dialog that appears, enter a name for the new project and press **Ok**.
4. Choose **Project** → **Save** *<Projectname>* to save the modified project.

2.3.2 Adding an existing Visaj project

First, copy your Visaj project to your SNIFF+ project directory. Make sure that the Visaj project file type (.vcl) and the Java file type (.java) are part of the SNIFF+ project. If they aren't, add them to the project. For details, please refer to the SNIFF+ User's Guide.

Then, in the SNIFF+ Project Editor:

1. Make sure that the relevant SNIFF+ project is selected in the Project Tree.
2. From the menu, choose **Project** -> **Add/Remove Files to/from** <Projectname>.
3. In the Add/Remove Files dialog that appears, select the Visaj project file, press the **Add** button, then press **Ok**.
4. Choose **Project** -> **Save** <Projectname> to save the modified project.

2.4 Working with SNIFF+ in Visaj

2.4.1 Loading the Visaj project into the Visaj Class Editor

In any SNIFF+ tool, choose **Tools** -> **Visaj**

OR

In the SNIFF+ Project Editor, double-click on the Visaj project file

2.4.2 Java code generation

Java code is automatically generated and stored in your SNIFF+ project directory when you

- save a file in the Visaj Class Editor
- execute commands in the SNIFF+ menu of the Visaj Class Editor
- modify the properties of a class in the Visaj Class Editor

2.4.3 Accessing SNIFF+ commands

A custom menu called **SNIFF+** is available in Visaj's Class Editor. The menu allows you to directly

- browse the locally defined and inherited members of a class
- browse the inheritance relationships of classes
- edit the generated source code
- browse the symbols used in the project

For a description of the menu commands in the SNIFF+ menu, please see “The SNIFF+ Menu” on page 203.

2.5 Tools not supported in the SNIFF+ Visaj integration

- Resource Bundle Editor
- Image Editor
- The Project Window

In the SNIFF+ Visaj integration, menu entries for the above tools don't appear in the Class Editor.

The rest of this manual is based on the original Visaj documentation therefore please ignore documentation related to the tools which aren't supported in this integration, i.e., those mentioned above.

Note – *In the SNIFF+ Visaj integration, java code is automatically generated and stored in your SNIFF+ project directory, for details please see “Java code generation” on page 11.*

This chapter provides a quick tutorial which takes you through the steps required to create the simple application shown in Figure 3-1. As you follow the step-by-step instructions in the tutorial you will be introduced to some of the major features of Visaj.

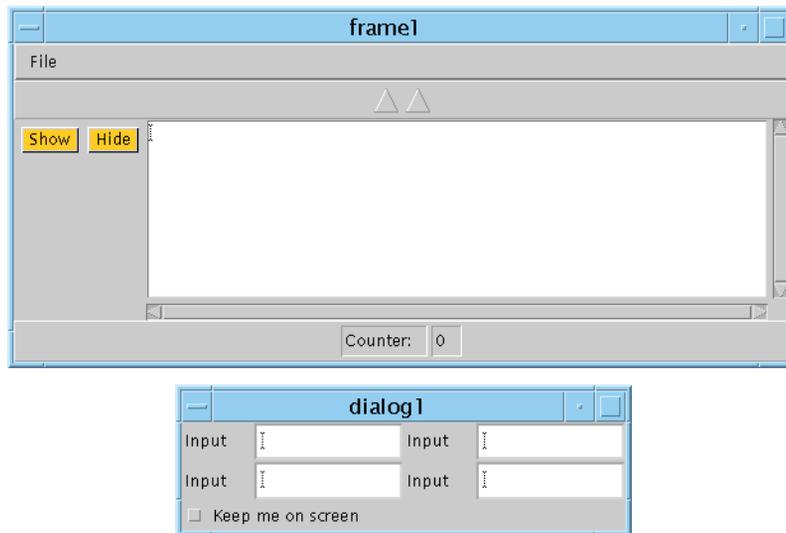


Figure 3-1 Final application

3.1 The Tutorial

The tutorial is divided into the following sections:

1. Part One - Building the Hierarchy. How to put together the various elements of the user interface.
2. Part Two - Layouts. How to lay out the those elements.
3. Part Three - Adding Event Bindings. How to build actions into the application.

At the end of each of the three parts listed above, you are given the choice of generating code and stopping the tutorial or continuing. Do as much of the tutorial as you feel you need to get started. Once you have done that, the following sections help you to finish the application:

4. Generating Code.
5. Adding Your Own Code - you will only need to do this if you complete Part Three - Adding Event Bindings.
6. Compiling and Running.

Remember that Visaj provides extensive online help, see the *Online Help* section on page 5 for details.

3.1.1 Part One - Building the Hierarchy

1. Start Visaj.

See the *Integration with an IDE* section on page 161 for details on how you can start Visaj from your IDE (Integrated Development Environment).

The first window displayed by Visaj is the Class Editor. This is described more fully in Chapter 4, “The Class Editor” on page 37.

The tutorial uses two components, the CountingLabel and the Ticker, which are supplied with Visaj for the purpose of demonstrating how to use invisible Beans. To have these components on your palette, you must first load in the JAR file containing them.

2. Select “Load Jar file...” from the Palette menu.

A File Dialog is displayed, ready for the name of the JAR file.

3. Select `demos.jar` from the Visaj install directory¹.

The extra components are merged into the existing palette.

Note – *Once you have completed this tutorial, you will have a saved design containing some of the beans in `demos.jar`. In order to open the saved design, you will have to load `demos.jar` first.*

See the *Loading JAR Files* section on page 60 for more information on this topic.

4. Click on the Frame icon from the component palette along the left hand side of the window.

The *Object Palette* section on page 59 provides more detail on using the component palette.

At this point, a separate window appears on the screen. This is the dynamic display and shows exactly the user interface that you are building.

5. Click on the MenuBar icon.

The containment hierarchy now shows the Frame with a MenuBar child. Since the MenuBar is a container Visaj has automatically selected this component in the hierarchy ready for the Menu children to be added.

6. Add a Menu component and 2 MenuItems to the design.

7. Click on the Frame at the root of the hierarchy.

8. Add a Panel from the Containers section and two Buttons from the Basic section.

The Buttons are added as children of the Panel.

You should now have the hierarchy shown in Figure 3-2.

1. The *install directory* is where Visaj has been installed by the installer, not the temporary area where Visaj is “unpacked” on Microsoft Windows.

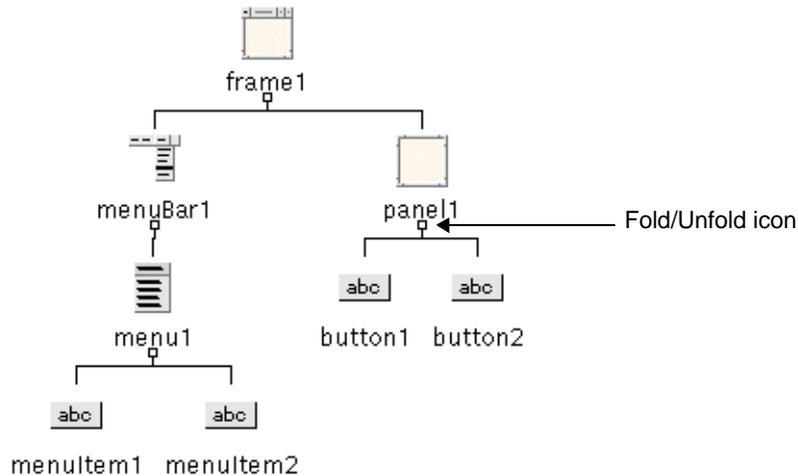


Figure 3-2 Initial hierarchy

9. **Select menuBar1 and select “Fold/Unfold nodes” from the View menu to fold this part of the hierarchy.**
Alternatively, you can click over the small box under MenuBar1. This is the Fold/Unfold icon.
10. **Select panel1 and again choose Fold from the View menu, or click over the small box under Panel1.**
11. **Add a Toolbar from the Diamonds section to your Frame.**
12. **Add two ArrowButtons, also from the Diamonds section, to your Toolbar.**
13. **Add a StatusBar (from the Diamonds) to the Frame.**
14. **Add a Label (from Basic) and a CountingLabel (from the demos.jar section) to the StatusBar.**
The CountingLabel is one of the Beans loaded from `demos.jar`. A separate section for all the Beans found in that file is added to the bottom of the object palette.
15. **Select the Frame again and add a TextArea (from Basic) to it.**

16. **Make sure no objects are selected by clicking over the design area background.**

To add invisible Beans, there must be no current selection.

17. **Add a Ticker invisible Bean to your design. This is found in the `demos.jar` section at the bottom of the object palette.**

Invisible Beans appear at the top of the design area alongside hierarchy root objects, as shown in Figure 3-3.

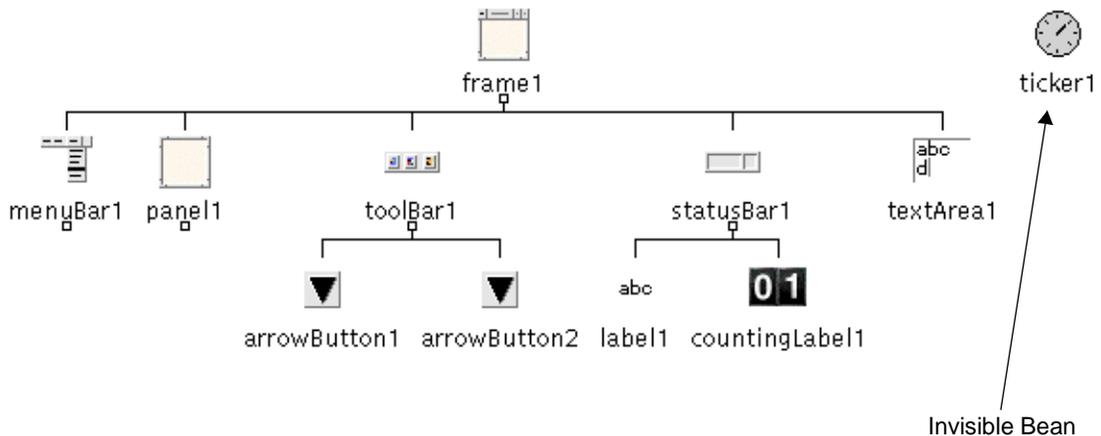


Figure 3-3 Full window hierarchy

Now that you have finished adding components and Beans, your dynamic display should appear as shown in Figure 3-4.

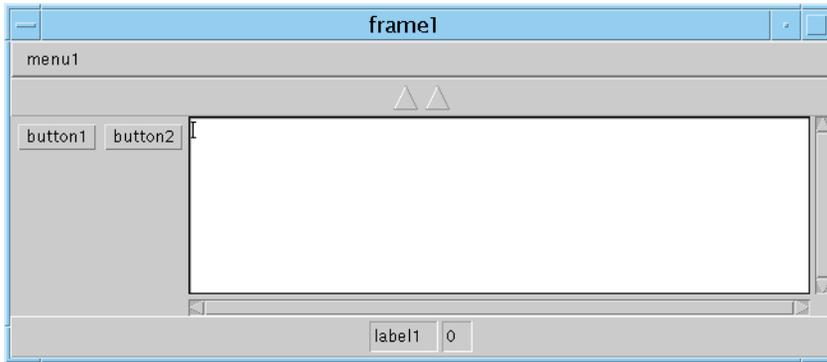


Figure 3-4 Initial window

- 18. Choose “Save” from the File menu and type “mydesign.vcl” into the File Dialog.**

The “vcl” suffix is a convention for Class Editor save files. It is good practice to save your design at regular intervals.

Setting Properties

Having added the components to the design we now need to set some properties.

- 19. Unfold the MenuBar branch of the hierarchy and double-click on menu1 to display the Property Sheet.**

By default, double-clicking on an object displays the object’s Property Sheet. Alternatively, you can display the Property Sheet by pressing the toolbar icon shown in Figure 3-5.



Figure 3-5 Property Sheet Icon on Toolbar

- 20. Select “label” from the java.awt.MenuItem group of properties. The Editor for this property type appears at the bottom of the panel. Change the label from “menu1” to “File”.**

☞ More information on the setting of properties and the effect this has is given in the "Properties" section on page 60.

We have some more properties to set. There is no need to close the Property Sheet and redisplay it, as it updates according to the current selection. If there is no selection, the title of the Property Sheet changes to reflect this.

21. **With the Property Sheet still displayed select first menuItem1 and then menuItem2 changing the labels to "Open" and "Exit" respectively.**
22. **Click on the background to deselect all objects in the design area.**
23. **Unfold panel1.**
Either click the fold icon underneath it or select "Fold/Unfold nodes" from the View menu.
24. **Use the mouse to draw a band around the two button children of the panel (click and hold the mouse button down to draw a box around the two buttons).**

☞ Multiple selection can be used to modify the properties of many components simultaneously.

25. **Change the background color property to "orange". The change is applied immediately to both buttons.**
Because the two labels are highlighted in the dynamic display, it looks as though the *foreground* color has been set. When you select something other than the labels, you can see the correct background color.
26. **Select button1, with the Property Sheet still displayed, and change its label property to "Show".**
27. **Select button2 and change its label property to "Hide".**
28. **Select the label child of the StatusBar and change the text property to "Counter:".**
29. **Close the Property Sheet.**
30. **Save your design.**
It is good practice to save your design at regular intervals.

You have now created a simple design with a default layout. If you wish to stop following the tutorial at this point, go straight to the *Generating Code* section on page 33. That section describes how to generate code for your application. You may continue the tutorial in the following section, which describes the Layout Editor and adds a dialog to the new user interface.

3.1.2 Part Two - Layouts

All container type components, such as the Frame, have a layout property associated with them. This *layout* allows you to arrange the container's children in a pre-defined manner. You can change the layout type of a container, as with any other property - this is demonstrated in Step 37 below. Visaj provides interactive dialogs which help you to arrange the container's children according to the rules of the container's layout. See the *Layout Editors* section on page 69 for details on these.

This section of the tutorial demonstrates the editors for the BorderLayout and the GridBagLayout. The GridBagLayout belongs to the new sub-dialog, which is added in this section.

The layout of a Frame container defaults to a BorderLayout. The BorderLayout allows you to position components in North, East, South, West and Central positions. Visaj's powerful layout editors make this easy.

- 31. With the Frame selected in the hierarchy, press the Layout Editor button in the toolbar. The BorderLayout Editor dialog appears as shown in Figure 3-6.**

The *Border* section on page 70 describes the Border Layout Editor.

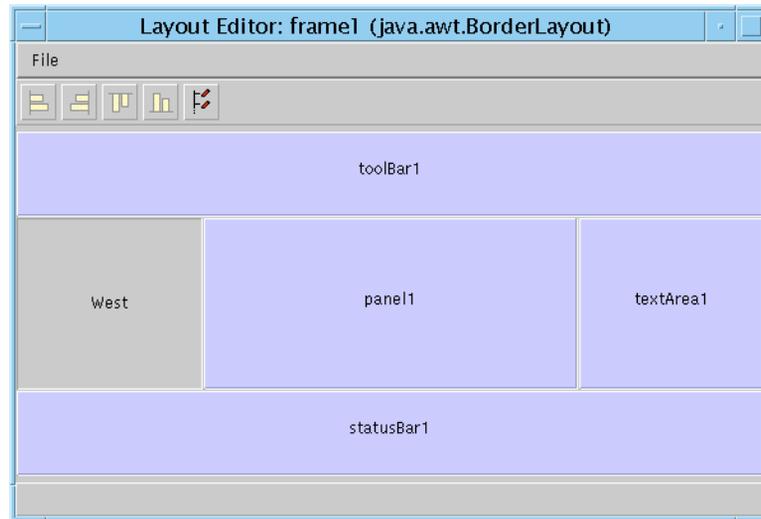


Figure 3-6 Border Layout Editor: first view

32. Use the mouse to drag the components to the correct positions so that they appear as shown in Figure 3-7.

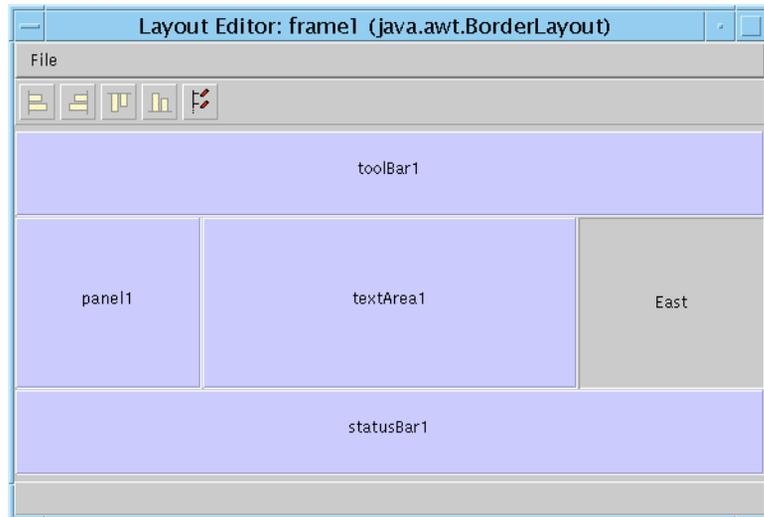


Figure 3-7 Border Layout Editor after arranging components

33. **Close the Layout Editor.**
34. **Save your design.**
It is good practice to save your design at regular intervals.

Adding Another Dialog

The next stage of the tutorial shows how to add a sub-dialog to your design. Dialog components need a parent, so we are going to make it a child of the Frame.

35. **Select the Frame, if it is not selected already.**
36. **Add a Dialog, from the object palette, to the Frame.**
This is found in the Containers section.
37. **With dialog1 selected, display its Property Sheet.**
38. **Select the layout property from the java.awt.Container section and choose GridBagLayout from the option menu at the bottom of the Property Sheet.**
39. **Add 4 Labels and 4 TextField children to the dialog.**

40. Add a Checkbox to the dialog.
41. Select the Checkbox and change its “label” property to “Keep me on screen”.

☞ Go back to the "Setting Properties" section on page 18 if you need a reminder of how to do this.

GridBag Layout Editor

42. Select the dialog and display the GridBagLayout Editor using the Layout Editor toolbar button.
This is shown in Figure 3-8.



Figure 3-8 Layout Editor Toolbar Icon

43. Drag the components in the editor to achieve the layout shown in Figure 3-9.

☞ See the GridBag section on page 72 for a detailed description of the GridBag Layout Editor.

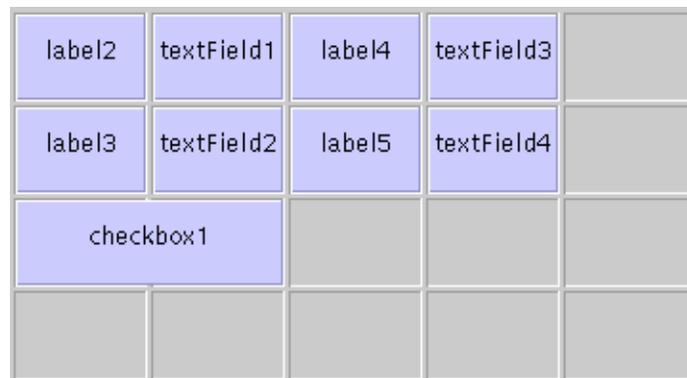


Figure 3-9 GridBag Layout Editor

- 44. Select checkbox1 in the Layout Editor and drag the right edge so that it fills two columns, as shown in Figure 3-9.**
This is so that the all the labels in the first column are not forced to be the same width as the Checkbox.
- 45. With Checkbox1 still selected, set the Anchor option menu to “West” and the Fill option menu to “None”. Press “Apply”.**
This puts the Checkbox on the left and stops it trying to fill both columns.
- 46. Select label2, unset the RelativeX and RelativeY toggles and press “Apply”.**
The other components lost their relative x and y settings when they were moved. Relative x and y settings are not required for this tutorial.
- 47. Hold down the shift key and click on each of the components except the Checkbox.**
We have already set the Fill option for the Checkbox.
- 48. Change the Fill option menu from “None” to “Both” and press “Apply”.**
- 49. Click in an empty cell to deselect the components.**
- 50. Hold down the shift key and click on the two TextFields in the top row of the grid. Change the Column Weight to “1” and press “Apply”. Repeat for the two Labels in the top row and change the Column Weight to “2” and press “Apply”.**
The column weight affects the way objects stretch when the container grows horizontally.
- 51. Close the GridBag Layout Editor.**
- 52. Select one of the four Labels which are children of dialog1. Use multiple selection (Shift + mouse button 1) to select the other three as well.**
- 53. Change the text property of the 4 Labels to “Input”.**
- 54. Select the four Textfields and change the columns property to “10”.**
- 55. Save your design.**

If you wish to stop following the tutorial at this point, go straight to Generating Code on page 33. The tutorial continues in the following section with actions being added to some of the buttons in the design.

3.1.3 Part Three - Adding Event Bindings

For the last stage of the tutorial, we shall add some basic functionality to the application. We will set up the buttons in the Panel to hide and show dialog1, set up the Ticker and CountingLabel so that they display the amount of time the dialog is shown on the screen and set up the Exit button in the File menu to exit the application.

☞ For more information on event bindings, see Chapter 6, “Event Bindings”, starting on page 79.

56. Fold away the components underneath dialog1 and under the ToolBar and StatusBar.

Do this so that you can see both the buttons and dialog1 in order to complete the next step. You may also need to press the “Method Editors Only” toolbar button. This is shown in Figure 3-10.



Figure 3-10 Method Editors Only Toolbar Icon

57. With the Control key held down, use your mouse to drag a line from button1 to dialog1.

This is a fast way of displaying the Event Binding Editor for a new binding, primed with the two objects at either end of the line drawn in the design area.

There are two other ways of displaying the Event Binding Editor:

1. Selecting the “Event bindings” tab panel on the left of the Class Editor window and pressing the “New” or “Edit” buttons.
2. Pressing the Event Binding button on the toolbar, shown in Figure 3-11.



Figure 3-11 Event Binding Icon on Toolbar

➤ See Chapter 6, “Event Bindings”, starting on page 79 for a full description of the Event Binding Editor.

58. Check that the Source of the event binding is “button1” and the Destination Object is “dialog1”.
59. Set the Type to “actionPerformed” and select the Handler Method “show()”. This is shown in Figure 3-12. Make sure that you select the “show” method which has no parameters. The “show” method *with* parameters is a deprecated method.

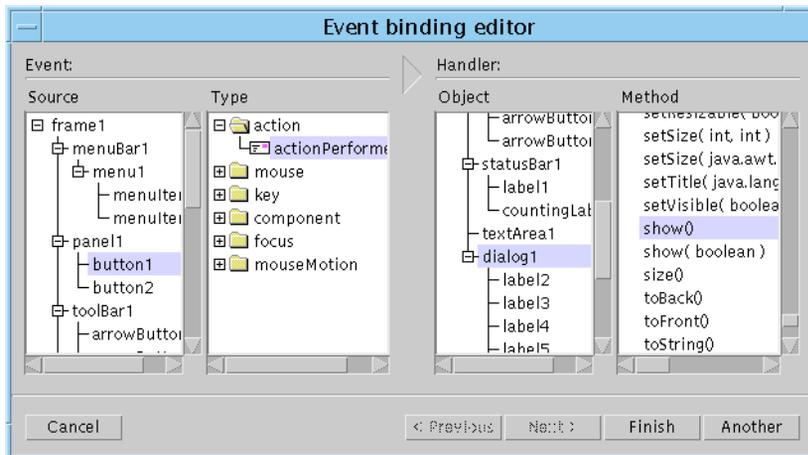


Figure 3-12 Event Editor

60. Press the “Finish” button. The completed event binding now appears in the Event Binding List. Pressing “Another” also creates the event binding, leaving the dialog on the screen ready for you to add another binding.

61. **Add another new event binding so that the Source is “button2”, the Type is “actionPerformed”, the Object is “dialog1” and the Method is “setVisible”.**

This method has a parameter; when you select it the “Next>” button becomes enabled. The “Finish” button is not enabled, showing that you have not completed the specification of the event binding.

62. **Press the “Next>” button. The second page of the Event Binding Editor is displayed, as shown in Figure 3-13.**

Now that you are on the second page, the “<Previous” button is enabled, allowing you to move back to the first page. The “Finish” button will not become enabled until you have entered a value for the parameter.

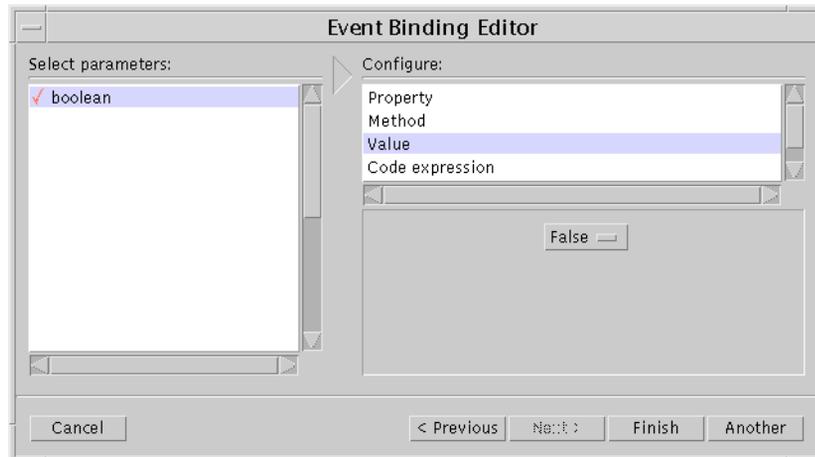


Figure 3-13 Event Binding Editor Second Page

63. **Select the boolean parameter, select “Value” from the list on the right and change the option menu to “False”.**
setVisible(false) hides the dialog.
64. **Press “Finish” to create this event binding.**

Try out the two new event bindings in the dynamic display. Press button2 - the dialog disappears. Press button1 - it reappears. We are now going to change this last event binding so that it hides the dialog only if the Checkbox in the dialog is not set.

65. Edit the last event binding.

Select the last event binding from the Event Bindings List in the Class Editor and press “Edit”. This is shown in Figure 3-14.

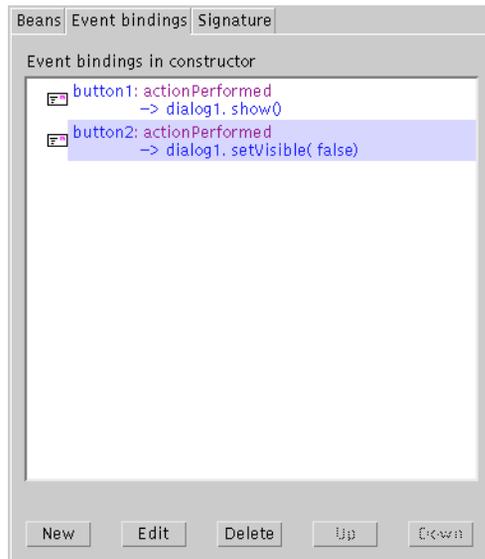


Figure 3-14 Event Binding to Edit

66. On the parameters page of the Event Binding Editor, select the boolean parameter.

The tick next to this parameter indicates that it has already been configured. The area on the right shows that you have set an explicit value for this parameter.

67. Select “Property” from the list on the right.

We are going to set the value of the parameter according to the property of another object. Two lists appear in the lower right of the Editor to allow you to select any object from your design and any properties of that object which have the same type as the selected parameter.

68. Select “checkbox1” (beneath “dialog1”) from the list of objects in the design.
69. Select “state” from the list of properties.
Figure 3-15 shows how to configure this parameter.

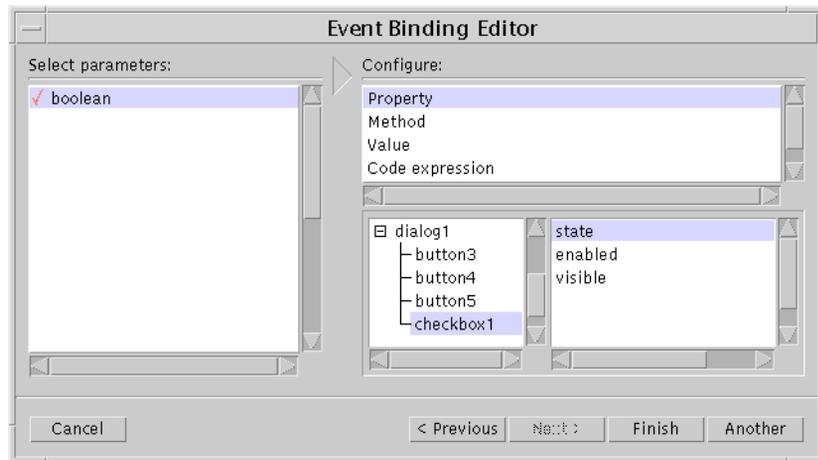


Figure 3-15 Checkbox Property for Boolean Parameter

70. Press “Finish”.
The binding is now set so that when the Checkbox is set, the dialog does not disappear. When the Checkbox is *not* set, the dialog *does* disappear. Try it out in the dynamic display.

Event Bindings for Invisible Beans

To demonstrate how the invisible Beans work, we are now going to add some more event bindings to make the CountingLabel count tenths of a second while the dialog is displayed.

71. Add another new event to the Show button (button1) to make the Ticker start, as shown in Figure 3-16.
The Source is “button1”, the Type is “actionPerformed”, the Destination Object is “ticker1” and the Method is “start()”.

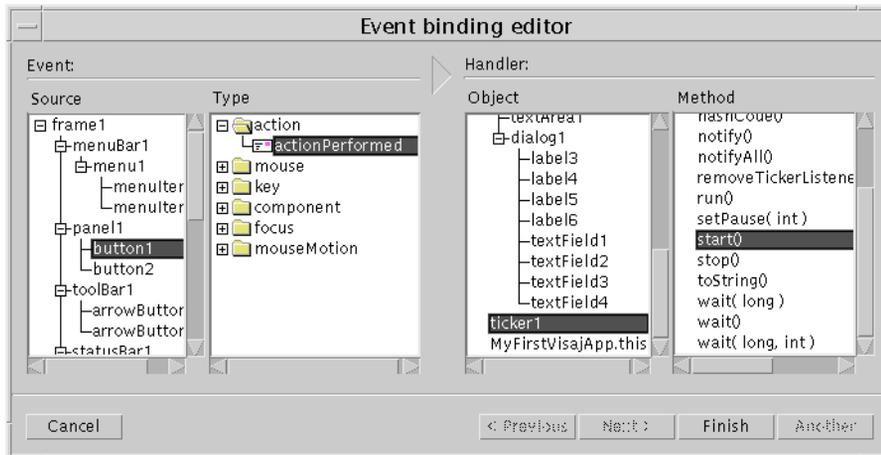


Figure 3-16 Ticker Start Event

72. Add another new event binding to button2 to stop the ticker, as shown in Figure 3-17.

For this binding, the Source is “button2”, the Type is “actionPerformed”, the Object is “ticker1” and the Method is “stop()”.

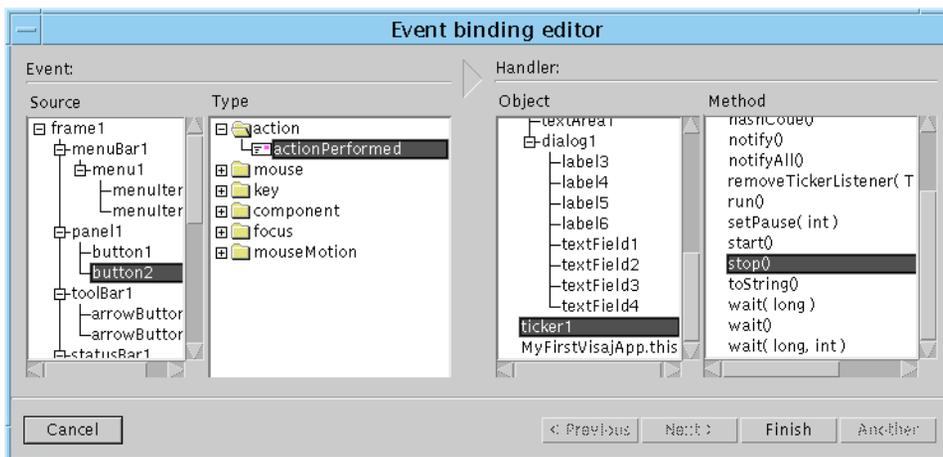


Figure 3-17 Ticker Stop Event

73. Now add a new event to the Ticker itself to increment the CountingLabel as it ticks, as shown in Figure 3-18. The Source is “ticker1”, the Type is “tick”, the Destination Object is “countingLabel1” and the Method is “increment()”.

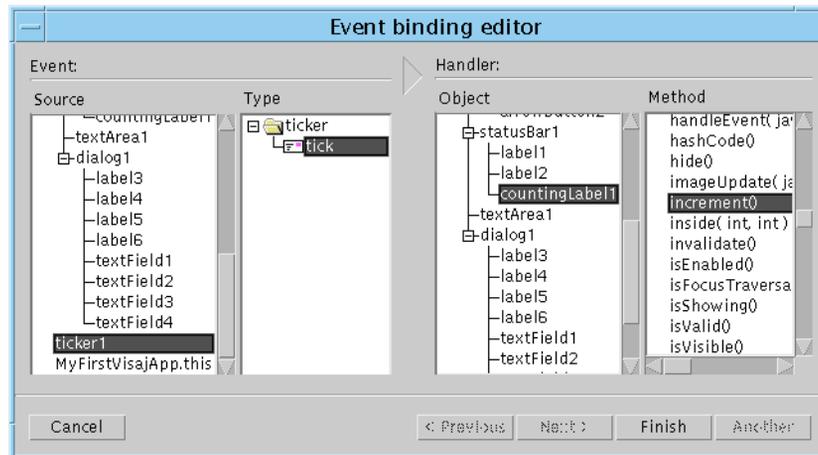


Figure 3-18 Ticker Tick and Increment Event

74. Try pressing the buttons in the dynamic display again and watch the CountingLabel show the seconds that pass when the dialog is displayed.

Event Bindings to Other Methods

Finally we are going to create a new method called “myExit” which will be called when the user selects the File/Exit button.

To do this, we shall add another method to our class. If you have hidden the Class Structure View, bring it back by pressing the “Show Method Editors and Class Structure” button on the toolbar. This button is shown in Figure 3-19.



Figure 3-19 Toolbar Icon to Redisplay Class Structure

75. **Select “Properties...” from the Class menu of the Class Editor.**
76. **Double click on the class name, change “MyClass” to “MyFirstVisajApp” and press “Apply”.**

Note – The name of the method in your class also changes because it is the constructor and therefore must have the same name as the class.
77. **Close the Properties dialog.**
78. **Select “Add new method” from the Method menu to add a new method to the class.**

A method named “method2” appears in the class structure.
79. **Select “method2”.**
80. **Display the “Signature” tab panel from the Method Designer and change the name of method2 to “myExit”.**
81. **Select the “MyFirstVisajApp” method in the Class Structure view again and display the Event Binding Editor.**
82. **Add a new event binding such that the Source is “menuItem2”, the Type is “actionPerformed”, the Destination Object is “MyFirstVisajApp.this” and the Method is “myExit()”, as shown in Figure 3-20.**

Make sure that “MyFirstVisajApp” is selected in the Class Structure View before adding this event binding.

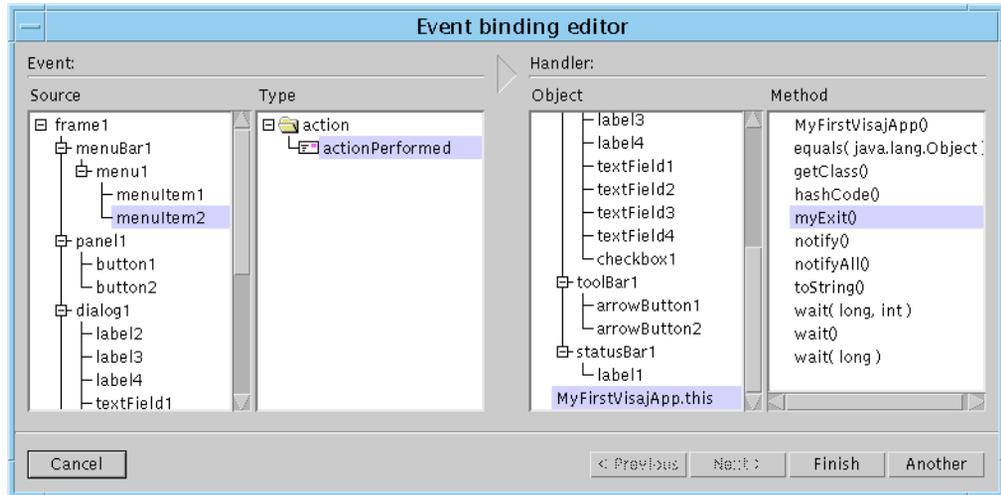


Figure 3-20 Calling New Methods in Event Binding Editor

83. Save your design.

The application design is now complete. The following section explains how to generate the code for the design.

3.1.4 Generating Code

1. **Select the constructor (MyFirstVisajApp) in the Class Structure View and set the “Main method” checkbox in the Method menu.**

This tells Visaj to generate a main method and call the constructor.

☞ See the “Main Method” section on page 49 for more information on generating a main method from Visaj.

2. **Select “Generate java...” from the Generate menu, specify a target directory in the directory selection box and press “OK”.**

See Chapter 11, “Generated Code”, starting on page 151 for more information on generating code from Visaj.

3. Save your Visaj design.

You can now exit Visaj if you wish.

Remember that if you wish to reload your design into Visaj, you will have to load the `demos.jar` file beforehand, as described in Step 2 and Step 3 on page 14.

3.1.5 Adding Your Own Code

You only need to do this part if you completed Part Three - Adding Event Bindings. If you did not, then go straight to the *Compiling and Running* section on page 35. All of the following is done external to Visaj.

 This tutorial shows you how to edit the generated code directly. See the "Adding Your Own Code - Subclassing" section on page 153 for a description of another way of adding code.

1. Edit the generated file, `MyFirstVisajApp.java`.

2. Find the “myExit” method and add the call to “`System.exit(0)`”. Make sure that you add your code outside of the special comments, so that it is retained if you regenerate the code.

The line to add is shown below.

```
protected void myExit() {
    //vj- <VJ-BeginMethodDef>

    //vj= <VJ-MethodCode>
    //vj+ <VJ-DefineAWTMembers>
    //vj- <VJ-DefineAWTMembers>

    //vj+ <VJ-EndAWT>
    //vj- <VJ-EndAWT>

    //vj+ <VJ-EventListenerClass>
    //vj- <VJ-EventListenerClass>

    //vj+ <VJ-AddEventListeners>
    //vj- <VJ-AddEventListeners>

    //vj= <VJ-Classes>
```

Add the next line:

```
System.exit(0);

//vjt+ <VJ-EndMethodDef>
}
```

☞ See the *Editing the Code* section on page 154 for more information on where you may safely insert your code.

3. Save the file.

3.1.6 Compiling and Running

You now have the source of your application. Perform the following steps outside of Visaj.

- 1. Add the `diamonds.jar` and `demos.jar` files from the Visaj install directory and the current directory (`.`) to your CLASSPATH. Compile the code.**

You must have these JAR files in your CLASSPATH because you are using some of the Diamond components and the two beans from `demos.jar` in your design. The *Using the Diamond Components* section on page 159 provides more information on this.

- 2. Compile your application.**

Remember that the java compiler is case sensitive. You should specify `MyFirstJavaApp.java` as the file to compile using exactly that capitalization.

- 3. When the code has compiled, run your application.**

4.1 Introduction

A Java application consists of one or more class files. Classes contain variables and methods. It is the methods in the classes which perform the functions of the application. With Visaj you can construct classes containing methods which create the user interface of your application. Classes are defined using the Class Editor. The Class Editor is the first window displayed when you run Visaj. See the *Integration with an IDE* section on page 161 for details on how you can start Visaj from your IDE (Integrated Development Environment).

The Class Editor, shown in Figure 4-1, consists of a menubar, a toolbar, a tabbed panel on the left of the window and a tree view of the class structure on the right.

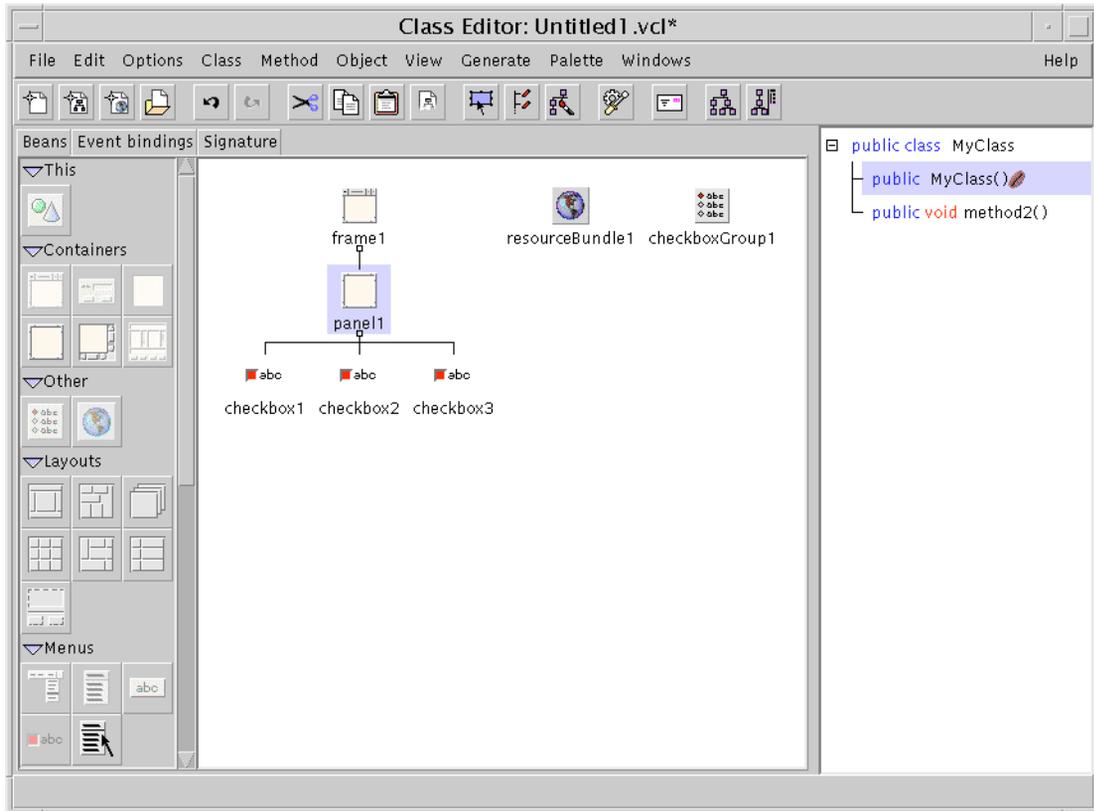


Figure 4-1 The Class Editor

4.2 Class Structure View

A tree structure of the class is shown on the right side of the Class Editor window. Each method in the tree can be selected by clicking over it. The area on the left of the Class Editor displays three sets of editable information for the currently selected method:

1. Bean hierarchy
2. Event bindings
3. Method signature

These are described in the *Method Editing* section on page 47.

4.2.1 Editing the List of Methods

Add and delete methods from the list of class methods, shown in Figure 4-2, by selecting the relevant item from the “Method” menu.



Figure 4-2 List of Class Methods

Methods shown with a bean icon are methods which are capable of creating or editing a bean - that is, they have a containment hierarchy associated with them. Methods without the bean icon would, typically, perform a non-interface operation such as the “myExit” method in the tutorial in Chapter 3 which simply exits the application.

4.2.2 Default Constructor

Visaj automatically gives you one method when you start a new class. This method is the default constructor. Change this to a simple class method in the Method Signature page of the Class Editor, described in the *Editing the Method Signature* section on page 48. Any method can be made the constructor by setting the appropriate toggle on the Method Signature page.

4.2.3 Editing Properties of the Class

The “Properties...” item in the Class menu displays the dialog shown in Figure 4-3.

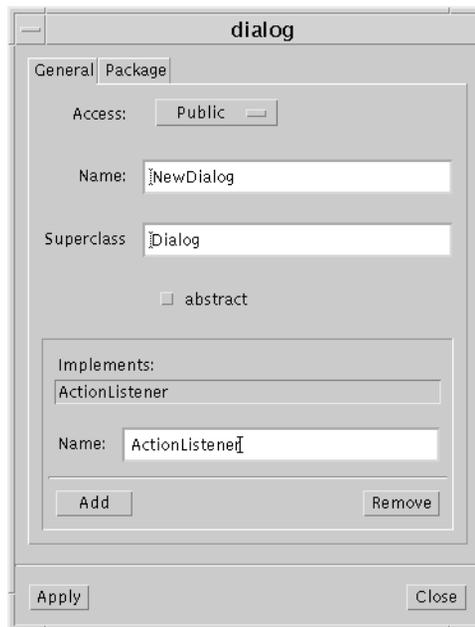


Figure 4-3 The Class Properties Dialog

This dialog allows you to edit the properties of the class. You can change its name, access and superclass. By default, the access type of the class is public, its name is “MyClass” and its superclass is `java.lang.Object`. You may also provide a list of interfaces which this class implements.

The class name is used as the filename of the Java source file when you generate Java. This is a requirement of the Java language. If you generate Java, change the class name and then regenerate code, the new class name is used as the filename of the new Java source file. In addition, if you have “Update existing files” selected in the Generate dialog, the previous Java source file is removed. This is to ensure that the generated source is always in step with the design since you have requested an “Update”. If you wish to retain your previous file, make sure that the “Update existing files” toggle is not selected. In this case, however, any changes you may have made to the generated source file are not retained.

Selecting the “Package” tab in the Class Properties dialog allows you to specify the package in which your class is found and any packages it imports.

Populate your class with new methods by selecting the “Add method” item from the Class menu of the Class Editor.

4.2.4 *Automatically Added Methods*

There are two types of method which may be added to your class automatically by Visaj:

1. Exception methods
2. Interface methods

Exception methods are added to your class when you create an event binding using a method which can throw an exception. The exception handler matches the type of exception which can be thrown, as shown in Figure 4-4. In this case, the event binding method handler was the “setPage” method of the Swing component, JEditorPane. This method can throw an IOException.



Figure 4-4 Exception Handler Added to Class

These are convenience methods added by Visaj. If an exception is thrown, this method is called. You can safely add code for handling the exception - the code is retained if you choose “Update existing files” in the Generate dialog. You cannot edit these methods (except to make them abstract) because they are expected to have a particular signature. Although you can remove these methods (by selecting “Delete” from the Method menu), they reappear in your class structure editor when code is generated if they are still referenced anywhere in the generated code.

If you change the properties of your class (by choosing “Properties” from the Class menu) so that it implements an interface, the methods of the interface appear in the class structure editor. These methods must be generated otherwise your application will not compile. You cannot edit these methods except to make them abstract.

4.3 “this”

`this` is a keyword in the Java language and refers to “this object” - that is, an instance of the current class. For example, in the following piece of code the keyword `this` disambiguates between the class variable `width` and the parameter `width` which is passed into the constructor:

```
class Example {
    int width;
    public Example(int width) {
        this.width = width;
    }
}
```

`this` is selectable from the palette. The type of `this` is whatever has been defined as the superclass of the current class. By default, the superclass is `Object`. The palette icon, Property Sheet and behavior all reflect the superclass. When selected from the palette, `this` is added to the design as you would expect from its type. So, if the superclass is derived from `java.awt.Component` or `MenuItemComponent`, `this` is added to the hierarchy. If it is not (an instance of `java.lang.Object` or `java.awt.Color`, for example), then `this` appears in the invisible bean area.

4.3.1 An Example

The use of `this` can be explained more fully with a simple example. Try the following, assuming you have a fresh Class Editor window in front of you:

1. Specify `Frame` as the superclass by selecting “Properties...” from the Class menu and filling in the dialog as shown in Figure 4-5.

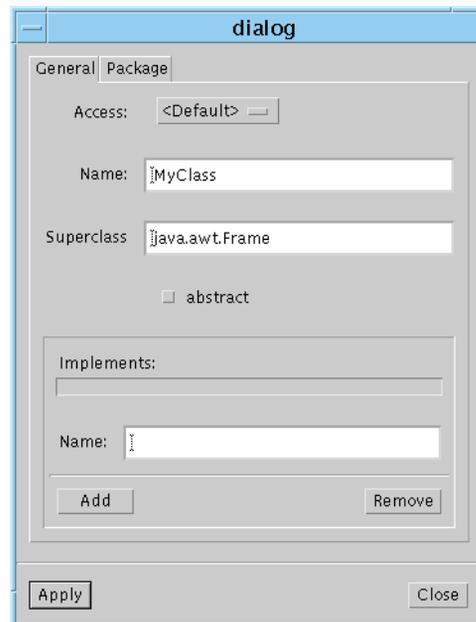


Figure 4-5 `java.awt.Frame` as the superclass

The Class Structure now tells you that the superclass of your current class is `Frame` and the palette icon has changed appropriately, as shown in Figure 4-6.



Figure 4-6 this as a Frame

2. **Select this from the palette and add a Label and a Button to it, as shown in Figure 4-7.**

this is a Frame, so you can add children to it.

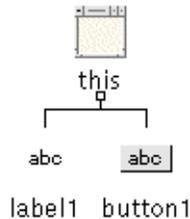


Figure 4-7 this in a Hierarchy

3. **Generate code for your small design.**
See the *How to Generate Code* section on page 151 if you are not sure how to do this.
4. **Look at the code which has been generated.**
The relevant part of that code is shown below.

The code fragment listed below shows that the Label and Button components have been added to the class itself and initialized in the constructor, which is the method in which they were defined in Visaj.

```
public class MyClass extends java.awt.Frame {
//vj- <VJ-BeginClassDef>
```

```

//vj+ <VJ-DataMembers>
protected Label labell ;
protected Button button1 ;
//vj- <VJ-DataMembers>

//vj= <VJ-Methods>

//vj+ <VJ-BeginMethodDef>
// Method# 1
public MyClass() {
//vj- <VJ-BeginMethodDef>

    //vj= <VJ-MethodCode>
    //vj+ <VJ-DefineAWTMembers>
    this.setTitle( "null" );
    labell = new Label();
    labell.setText( "labell" );
    button1 = new Button();
    button1.setLabel( "button1" );
    {
        String strConstraint;
        strConstraint = "Center";
        this.add(labell, strConstraint, -1);
        strConstraint = "North";
        this.add(button1, strConstraint, -1);
    }
    this.pack();
    this.show();
...

```

For comparison, the following code fragment shows what is generated when a Frame is selected from the palette, instead of this:

```

public class MyClass extends java.awt.Frame {
//vj- <VJ-BeginClassDef>

    //vj+ <VJ-DataMembers>
    protected Frame frame1 ;

```

```
protected Label label1 ;
protected Button button1 ;
//vj- <VJ-DataMembers>

//vj= <VJ-Methods>

//vj+ <VJ-BeginMethodDef>
// Method# 1
public MyClass() {
//vj- <VJ-BeginMethodDef>

    //vj= <VJ-MethodCode>
    //vj+ <VJ-DefineAWTMembers>
    frame1 = new Frame();
    frame1.setTitle( "frame1" );
    label1 = new Label();
    label1.setText( "label1" );
    button1 = new Button();
    button1.setLabel( "button1" );
    {
        String strConstraint;
        strConstraint = "Center";
        frame1.add(label1, strConstraint, -1);
        strConstraint = "North";
        frame1.add(button1, strConstraint, -1);
    }
    frame1.pack();
    frame1.show();
    ...
}
```

Using this in a method design is of most use for classes that are derived from AWT components, as in our example above. This allows methods to add components to the base component, which is the class itself. In such a case, you would normally use the constructor, as we have done in this example. Then, when an instance of the class is created, the components within it are created too.

4.3.2 *Points to Remember*

You may have only one instance of “this” per method. If you change the superclass of the current class after having added “this” to any methods, they become simple variables of the type they were when added to the method. That is, if “this” is added to the containment hierarchy when the superclass was a Panel and the superclass is now a Dialog, “this” becomes a simple Panel and retains its position in the hierarchy and any properties which have been set on it.

4.4 *Method Editing*

The left area of the Class Editor contains a tabbed panel which provides three editors for a method. Select a method from the class structure on the right to view and edit information on that method. The three editors are:

1. Beans (the user interface builder)
2. Event bindings
3. Method signature

The user interface builder, described in Chapter 5, “Beans View”, starting on page 55, allows you to build the user interface Beans and invisible Beans of your application. This is a powerful graphical building tool complete with properties, layouts and a true representation of the user interface.

Event bindings are connections between Java beans. They provide a quick way of adding functionality to your user interface. They take effect immediately in the dynamic display and are generated into the code. Use the Event Binding Editor to link the action of one bean to a method in another. This is fully described in Chapter 6, “Event Bindings”, starting on page 79.

Editing a method signature is explained in the following section.

Pressing the “Method Editors only” button on the toolbar hides the Class Structure View, thereby allowing more space for the design of your methods. Pressing the “Show both” button on the toolbar makes the Class Structure reappear.

4.4.1 Editing the Method Signature

Selecting the “Signature” tab on the left of the Class Editor displays the panel shown in Figure 4-8.



The screenshot shows the "Signature" tab of the Class Editor. It features a "Type" field with "void" and a "Name" field with "method". The "Access" is set to "Public". There are checkboxes for "Constructor", "Final", "Abstract", "Synchronized", "Static", and "Native", all of which are currently unchecked. Below these are two large empty text areas for "Exception/s:" and "Parameter/s:". At the bottom of each area are "Add" and "Remove" buttons. Below the "Exception/s:" area is a small "Exception:" field with a text input and a "Type:" label. Below the "Parameter/s:" area is a small "Type:" field with a text input and a "Name:" label.

Figure 4-8 Method Signature

Use this panel to modify the method declaration for the method that has been selected, and is shown highlighted in the list. You can change a method's name, its return type, select one or more of the modifiers static, final, synchronized, native or abstract, change the method's access type from “public” to protected, private or “default” no specified type.

If you select the modifier “native” or “abstract”, then the method cannot contain code, and the Beans and Event pages are disabled. If you already have beans in this method, you cannot set either of these toggles.

If you make any method in the class “abstract”, then the class itself is also abstract.

The two lists in the lower area of the panel allow you to give the method parameters, or arguments, and to state which exceptions are to be thrown by the method.

Note that all three of the type fields here (and the superclass field in the class properties dialog) do not require a full class name, if the class is in one of the base java packages, or in Swing. You need only type “IOException” and it will resolve this as “java.io.IOException”.

In this panel, and in the method list, blue indicates a Java keyword and red either a built-in Java type, or a class.

4.4.2 *Main Method*

You can tell Visaj whether or not to generate a main method using the “Main method” item in the Method menu. The possible options are:

1. You have not set the “Main method” checkbox for any of the methods in your class. No main method is generated.
2. The constructor has the “Main method” item set for it. This results in just the constructor being called from the generated main method.
3. A method other than the constructor has the “Main method” checkbox set. The generated main method calls this method *after* calling the constructor.

A method selected in this way is indicated by a star in the Class Structure View.

4.5 Importing X-Designer Save Files

Visaj can import designs created in X-Designer, the graphical user interface builder for Motif. This enables you to move legacy Motif C/C++ designs quickly to Java. Visaj also imports designs created in Sun Microsystems' Workshop Visual in exactly the same way.

4.5.1 Importing the Design

The File menu contains an Import pullright menu. This contains one item: "X-Designer bridge file". Selecting this displays a file dialog allowing you to specify the design to import.

There are some features in X-Designer which have no equivalent in Visaj. If your import file contains any of these, a dialog is displayed describing the situation. The possible messages are:

1. One or more classes were expanded into simple component hierarchies.
2. String/font/color/pixmap objects in your design were expanded into simple property settings.
3. Forms were converted to null layouts.

Each of these is described in the following sub-sections.

Apart from the discrepancies described above, the imported design should reflect the Motif application exactly.

One or more classes were expanded into simple component hierarchies

In X-Designer, you can make any component a class. In Visaj, to break your design into classes you build separate hierarchies in different designs, generate code, compile it and then add the generated hierarchies to the palette as beans for use in larger designs.

Converting from the X-Designer model to the Visaj model is better done under your control than in an automated import function, as the compiler/debugger/etc. environment can vary greatly between users.

So, while importing, Visaj ‘expands’ classes, basically ignoring whether or not any given component is a class. You can cut sub-hierarchies and paste them into new Visaj designs as you see fit after importing the design.

String/font/color/pixmap objects in your design were expanded into simple property settings

Visaj has no equivalent to these (or for that matter, no equivalent to pixmap objects - but see above). So, all references to objects are expanded into simple resource values. For example, if you have a label `l` whose text is `<fred>`, `<fred>` being a string object with value “Hello!”, then after import, `l` will have the text “Hello!”, and `<fred>` will have been forgotten.

All XmForms in the design were changed to absolute positioning using a null layout

The layout manager `com.pacist.mwt.FormLayoutManager` is not supported. Components whose layout is controlled by this manager are positioned absolutely as a result. You are recommended to change your design to use one of the Java layout managers.

4.6 Applets

Make your class into an applet by specifying `java.applet.Applet` as the superclass, as described in the *Editing Properties of the Class* section on page 40. Having done this, the “this” icon on the object palette changes to the applet icon. Select “this” from the palette to add objects to your applet.

If you are using Swing, you can make your class into a `JApplet` from the Swing component set by making `javax.swing.JApplet` the superclass.

4.7 Generating Code

To generate source code for your class, select “Generate Java...” from the Generate menu or press the Generate button on the toolbar. Code generation is described in Chapter 11, “Generated Code”, starting on page 151.

4.8 Windows Menu

As well as providing a way of displaying a new window onto your Visaj save file and of moving between all open Visaj windows, the Windows menu allows you to display the Font and Color Selectors.

4.8.1 Font Selector

The Font Selector, shown in Figure 4-9, provides a means of selecting a font.

To set the font of a component, first display the component's property sheet. You can then use *drag and drop*: either drag the selected Font from the Selector (using the mouse button) and drop it directly over a property of type "Font" or select the property first and then drop the font into the area at the bottom of the property sheet where the current value is being displayed. Alternatively you may *copy* (Ctrl+C) from the sample area and *paste* (Ctrl+V) into the property sheet editing area.

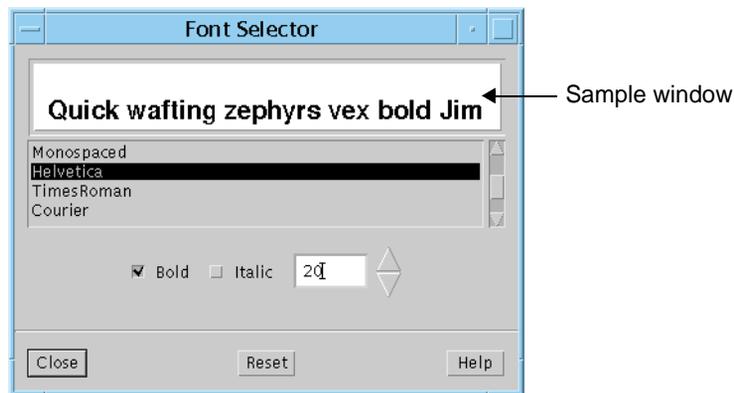


Figure 4-9 Font Selector

4.8.2 Color Selector

The Color Selector, shown in Figure 4-10, is a dialog with tabbed panels providing different color viewing models. Some viewing models require you to click over a color to select it, some provide sliders. The color you have most recently selected appears on the right of the sample area at the top of the Selector window. The left of this area shows the original color.

To set a component's color, display its property sheet, select the property which is expecting the color, drag your chosen color from the sample area of the Color Selector (with the mouse button) and drop it either directly into the property, as displayed in the sheet, or into the editing area at the bottom of the property sheet. Alternatively you may *copy* (Ctrl+C) from the sample area and *paste* (Ctrl+V) into the property sheet editing area.

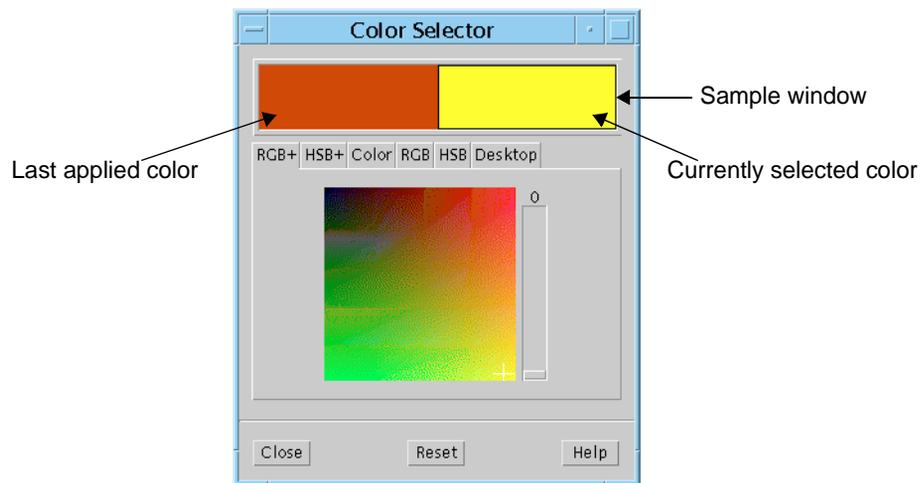


Figure 4-10 Color Selector

4.9 Displaying Other Tools

To display Visaj's other tools, either select one of the "New" options from the File menu or the toolbar, or open a saved file of the appropriate editor.

4.10 Option Menu Items

The Option menu contains items for your general use of Visaj. There are two items: “Authentication...” and “Java Console...”. Selecting “Authentication...” displays the Licensing dialog which is described in the accompanying document on licensing. If you are using Visaj from an IDE (Integrated Development Environment), this option is disabled as licensing is controlled by the IDE and not by Visaj.

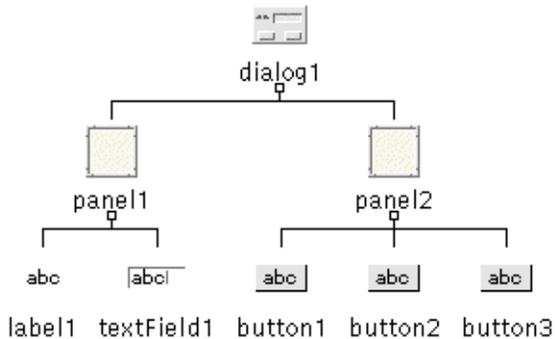
Selecting “Java Console...” displays a simple window containing a read-only text area. This window displays any exceptions which have occurred while you are using Visaj. You only need to check here if you feel that Visaj is not responding correctly. This window is for the display of information only.

The “Beans” page of the method editing panel in the Class Editor, shown in Figure 5-2, is the tool for building the user interface of your application. User interfaces in Java are built hierarchically with AWT components grouped inside containers and containers inside a window or dialog. Each container controls the way its children are laid out and behave when resized. Figure 5-1 shows how Java components may be organised in a simple dialog. The topmost view is the dialog as it appears on your screen, in the middle is a tree view showing how you would build up the hierarchy of AWT components and the view at the bottom is a schematic, theoretical diagram of the containments.

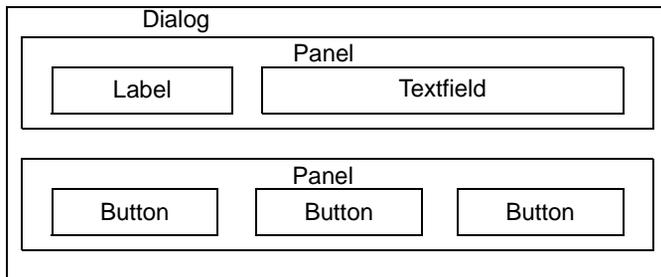
Note – *The schematic view is for clarification only - Visaj does not display this view.*



Dialog on the screen



Hierarchical view



Schematic view

Figure 5-1 Java Containment Hierarchies

The graphical editor for containment hierarchies is found on the left of the Class Editor. Each method in your class can have a containment hierarchy defined for it. Select a method from the class structure view and check that the panel labelled “Beans” is visible on the left. This panel is a graphical tool for building user interfaces.

There are two main areas: the containment hierarchy and the object palette. The design area is the place where containment hierarchies are built and invisible beans are displayed. The object palette displays all objects available to you for building up your user interface. This is described in more detail in the *Object Palette* section on page 59. The *Opening Other Palette Files* section on page 59 and the *Loading JAR Files* section on page 60 describe two ways of adding objects to the palette.

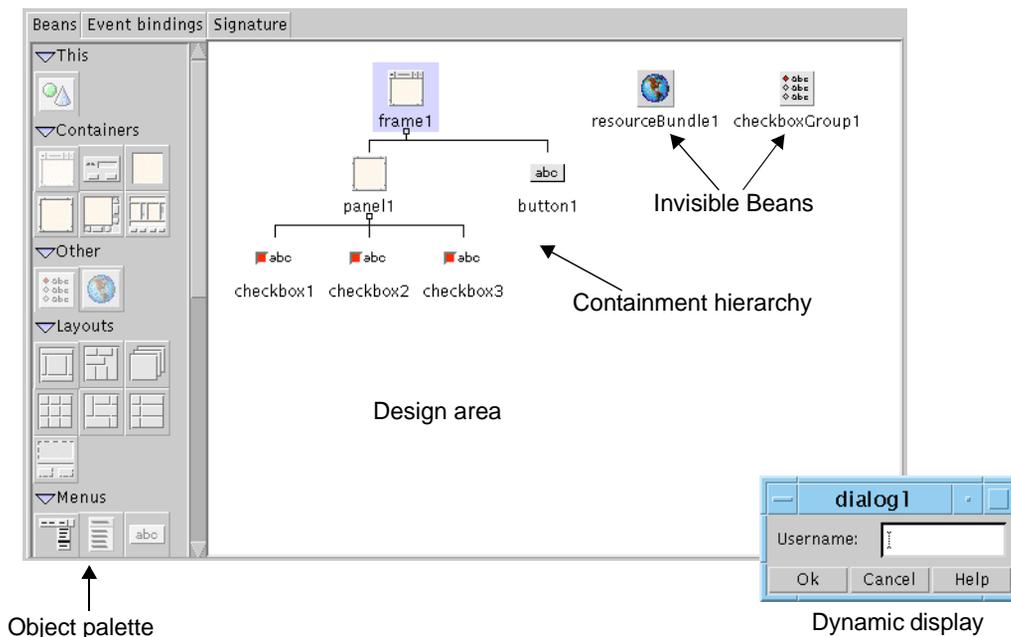


Figure 5-2 Beans View

5.1 Dynamic Display

The dynamic display is a preview of the window you are designing, showing how it will look when the code is generated, compiled and run. The dynamic display is a floating window, as shown in Figure 5-2.

The dynamic display and the containment hierarchy can be recreated by pressing the Reset button on the toolbar (or selecting “Reset” from the Object menu). The selected object and its children are rebuilt using any properties you have set for them. If no objects are selected in the design area, the whole hierarchy is rebuilt. Use this option when you have made many settings to ensure that the dynamic display accurately reflects your design.

5.2 *Dummy Frames*

If the root of your hierarchy is not a window capable of being displayed on your screen, Visaj will place the dynamic display for your hierarchy in a suitable window with the title “Dummy Frame”. This is useful if you want to create a method which creates a component or group of components - for example, a group of labels and text fields which you wish to use again and again in your application. Or you may have a Choice component which you have loaded with a list of color names which you wish to use in multiple dialogs; making the Choice a hierarchy on its own would allow you to do this.

5.3 *Building Hierarchies*

To create containment hierarchies, click over the objects in the object palette. If no objects are selected in the design area, the new object becomes the *root* of a hierarchy. Visaj assists you in creating a viable hierarchy by only allowing you to select those objects which can be made children of the selected object; all others are made insensitive.

Note – *With the Swing component set, all components can take children.*

Objects are added as children of the current selection. As objects are added, they are automatically selected if they are themselves containers. If not, the parent remains selected. With the Swing set, Visaj only selects containers automatically even though all components may take children.

To start another hierarchy in the same method, make sure that no objects are selected in the design area and then select the new root object from the object palette. Clicking on the design area background deselects all objects.

Select an object in the hierarchy by clicking over it. To select more than one object, drag a rectangle over them or hold down the Shift key to add to the selection list.

5.4 Object Palette

Objects are divided into groups on the palette. Each group is labelled. The group label is a toggle button which, when selected, folds away or unfolds that group.

By default, only the icon representing the object is shown on the palette. By selecting “Show labels” from the Palette menu, you can also see the name of the object.

Invisible Java beans also appear on the object palette. When you add them to the method they appear at the top of the design area.

There are two ways of adding objects to the palette: opening another palette file and loading a JAR file. You also have the option of changing the entire palette you are using. All of these are described below.

5.4.1 Opening Other Palette Files

Add objects to the palette by selecting “Merge palette file...” from the Palette menu. You are prompted for the name and location of the pre-defined palette file. The format of a palette is described in the *Palette File* section on page 162. For Visaj to find the objects listed in the palette file, make sure that you have set your CLASSPATH environment variable¹ to give the location of the separate class files or the JAR file containing the classes.

Taking the trouble to define a palette file gives you more flexibility over the grouping of objects when they appear on the palette and over which objects are loaded from a JAR file.

1. Environment variables are available on UNIX and Microsoft Windows. Other platforms, such as the Apple macintosh, use their own method of setting a CLASSPATH. Please refer to the relevant Java documentation for more information.

If you have saved a design which uses a bean from a merged palette, Visaj tries to load the bean for you automatically when you next open that design.

5.4.2 *Loading JAR Files*

Add objects to the palette by selecting “Load Jar file...” from the Palette menu and giving the name and location of a JAR file in the File Dialog. Visaj adds all beans in the JAR file to the palette, grouping them under one heading which is the name of the JAR file. You do not need to set your CLASSPATH environment variable¹ for this option. Although this is often easier than creating palette files, you cannot control which objects are loaded as Visaj will load in all the beans it finds in the file and they are all grouped together.

If you have saved a design which uses a bean from a loaded JAR, Visaj tries to load the JAR for you automatically when you next open that design.

5.4.3 *Change Palette*

The Palette menu contains a pullright menu, labelled “Change”, with two items: “AWT” and “Swing”. Selecting these options changes your current palette to the AWT components or the Swing set respectively. See Chapter 7, “Swing Component Set” for information on using Swing.

Using this method, the palette is *changed* and not *merged* with existing components. Use “Merge palette file...” if you wish to merge the components on your palette with AWT or Swing components.

In order to change to the Swing palette, you must have the Swing classes in your CLASSPATH. See the *Loading Swing Components* section on page 90 for more details.

5.5 *Properties*

All objects have properties. These include, for example, the colors, dimensions, label and visibility of the object. Many (sometimes all) of these properties are inherited from an object’s superclass. To edit the properties

of an object, either double-click the object in the hierarchy area or select the object and select Properties from the Object menu or from the toolbar. This causes the Property Sheet to be displayed, as shown in Figure 5-3.

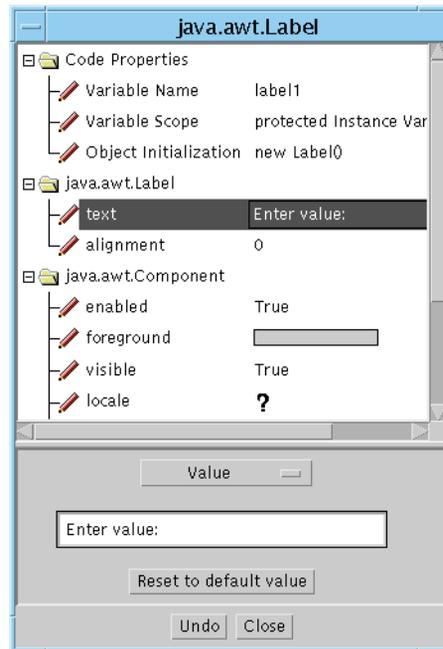


Figure 5-3 Property Sheet for a Label

The Property Sheet is organized in the form of a tree, with each node indicating from which superclass the properties have come. Each node in the tree can be folded or unfolded. This is most useful where there are a large number of properties which can be changed. Next to each property there is a pencil icon which indicates whether or not the property is editable. If the pencil is “crossed out” (has a line across it), it cannot be edited.

To edit a property, select it in the tree. The lower area of the Property Sheet displays the options available for that property or a means of entering a value. Properties are set as soon as they are changed. They take immediate effect in the dynamic display.

The “Undo” button at the bottom of the property sheet undoes the last change made in the current property sheet.

Note – See the *Packing Frames* section on page 153 for special information concerning the size property for *Frame* and *JFrame* components.

5.5.1 Variable Name

The Property Sheet contains a field at the top of the window labelled “Variable Name”. Although this is not a true property in the same sense as the other properties in this window, it is a value associated with the selected object. The variable name is the name used to refer to the object in the generated code.

5.5.2 Variable Scope

The field at the top of the Property Sheet labelled “Variable Scope” provides a means of controlling the declaration scope of an object. When this field is selected, two option menus appear in the editing area at the bottom of the Property Sheet. One of the menus allows you to choose whether the object is an “Instance” or a “Local” variable. This affects where the object is declared; instance variables are declared as members of the Class, local variables are declared in the current method and are therefore not accessible from the rest of the Class. The other option menu only applies when you have chosen “Instance variable” and refers to the accessibility of it. There are four options:

1. **Default.** This is the default assigned by Java when none of the other options are specified. It means that the variable is only accessible from the current class and other classes in the same package. Even subclasses, when they are not in the same package, would not be able to access them.
2. **public.** This makes the variable visible everywhere.
3. **private.** The variable is only accessible within the class in which it has been declared.
4. **protected.** When “protected”, the variable is visible within the current class, all subclasses and all classes in the same package.

5.5.3 *Object Initialization*

This field in the Property Sheet enables you to control the way in which any object in your design is initialized. By default, a new instance of the object is created. You may change this to one of the following:

1. New
2. Beans instantiation
3. Code Expression
4. Variable Name
5. Deserialization
6. No Initialization

Each of these is described in the following sub-sections.

New

This is the default for non-serialized beans. A new instance of the object is created. This option is not present for serialized beans which you have loaded onto your palette as there is no new method for them.

Beans instantiation

Select this option to instantiate the bean. A call to `Beans.instantiate` is generated, passing in the name of the bean.

Code Expression

You may type any arbitrary expression for the initialization of the object. Use this option when you wish to initialize the object from a routine in another part of your application, for example.

Variable Name

Typing the name of a variable into the text field will cause Visaj, when requested, to generate the code to assign that variable to your object. Visaj simply generates the assignment. You must check that you have typed the name correctly and that the variable is in scope.

Deserialization

If this is the selected option, the object is created by being deserialized out of the serialization file specified in the text field. The serialization file, whose name you provide here, is generated by Visaj when the Java class files are generated. This is described in the *What is Generated* section on page 152. Visaj provides a default name for the serialization file based on the variable name of the object.

This option is the default for any serialized beans which you may have loaded onto your palette.

Visaj automatically sets the “Object Initialization” property to “Deserialization” when you customize a bean. If you subsequently change this option, the customization will not take effect in the generated code. More about customizing beans is given in the *Customizers* section on page 68.

No Initialization

The object will not be initialized in the generated code.

5.5.4 Code Expression, Value and Variable Name

When editing normal properties (that is, properties other than those listed under “Code Properties”), the lower editing area displays an option menu containing the items “Value”, “Code expression” and “Variable Name”. Choosing either “Code Expression” or “Variable Name” causes a text field to be displayed where you may type in either any expression, which is then used to set the selected property, or the name of a variable which is in scope. For example, you may wish to call a method which returns a property value, or you may wish to create a new instance of a class to assign to the property. These expressions are not used dynamically inside

Visaj, but you will see its effect when the generated code is run. One example of the use of the “Variable Name” option is given in the *Using CheckboxGroups* section on page 76.

5.5.5 *Reset to Default Value*

At the bottom of the property sheet there is a button labelled “Reset to default value”. This is enabled if you have changed the selected value in the property sheet and, once pressed, sets the value back to the default.

5.5.6 *Setting Color and Font Properties*

To set a color or font property using the Color or Font Selector, drag the required color or font from the sample window of the Selector either into the editing area at the bottom of the property sheet when an appropriate property is selected or directly onto the property value where it is displayed in the property sheet. The Color and Font Selectors are available from the Windows menu of the main Visaj window. For more details on using the Selectors, see the *Font Selector* section on page 52 and the *Color Selector* section on page 53.

5.5.7 *Default Labels*

Objects such as Buttons, Labels and Checkboxes are given a unique default label based on the object name and a number, as shown in Figure 5-4. Visaj does this so that the object displays properly in the dynamic display; without a label, the object would shrink very small and, in the case of Labels, would not be visible at all.

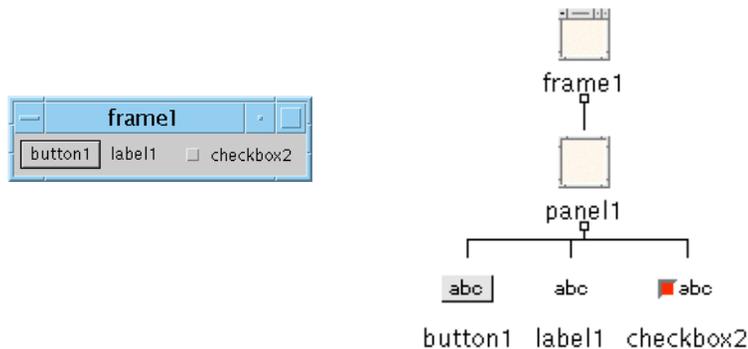


Figure 5-4 Default Labels Assigned by Visaj

If you do not set your own labels using the Property Sheet, the default labels are generated into the code so that the final application looks exactly as you see in Visaj.

5.5.8 Image Properties

Many objects have image properties. The Frame, for example, has “iconImage”, which is the image it displays when iconized. When this type of property is selected, the property editor shown in Figure 5-5 appears at the bottom of the property sheet.



Figure 5-5 Image Property Editor

The option menu at the top of this section lets you choose the standard types of value for a property. Choosing “Image resource” (and for Swing “Swing ImageIcon resource”) lets you provide an image file to load immediately and the location of it at runtime.

Runtime Resource Path

The “Runtime resource path” is the name of the image, optionally including its package, to be used when the generated code is run.

In the generated code, image files are loaded as resources using the class loader. The string typed into the “Runtime resource path” text field is passed directly to the class loader method `getResource`. See your Java documentation for a detailed description of the mechanism used by `getResource`. The *Books on Java* section on page 209 lists some suggested books.

If, at runtime, your image cannot be found a `NullPointerException` is displayed. Check that you have entered the correct package name in Visaj and that your `CLASSPATH` is set correctly.

Design Time File

When you press apply, Visaj tries to load the image specified as “Design-time file” and displays it in the property sheet. If the image cannot be located, an error dialog is displayed.

5.5.9 *Multiple Selection*

You may select multiple objects by either dragging a rectangle around them or by holding down the Shift key while selecting to add to the selection. If more than one object is selected, the property sheet displays only those properties which are common to all the selected objects.

5.5.10 Layout Properties

Some properties affect the layout manager, which is itself a property of an object. One example of such properties is the number of rows and columns in a Grid. To set the properties of a layout manager, press the Properties icon in the toolbar of the Layout Editor. Layout Editors are described in the *Layout Editors* section on page 69.

5.6 Customizers

Objects which are Java Beans are often provided with a *Customizer*. This is a means of setting properties of the bean in addition to those available from the Property Sheet. See the *Properties* section on page 60 for more information on Property Sheets. If a customizer is available for an object, the “Customize...” option in the Object menu is sensitive. An example of such a dialog is shown in Figure 5-6.

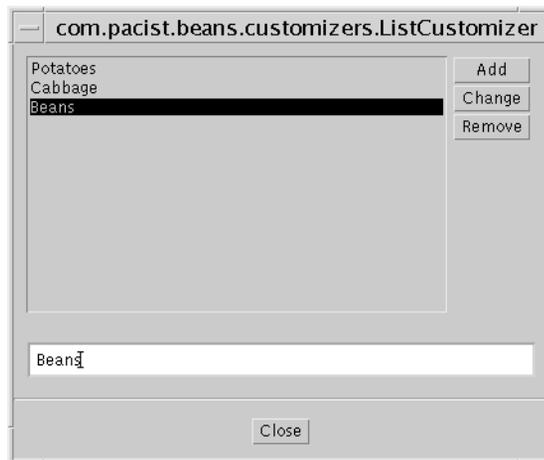


Figure 5-6 Customizer

☞ Customizers provide the means of populating List and Choice components.

When you customize a bean, Visaj changes its “Object Initialization” property to “Deserialization”. If you change this property back to “New”, your customization will not take effect in the generated code. Changing the “Object Initialization” property is described in the *Object Initialization* section on page 63.

5.7 Layout Editors

All components which are containers (Frame, Panel etc.) have a *layout* associated with them which controls how the children of the container are arranged. To find out how to apply a particular layout to a container, see the *Using and Applying Layouts* section on page 74.

There are a number of different types of layout which are part of the Java AWT and more supplied by individual component vendors. These layouts allow you to arrange the objects within the container in a specified way. To do this dynamically, use Visaj’s Layout Editor. Select the Layout icon from the toolbar when a container is selected in the hierarchy. The Layout Editor is displayed in its own window with layout functions appropriate to the type of layout set on the container component. Changes that are made in the Layout Editor are reflected immediately in the dynamic display.

The layout types supported by default are:

1. GridBag (from the Java AWT package)
2. Grid (from the Java AWT package)
3. SuperGrid (from the IST Diamonds)
4. Flow (from the Java AWT package)
5. Border (from the Java AWT package)
6. BorderLayout (from the Swing component set)
7. Null (no layout)

These are described individually below.

Note – *To undo and redo actions in the Layout Editor, use the Undo or Redo buttons in the Class Editor to which it belongs.*

5.7.1 *Grid*

Components may be swapped around within the grid by dragging them over one another but the number of rows and columns, along with the horizontal and vertical spacing is set from the Property Sheet. Do this by pressing the property sheet icon in the toolbar of the Layout Editor. Since the order of the components in the containment hierarchy must be the same as the order in the Grid layout, when objects are re-ordered in the Layout Editor their order is automatically changed to reflect this in the hierarchy.

5.7.2 *SuperGrid*

A Layout Editor for the SuperGrid Diamond is also supplied. The SuperGrid Layout Editor is very similar to that of the Grid, allowing you to re-order objects within a grid. The order of components in the containment hierarchy dictates the order in which they appear in the Layout Editor. In order to retain this parallel (so that your generated code will act exactly as Visaj shows), the ordering of objects in the hierarchy is updated as objects are re-ordered in the Layout Editor. Properties such as the number of rows and columns are edited in the Property Sheet, as described for the Grid Layout Editor. For more information on the SuperGrid layout, see the *SuperGrid Layout* section on page 195.

5.7.3 *Flow*

The Flow Layout Editor is also similar to that of the Grid, except that it is effectively one long row which is wrapped around according to the size of the containing window. Children can be re-ordered by dragging them over one another. The Layout Editor simply displays one long row; the dynamic display shows the wrap-around of this row, as you resize the window.

5.7.4 *Border*

The Border Layout Editor shows the geographical positions possible (North, West, Center, East and South) and allows you to drag components into these positions. If a component already occupies the position, the two components are swapped.

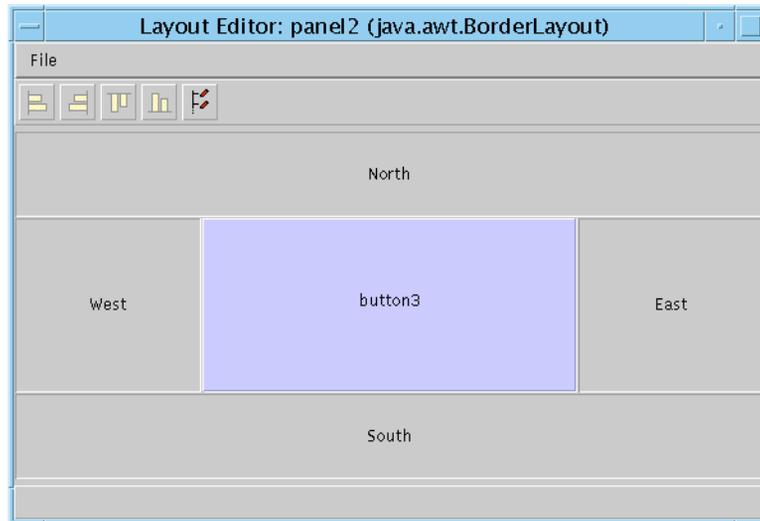


Figure 5-7 Border Layout Editor

By default, a Frame component is given a Border layout and a Panel is given a Flow layout.

5.7.5 Null Layout (*Absolute Positioning*)

Using the Property Sheet you may choose “Null” for the layout of a container. This means that the container has no layout manager associated with it. This is useful if you wish to use *absolute positioning* to lay out the children of the container. Although there is no layout manager, the Layout Editor still allows you to move and resize the child components.

Use the four alignment buttons in the toolbar of the layout editor, shown in Figure 5-8, to ensure that your objects are aligned neatly. The buttons become sensitive when more than one object is selected. The objects are aligned to the last selection.

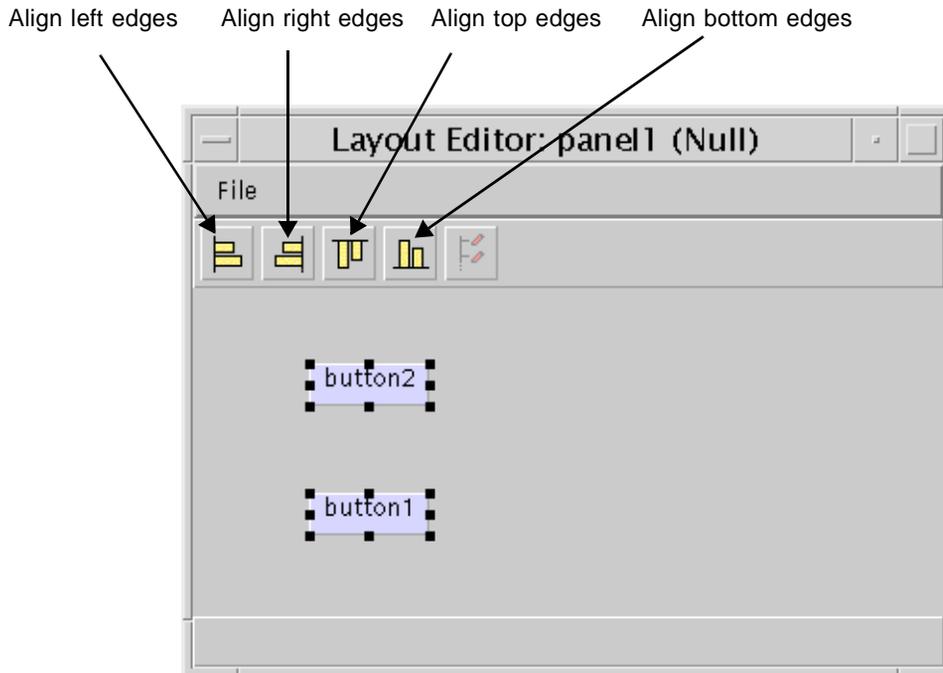


Figure 5-8 Alignment Buttons in Layout Editor

5.7.6 GridBag

The GridBag layout is the most flexible and most complicated of the layout types.

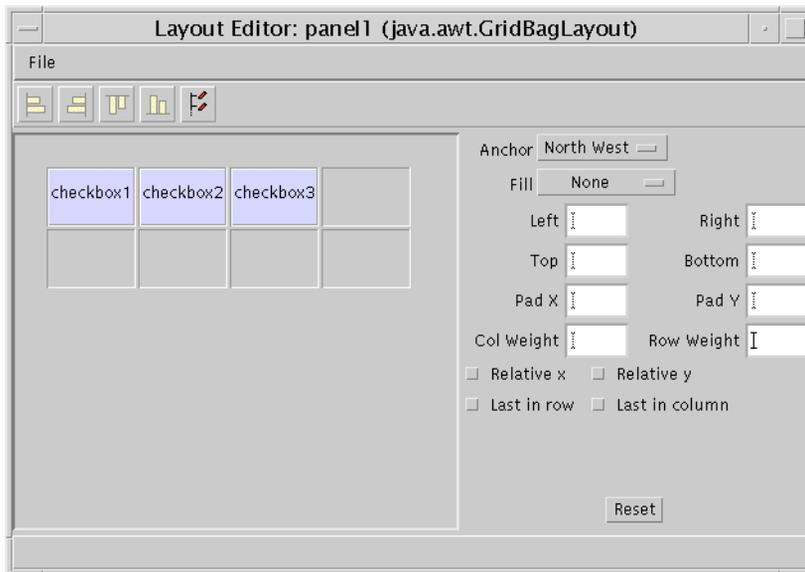


Figure 5-9 GridBag Layout Editor

The alignment buttons in the toolbar of the GridBag Layout Editor work exactly as described for the Null layout, above.

The area on the right of the editor window allows you to edit the GridBag Constraints for a particular component in the GridBag. Select the component in the grid area by pressing the mouse button with the cursor over it. Edit the constraints in the text boxes on the right. The constraints are applied to the current selection as you change the focus from one text box to another.

The left area of the Layout Editor is a logical representation of the components within the GridBag. The components are not shown actual size; this makes moving and resizing easier for small components. Components can be moved around inside the GridBag and, as you do this, the GridBag itself will change its dimensions correspondingly. Components can also occupy more than one cell in the grid. Do this by selecting the component and resizing it using the handles which appear around the edge. All of these changes to the GridBag layout are effective immediately in the dynamic display.

The GridBag layout allows you position components anywhere in an arbitrary grid. This means that it is possible to have empty rows and columns between two components. The GridBag sees empty rows and columns as having zero width and height, since the width and height of a row or column is calculated by its contents. The GridBag Layout Editor indicates that a row or column has a zero width and height by displaying them in a different color (pink). Since the dynamic display shows the actual sizes of the components, the empty rows and columns appear not to be there.

5.7.7 *BoxLayout*

The Swing component set includes `BoxLayout`. This is a layout manager which places each of its children from left to right or from top to bottom, according to the parameter passed in when the `BoxLayout` is created. Components are arranged from left to right (or top to bottom), in the same order as they were added to the container. `Visaj` lets you use the `BoxLayout` via the `Box` class. `Box` is a container which uses `BoxLayout`. A Layout Editor is provided for `BoxLayout`. All of this is described in the *Swing Tips* section on page 95.

5.8 *Using and Applying Layouts*

Layouts are selectable from the palette. When a layout is selected from the palette, it is applied to the layout property of the currently selected container. Change the layout of a container by either:

1. Selecting another layout from the palette with the container selected in the hierarchy.

or:

2. Change the layout property in the container's Property Sheet. See the *Properties* section on page 60 for details on how to do this.

5.9 *Invisible Beans*

Java Beans are reusable software components. Some have their own visual representation, some do not. Invisible Beans are Beans which have no visibility to the user - for example, a database access component. These

Beans can be added to your palette in the same way as any other component is added - see the *Palette File* section on page 162 for more details on adding components to your hierarchy.

In order to add an invisible Bean to your design, make sure there is no current selection. Do this by clicking over the background of the design area. The invisible Beans are then displayed at the top of the design area, alongside the roots of hierarchies, as shown in Figure 5-10.

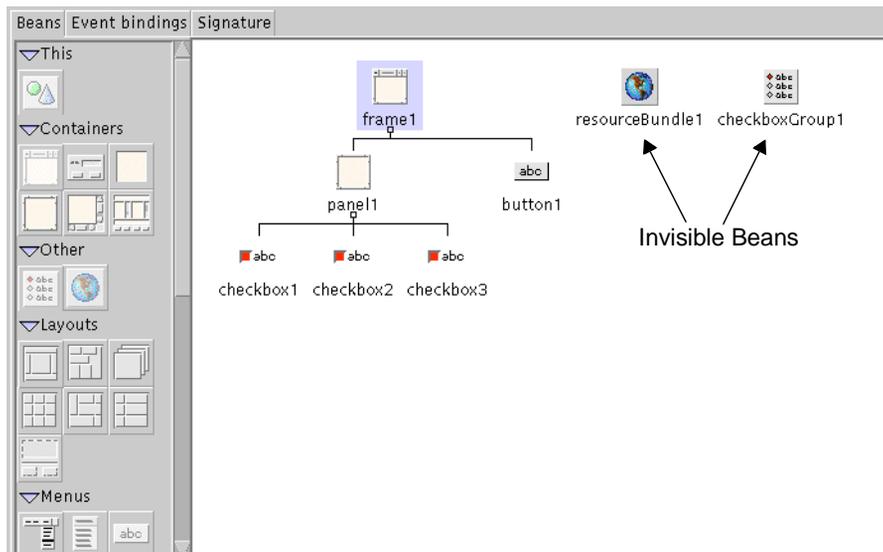


Figure 5-10 Invisible Bean Area

Once you have added an invisible Bean to your method, it will appear in the Event Editor, allowing you to link it in to your containment hierarchy. You may add any number of invisible Beans to your method.

Like the other objects on the palette, invisible Beans have properties editable from the Property Sheet and, if provided by the vendor, can also have customizers.

Two common types of invisible bean available on the default palette are `CheckboxGroups` and `ResourceBundles`. How to use `ResourceBundles` is described in the *Using Resource Bundles* section on page 144. `CheckboxGroups` are described below.

5.9.1 *Using CheckboxGroups*

CheckboxGroups provide a way of grouping Checkboxes to give them radio button behavior. As with all invisible Beans, make sure you have no current selection before adding a CheckboxGroup to your design. It then appears at the top of the design area. To apply a CheckboxGroup to a Checkbox, do the following:

1. Display the property sheet for the Checkbox (remember that you can select more than one Checkbox and apply any changes to all of them).
2. Select the checkboxGroup property.
3. Select the “Variable name” option from the editor at the bottom of the property sheet.
4. Type in the name of the CheckboxGroup you wish to apply to the Checkbox.

Although you will not see the radio behavior in the dynamic display, the generated code will display such behavior.

5.10 *Using Your Beans - Creating Reusable Components*

The Beans view in Visaj allows you to create standalone applications, applets and beans. Creating beans from your designs provides a way of reusing components. To generate a bean from your design, you do not need to do anything special in Visaj. Simply generate code as usual, having observed the following two points:

1. The Class must be public.
2. The Class must provide a default constructor with no parameters. For example:

```
public MyClass()
```

After you have generated code you will either need to create a JAR file containing your bean or, if you only wish to use the bean in Visaj, create a palette file which references your bean. Each of these options is described in the following sub-sections.

5.10.1 Putting the Bean in a JAR

If you are not using an IDE which creates beans from Java sources, you will have to do this yourself. The following description tells you how.

Once you have generated code for a design which you intend to use as a bean, compile it and then create the manifest file. This is a special file that contains information about the files packaged in a JAR. The "jar" command automatically creates a default manifest file but this default does not contain the vital information that the classes the contents of this JAR constitute a bean. Therefore, you will have to create the manifest file yourself.

The manifest file must be called "MANIFEST.MF" and contains a list of all the .class files in the JAR. must also be stored inside a directory or folder named "META-INF". Here is a simple example of the contents of a JAR file containing a bean whose class name is "MyClass":

```
META-INF/
META-INF/MANIFEST.MF
MyClass.class
```

To see the contents of a JAR, use the following command:

```
jar tvf filename.jar
```

The contents of the file META-INF/MANIFEST.MF are fairly simple. It just needs to list the name of the main class for the bean and indicate that this is indeed a bean. Write the following into your manifest file, replacing "MyClass.class" with the name of the main class of your bean:

```
Name: MyClass.class
Java-Bean: True
```

Now you need to package your files (the class file(s) for your bean and the manifest file) into a JAR. The following command does this:

```
jar cvfm myjar.jar META-INF/MANIFEST.MF class-files
```

"cvfm" stands for "c"reate, "v"erbose, "f"ile name, name of "m"anifest file.

The command line option "f" and "m" must be specified in the same order in which the corresponding files are specified.

The resulting JAR file is your bean. See the *Loading JAR Files* section on page 60 for details on how to load your bean onto Visaj's palette. Once loaded onto your palette you can use it in your designs. Remember that if you save any designs which use your new bean and then exit Visaj, you will need to reload the bean's JAR file before opening those saved designs.

5.10.2 *Creating a Palette File to Reference the Bean*

If you simply wish to use your newly created bean in Visaj, you can create a palette file for the bean and then use "Merge palette..." to add the bean into your existing palette. The bean can then be used inside Visaj. Remember that if you do this and save a design which uses the new bean, you will have to merge in the bean palette file before opening the saved file. See the *Palette File* section on page 162 for a description of palette files. Before creating the palette file, generate Java for your bean and compile the generated code.

6.1 Introduction

All objects within your Visaj design can be linked together using events. Events are messages which can be sent by the action of one component and may be received by the method of another. You can specify that a particular method in a specific component is called when an action (such as a button press) occurs in one component. Once you have created an event binding in Visaj, it is immediately active in the dynamic display. You can create as many bindings as you wish and try them out straightaway.

The Event Binding Editor, described in the following section, lets you create new bindings and edit existing ones. This is a wizard which can be displayed in one of the following ways:

1. Pressing the  icon on the toolbar.
2. Holding down the Alt or Control key while dragging from the source to the destination component.
3. Choosing “Add event binding” from the Method menu.
4. Pressing either “New” or “Edit” beneath the Event bindings list.

Event bindings, once created, are listed on the Event bindings tab of the method editing area. Here they may be edited, re-ordered or deleted. The Event bindings list is described in the *Event Bindings List* section on page 86.

6.2 Event Binding Editor

The Event Binding Editor, shown in Figure 6-1, is where new bindings are created and existing ones are edited.

6.2.1 Displaying the Editor

There are five ways to display the Event Binding Editor:

1. With the Alt or Control key held down, drag the mouse pointer from one component to another in the containment hierarchy. The Event Binding Editor then appears, primed with the first and second component as the source and destination of the binding respectively. The first event type in the list is also selected. This method can only be used to create *new* bindings.
2. Press the Event Binding Editor button  on the toolbar. The Editor is ready for you to create a new binding so it does not pre-select anything, unless a component is selected on the Beans page. The component is then used as the Source Object.
3. Press “New” in the Event Bindings List. The Event Binding Editor is not primed, except to show all the available components.
4. Press “Edit” in the Event Bindings List. You will need to select a binding from the Event Bindings List first.
5. Select the “Add event binding” item in the Method menu. This is exactly the same as pressing the corresponding toolbar button.

6.2.2 Description of the Editor

The event binding editor shows all the components in your design, all the event types associated with the selected source component and all methods available for the selected destination component. You must make a selection in the “Source” and “Object” lists for the “Type” and “Method” lists to appear.

There are usually a large number of methods for a particular handler. To quickly locate the one you require, select the Method pane (by selecting any method within it) and type the first letters of the method you are looking for. The display scrolls to methods beginning with the letters you type.

Some handler methods may be capable of throwing an exception. This is not indicated in the event binding editor. However, if you add an event binding with one of these methods, Visaj adds a method to your class which handles the exception when thrown. The method is displayed in the class structure editor on the right of the Visaj window but its signature cannot be changed. In the generated code, this method gives you a place to add your own code for handling exceptions.

Assignment

The first item in the Method list is not a method at all. This item, shown as “= [expression]”, allows you to assign a property or method return value to the “Object” when an event is fired. The “Next” button becomes enabled when the assignment operator is selected. Press this button to display the parameters page of the editor - described in more detail in the *Event Binding Editor: Parameters* section on page 83. You should use the parameters page exactly as described even though you are specifying the source of an assignment rather than a method parameter. Remember that, as with parameters, you can only select a method return value or property which is the appropriate type. You may set a “Value” if there is an editor available for the selected type.

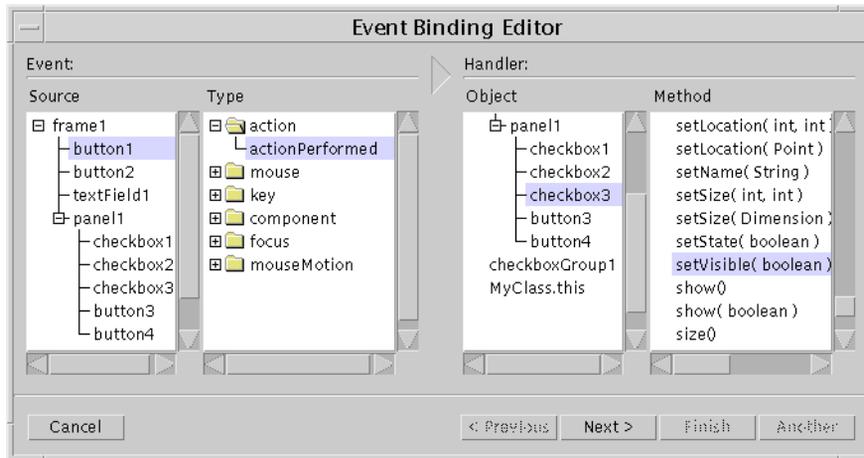


Figure 6-1 Event Binding Editor

At the bottom of the Event Binding Editor there are five buttons: “Cancel”, “<Previous”, “Next>”, “Finish” and “Another”. The Finish and Another buttons become sensitive only when you have entered enough information for the binding to be created. The only difference between these two buttons is that pressing Finish closes the dialog whereas pressing Another leaves the dialog open on the screen.

When you select a destination method, the Next button becomes sensitive if the method has parameters. Pressing the Next button (or the Return key) displays the second page of the Event Binding Editor, as shown in Figure 6-2.

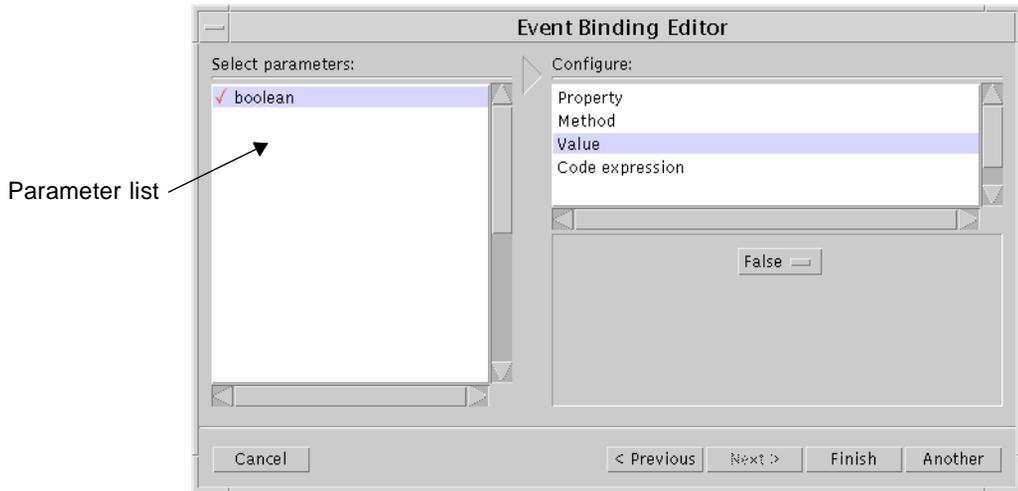


Figure 6-2 Parameters Page of Event Binding Editor

6.3 Event Binding Editor: Parameters

The second page of the Event Binding Editor allows you to specify values for the parameters to the destination method. The parameters are listed by their type on the left of the dialog. Each parameter is shown with a cross or tick next to it:

- ✓ A value has been set for this parameter
- ✗ No value has been set for this parameter

The right of the dialog contains two areas. The top area lists the ways of specifying the value of the selected parameter. The lower area contains the editor for the selected value type. The four value types are:

1. **Property**
2. **Method**
3. **Value**
4. **Code expression**

The following sub-sections describes each of the above.

6.3.1 Property

This allows you to set the value of the parameter to the value of a component's property. When "Property" is selected, two scrolling lists appear in the lower area. This is shown in Figure 6-3.

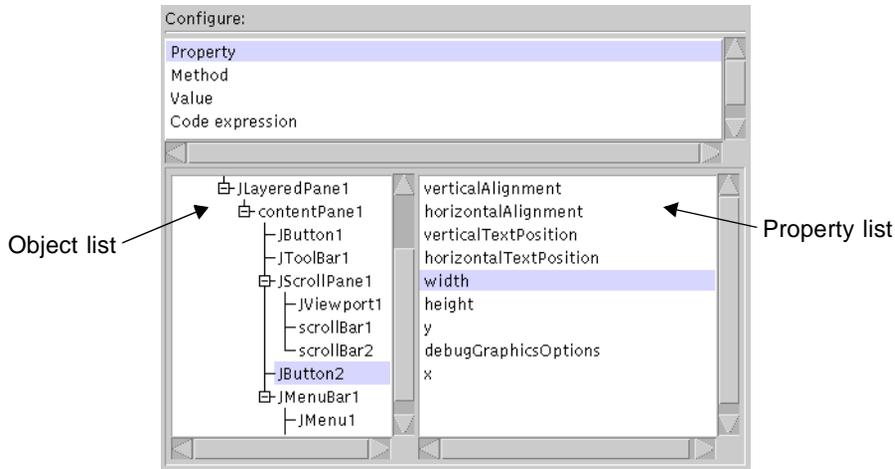


Figure 6-3 Property Value for Parameter

The list on the left shows all the objects in your design, including the event object being passed to the listener method. Selecting one of these objects causes all the properties of this object to be displayed in the list on the right if, and only if, it is assignable to the selected parameter.

You can assign any property to a parameter of type String as all objects are converted to Strings by way of their "toString" method. Properties which are going to be coerced in this way are shown in blue type.

The property list includes "this" if the object selected from the object list can be assigned to the selected parameter.

If the object selected from the object list has no properties of the appropriate type, the message "No properties of an appropriate type" is shown in the property list.

This type of parameter value is reflected in the dynamic display as soon as the binding has been completed, except when event objects have been selected from the object list.

6.3.2 Method

This option allows you to use the return value of another method in your design. When you select “Method”, a text field, a button labelled “Edit” and the Method selection dialog appears. The Method Selector is shown in Figure 6-4. When you have finished selecting a method it is shown in the text field. To display the Method Selector in order to edit a previous selection, press the “Edit” button.



Figure 6-4 Method Selection Dialog

The left side of this dialog lists all the objects in your design. When you select an object, the area on the right lists object methods which return the same type as the selected parameter. For example, if you have selected a parameter of type “int”, only methods which return an integer are listed. One of the objects which can be selected is “this” - the class currently being designed. If you have added a method to your class with a compatible return type, you will be able to select it as the method for the event binding.

If the selected method has parameters, the Next button is enabled. Pressing Next displays the parameters page of the Method selection dialog - this is identical to the parameters page of the Event binding editor and is used in exactly the same way.

Press Finish when you have selected a method or Cancel if you wish to close the dialog without selecting a method. You cannot continue working on your event binding until the Method selection dialog has been closed. The selected method is shown in the text field of the parameters page.

6.3.3 *Value*

This is an explicit value. When this is selected, a property editor appears in the lower area for you to set the value. If the property editor is a text field, you must press the Return key after typing the text to set the value. These values are used in the dynamic display once the binding has been completed.

6.3.4 *Code Expression*

This allows you to type in an arbitrary value or expression for the parameter. Bindings which use these are not active in the dynamic display. A text box appears in the lower area for you to enter the code expression. You do not need to press the return key at the end of the text.

6.4 *Event Bindings List*

The Event Bindings List, shown in Figure 6-5, makes up part of the method editing tools in the Class Editor. To view the Events Binding List, select a method from the class structure view and choose the “Event bindings” pane from the tabbed panel. Until you define a binding, this list is empty.

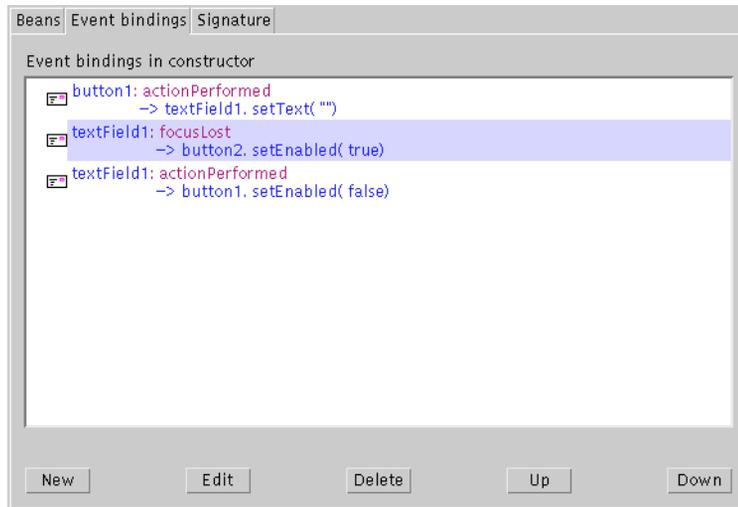


Figure 6-5 Currently Defined Event Bindings

Press “New” to populate this list, “Edit” to change the selected binding and “Delete” to remove it from the list.

6.4.1 Order of List

If you have more than one method bound to a component’s event, they are grouped together. Use the “Up” and “Down” buttons to re-order the methods within a particular component/event group. This order is significant because this is the order in which the event bindings are executed when the application runs. The generated code always reflects the order presented in this list. To make sure that the dynamic display also reflects any changes you may have made, rebuild the containment hierarchy by pressing the Reset button.

6.5 Invalid Bindings

An event binding is marked as “INVALID” if the source or destination object of a binding has been removed from the design after the binding was created. The invalid part of the binding is shown in red. An example invalid binding is shown in Figure 6-6.

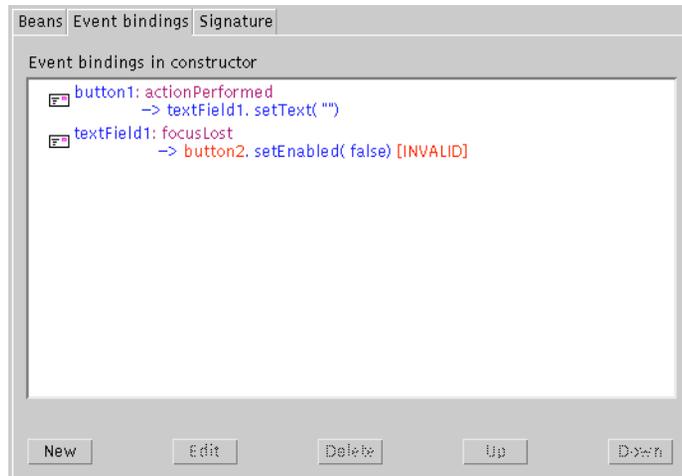


Figure 6-6 Invalid Event Binding

The binding remains intact, so if you paste the lost object back into the design (or press “Undo”), the binding is automatically revalidated.

7.1 Introduction

JFC (Java Foundation Classes), compatible with the JDK, includes the Swing components. For more detailed information on JFC and what it includes, try the following website:

<http://www.javasoft.com/products/jfc>

The Swing component set is a graphical user-interface (GUI) toolkit which allows you to design your graphical side of your application in such a way that it has the “look and feel” of the computer on which it is running *without modification*.

For example, when you create a user interface with Swing and run it under Windows, it has the appearance and behavior of a program written specifically for Windows. When you run the same program on a UNIX workstation, it runs just like any program written for UNIX. Equally, when you run it on an Apple Macintosh, it looks and behaves just like any program written specifically for the Mac. This “pluggable look and feel (plaf)” is possible because Swing components do not use any native code.

The Swing component set extends the AWT, but does not replace it. Swing components are called *lightweight* because they do not use any platform-specific implementations, such as the “peers” in AWT.

7.2 AWT to Swing Conversion

Using its SwingBridge™ technology, Visaj offers you fast, automatic conversion of your AWT designs to Swing. Simply select “Swing” from the “Translate” pullright menu in the File menu. Your converted design is loaded into a new window and can be used in exactly the same way as any other design in Visaj.

In order to use SwingBridge, you must make sure that Swing is in your CLASSPATH before starting Visaj. For details on how to do this, see the following section, *Loading Swing Components*.

7.3 Loading Swing Components

If you are using a JDK earlier than Java 2 (or JDK 1.2), there are two steps to follow in order to use the Swing components in Visaj. JFC is included in Java 2, so you do not need to set your CLASSPATH first - just follow the second step:

1. Make sure the Swing jar file is in your CLASSPATH:

- UNIX users can set the environment variable in the usual way, for example:

```
CLASSPATH=/swing/swingall.jar
```

- Windows users can do this by either putting the following line (modified to contain the true location of your Swing jar files) into

```
C:\AUTOEXEC.BAT:
```

```
set CLASSPATH=c:\swing\swingall.jar;%CLASSPATH%
```

or by running from the Start menu:

```
“Start->Settings->Control Panel->System”
```

and then clicking on the Environment Tab.

Remember that you do not need this step if you are using Java 2.

Note – *On Windows, you may need to reboot your machine to allow the modified environment to be used.*

2. Run Visaj with the -swing argument:

Visaj -swing

When you start Visaj in this way, your palette does not contain the AWT components. If you wish to use them, load in the palette file `AWT.palette` in `$VISAJ/palettes` (where `$VISAJ` is your Visaj install directory).

If you wish to start Visaj in the usual way, without specifying “-swing”, run Visaj *after having set your CLASSPATH* and do the following:

1. Select “Merge palette file...” from the Palette menu.

This displays a File Selection dialog prompting you for the name and location of the palette file.

2. From `$VISAJ/palettes` (where `$VISAJ` is your Visaj install directory), select the palette appropriate to the version of Swing you are using.

The possible palettes are:

`JFC10.palette` - for Swing 1.0.3

`JFC11.palette` - for versions of Swing earlier than 1.1 beta 3

`JFC12.palette` - for Swing 1.1 beta 3 and later

The Swing components appear on your palette, as shown in Figure 7-1.

If you try to load the Swing jar file by choosing “Load jar file...” from the Palette menu, an informative dialog is displayed. This describes clearly what you should do.

7.3.2 *Converting to New Package Names*

Visaj provides a simple means of converting existing designs that use the older `com.sun.java.swing` package to designs that use the `javax.swing` packages. To do this, start Visaj with the following flag:

```
-Dvj.convertPackages=true
```

Any `.vcl` files then loaded are automatically converted to use whichever version of the Swing component set is available.

7.4 *Using Swing*

Once you have loaded the Swing component set into Visaj, they are selectable from the palette in exactly the same way as the AWT component set.

For general information on using components in Visaj, see the appropriate chapter in this user’s guide. This chapter specifically covers the use of Swing components in Visaj. This includes coverage of the following points:

1. Adding Swing Components to a Design. This describes the slightly different behavior you can encounter when adding Swing components in Visaj.
2. Highlighting of Non-Opaque Components. This describes the way the highlighting of components is affected by the “opaque” resource.
3. Swing Tips. The sub-sections here cover Swing components which need more information for their use in Visaj.

7.5 *Adding Swing Components to a Design*

Swing components are added to your design in exactly the same way as any other components. Some Swing components are made up of a collection of objects. Visaj gives you access to all the constituent parts and

lets you choose whether or not you wish to view them. By default, they are not shown. JFrame is a Swing component with composite objects. When this is added to your design, the hierarchy shown in Figure 7-3 appears.

These extra components are not added by Visaj, they are an integral part of the Swing component.

To add children to the JFrame, you can select the JFrame component at the top of the hierarchy. Children are added to the appropriate widget, which is usually the contentPane. Menubars are added to the LayeredPane. You can, of course, select the appropriate pane directly.

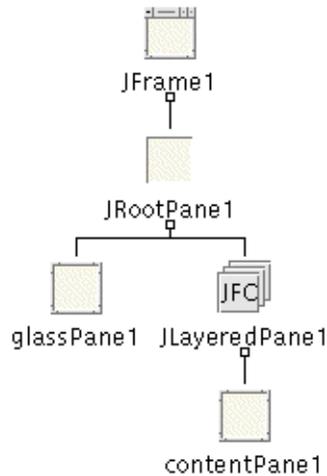


Figure 7-3 Hierarchy of Components Making up JFrame

7.5.1 Showing and Hiding Extra Children

Show and hide the “extra” components by selecting one of the following toggles from the View menu:

1. Collapse all composite components
2. Collapse/Expand selected composites

The first option switches the visibility of composite components in your design. The second option affects only selected components.

These options only change the view and allow you more space for your design. This has no affect on the generated code.

7.5.2 *Other Composite Components*

The following components also add children automatically:

1. JScrollPane
2. JRootPane
3. JInternalFrame
4. JDialog
5. JWindow
6. JApplet

You can mix Swing components with other components in the same design.

7.6 *Highlighting of Non-Opaque Components*

Those Swing components derived from JComponent which have their “opaque” resource set to “false”, are *not* highlighted in the dynamic display when they are selected in the design area. This is because non-opaque components become invisible when they are highlighted. Components which are non-opaque by default include:

1. JLabel
2. JCheckBox
3. JRadioButton

7.7 *Swing Tips*

Some of the more versatile Swing classes require additional information for their use in Visaj. The rest of this chapter covers this information - the Editor provided for BorderLayout, the customizers supplied for JList and JTable and how to use Borders and ButtonGroup in your design. A more general hints section is included at the end of this chapter.

7.7.1 BorderLayout

Swing provides a new layout component, the BorderLayout. This is assigned automatically to components such as JToolBar. In order to set the BorderLayout for other components in Visaj, use the Box component, available on the object palette in the “JFC Containers” section. Box is a component with a BorderLayout plus additional methods for using the layout. See your JFC documentation for a more detailed description of Box.

Box Layout Editor

To display the Box Layout Editor, press the Layout Editor toolbar button when a container with a BorderLayout is selected. More detail on Layout Editors in general is given in the *Layout Editors* section on page 69.

BoxLayout, shown in Figure 7-4, lays out its children either horizontally or vertically in a single line. Change the orientation by selecting from the option menu on the left. The change is immediately reflected in the dynamic display.

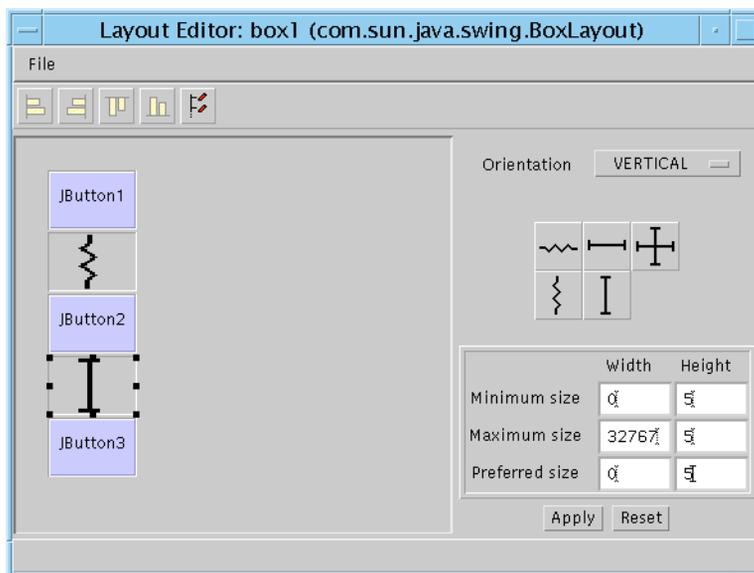


Figure 7-4 Box Layout Editor

The components controlled by this layout manager are shown on the left of the window. The area on the right contains:

1. The orientation option menu.
2. A panel of invisible components which help to control the layout.
3. A set of text boxes for editing the dimensions of the invisible layout components.

Table 1 on page 97 lists the invisible components and explains what they do.

Table 1: Invisible Layout Components in Box Layout Editor

Icon	Invisible Component Name
	Horizontal glue. This provides a stretchable horizontal space between components.
	Vertical glue. This provides a stretchable vertical space between components.
	Horizontal strut. This gives you a fixed amount of horizontal space between components.
	Vertical strut. This gives you a fixed amount of vertical space between components.
	Rigid area. This is an invisible component which gives you a fixed amount of horizontal <i>and</i> vertical space between components.

To add an invisible component, select a component in the area on the left of the Layout Editor (this can be a visible or invisible component) and then press the invisible component button on the right. The invisible component is added *before* the selection. The component is also added to the containment hierarchy.

To cut or delete an invisible component, go to the containment hierarchy, select it and choose cut or delete as you would for any other component.

The area beneath the invisible component buttons on the right of the Layout Editor allow you to change the minimum, maximum and preferred sizes of the selected invisible component once it has been added to the BorderLayout. The preferred size is the size used when the container is displayed. The maximum and minimum sizes affect what happens when the container is resized.

Remember to press the “Apply” button if you change the sizes. The changes do not take effect and may be lost unless the “Apply” button is pressed.

7.7.2 *JList*

To populate a JList component, select the JList and choose “Customize...” from the Object menu. The dialog displayed in Figure 7-5 is displayed.

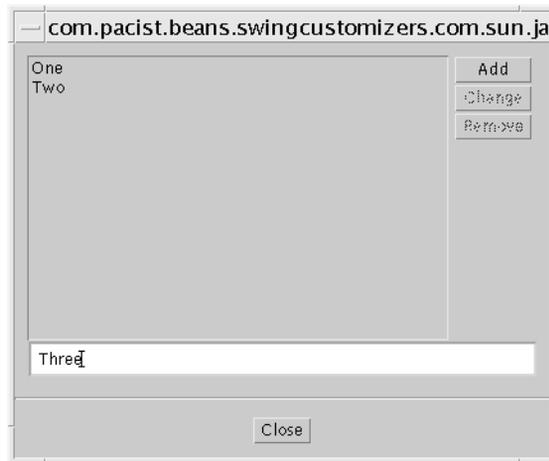


Figure 7-5 List Customizer

Type each string to be added to the list into the text field at the bottom of the dialog and press Return (or the “Add” button) to add them to the list. To change a list item, select it, change the string in the text field and press “Change”. To remove an item, select it and press “Remove”.

7.7.3 *JTable*

To fill in a *JTable*, use the *JTable* Customizer. Select the *JTable* and choose “Customize...” from the Object menu. The dialog shown in Figure 7-6 is displayed. This dialog has a column editor at the top and a graphical representation of the table underneath. The column editor allows you to change the column heading, specify the data type of each column and select whether the column can be resized or not. When you type a string into the column title field, it is not shown in the *JTable* until you press Return.

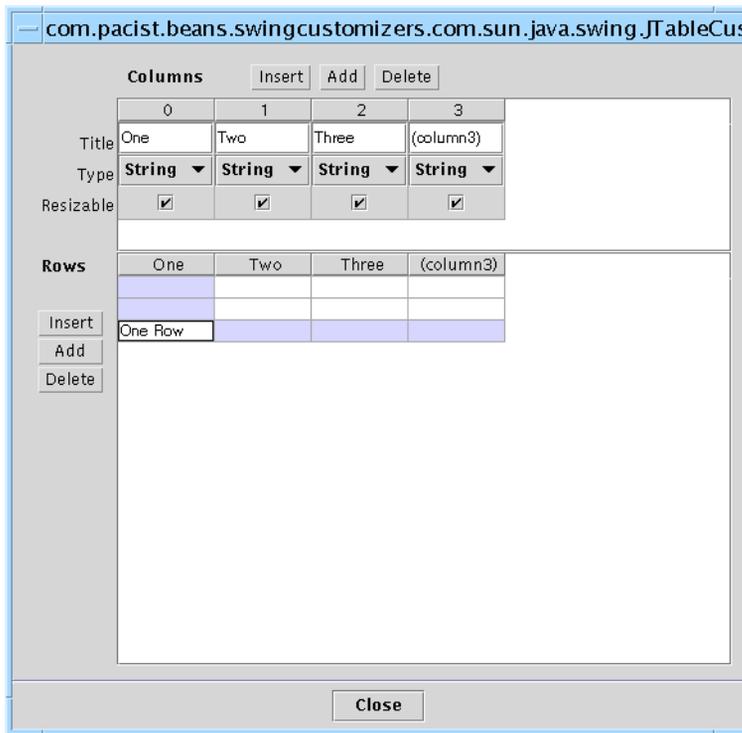


Figure 7-6 Table Customizer

To add a column to the JTable, press the “Add” button at the top of the dialog. To add a row, press the “Add” button at the side. Similarly, the “Insert” button inserts a row or column before the current selection in the graphical representation. “Delete” removes the selected row or column from the JTable.

7.7.4 Borders

Borders are Swing components which put a visible “edge” around a component. Any Swing component derived from JComponent has a “border” property, as shown in Figure 7-7.

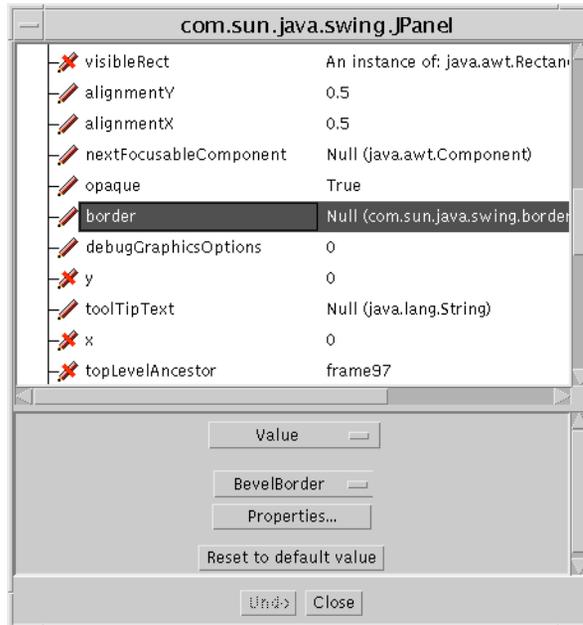


Figure 7-7 Border Property

This property refers to an instance of the type of Border required. By default, it is Null (no border).

The property editor for Borders is an option menu of the different Border classes available. When you select a Border, the “Properties...” button is enabled. Pressing this displays a Border Editor which allows you to change the properties of the selected Border.

The following Borders are available:

1. `BevelBorder`. Use the editor to change the colors of the highlight and the shadow and to specify whether the border is raised or lowered.
2. `SoftBevelBorder`. The editor for this border is identical to the one above.
3. `EmptyBorder`. The editor allows you to edit the insets for this border.

4. `MatteBorder`. Use the editor to edit the insets, change the color and the image used for the border.
5. `EtchedBorder`. The editor for this border allows you to specify whether it is raised or lowered and to change the color of the highlight and shadow.
6. `LineBorder`. Use the editor to alter the thickness and color of the line used for this border.
7. `TitledBorder`. The editor for this border allows you to type in the text of the title, to specify the position of the title, the font to use and the color of the border.

After you have edited the properties of a Border in the Border Editor, press the “apply” button in the bottom left hand corner of the Editor. The properties are not set until you do so.

As with all properties in the property sheet, you can also choose to specify the variable name of an existing Border or type in a code expression to set a component’s border property.

7.7.5 *ButtonGroup*

To add a `ButtonGroup` to your design, make sure that nothing is selected in the object hierarchy and select the `ButtonGroup` from the object palette. The palette icon is shown in Figure 7-8.



Figure 7-8 `ButtonGroup` Palette Icon

Specify which buttons belong to the `ButtonGroup` by adding the lines highlighted in the following piece of code. This code was generated from a simple design comprising a `JFrame` containing two `JCheckBoxes` and a separate `ButtonGroup`, as shown in Figure 7-9.

The location of the added code is important because of the need to retain this code when regenerating from Visaj. The code has been added after a “//vj-” line. For more information on safely adding code, see the *Editing the Code* section on page 154.

```
//vj+ <VJ-BeginMethodDef>
// Method# 1
public MyClass() {
//vj- <VJ-BeginMethodDef>

//vj= <VJ-MethodCode>
//vj+ <VJ-DefineAWTMembers>
buttonGroup1 = new ButtonGroup();
JFrame1 = new JFrame();
JFrame1.setTitle( "JFrame1" );
JRootPanel = JFrame1.getRootPane();
glassPanel = (JPanel)JRootPanel.getGlassPane();
JLayeredPanel = JRootPanel.getLayeredPane();
contentPanel = (JPanel)JRootPanel.getContentPane();
JCheckBox1 = new JCheckBox();
JCheckBox1.setText( "JCheckBox1" );
JCheckBox2 = new JCheckBox();
JCheckBox2.setText( "JCheckBox2" );
{
    String strConstraint;
    strConstraint = "Center";
    contentPanel.add(JCheckBox1, strConstraint, -1);
    strConstraint = "North";
    contentPanel.add(JCheckBox2, strConstraint, -1);
}
JFrame1.pack();
JFrame1.show();
//vj- <VJ-DefineAWTMembers>
```

Add these lines:

```
buttonGroup1.add(JCheckBox1);
buttonGroup1.add(JCheckBox2);
buttonGroup1.setSelected(JCheckBox2.getModel(), true );
```

```
//vj+ <VJ-EndAWT>
```

```

//vj- <VJ-EndAWT>

//vj+ <VJ-EventListenerClass>
//vj- <VJ-EventListenerClass>

//vj+ <VJ-AddEventListeners>
//vj- <VJ-AddEventListeners>

//vj= <VJ-Classes>

//vj+ <VJ-EndMethodDef>
}
//vj- <VJ-EndMethodDef>

```

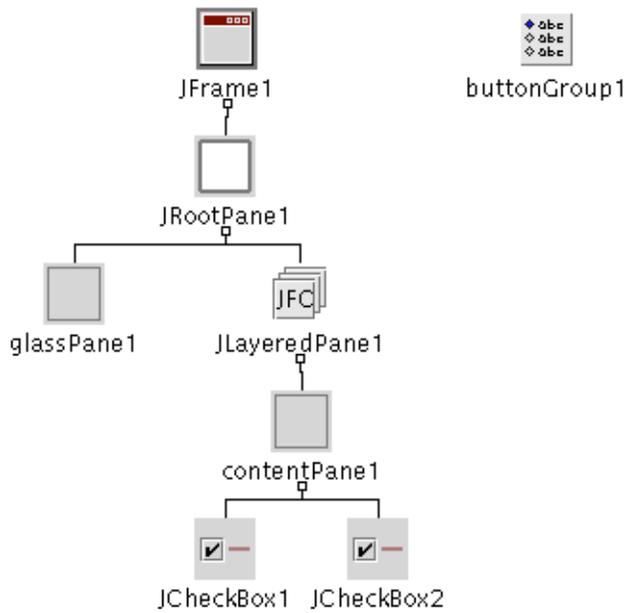


Figure 7-9 Simple Design for Code Example

7.7.6 JApplet

You can make your class into an instance of JApplet by setting the superclass to `javax.swing.JApplet`. The *Editing Properties of the Class* section on page 40 tells you how to change the superclass of your class. As soon as you change the superclass and press “Apply”, the “this” icon on the object palette changes to the JApplet icon. Selecting “this” from the palette allows you to add objects to your applet class.

7.7.7 Miscellaneous Component Tips

The following components each require some further explanation.

JSplitPane

When a JSplitPane is added to your design, the dynamic display appears to contain two buttons - one on the left and one on the right (as shown in Figure 7-10). These cannot be accessed and are only present as place holders. Any components added as children of JSplitPane replace the buttons.

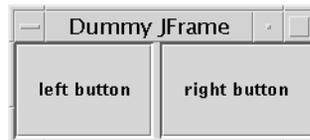


Figure 7-10 JSplitPane Dynamic Display

JDesktopPane

JDesktopPane has a default width and height of 0. To prevent this container from “collapsing”, set a preferred width and height in the component’s property panel.

JTabbedPane

Add tabs to the `JTabbedPane` simply by adding children to it - each child is one tab. A tab activation button appears for each child component. Change the name of the tab and its icon in the property sheet of the child component. Adding a container as a tabbed panel allows you to put any number of components on one tab. Change the orientation of the tabs in the property sheet of the `JTabbedPane`.

JLayeredPane

To set the layer number of each child of a `JLayeredPane`, set it in the property sheet of the child.

8.1 Description

The Image Editor, shown in Figure 8-1, can create new images or edit existing ones. It is a multiple document tool allowing you to work on more than one image at the same time. The completed image is stored in an external file.

The Image Editor allows you to change the color, appearance and size of your image. An extensive set of filters, capable of manipulating whole images or just parts of them, is provided as part of the Image Editor. The filters are described at the end of this chapter. The best way to learn about the filters is try them out - the undo facility lets you do this safely on your image.

A short tutorial, demonstrating just a small sample of the capabilities of the Image Editor, follows this section. The remainder of this chapter describes each area of the Image Editor in more detail.

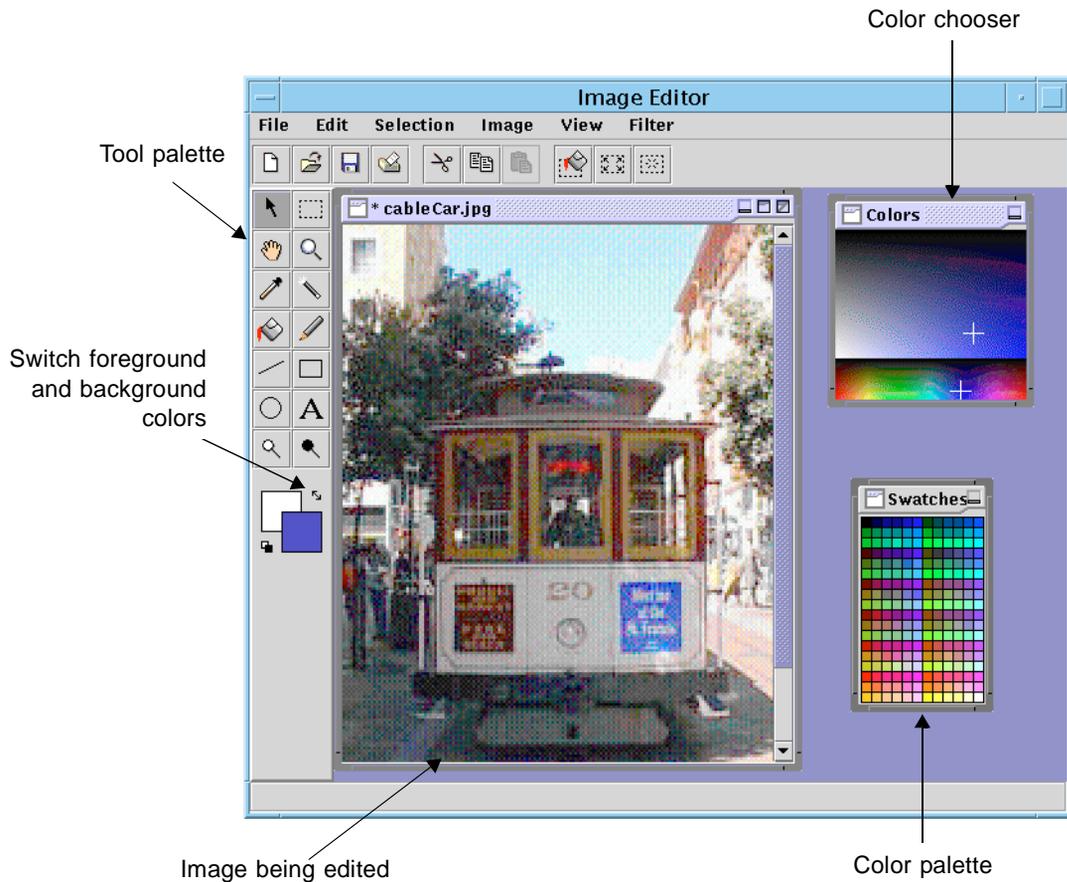


Figure 8-1 Image Editor

8.2 Tutorial

This simple step-by-step tutorial shows some of the capabilities of the Image Editor. You will create the image shown in Figure 8-2 by starting with a fresh window, giving it a textured background, adding text, embossing it, creating a drop shadow for the text and giving the appearance that the text fades out.

8.2.1 A Note Before you Start

During the course of this tutorial, remember that you can use the undo facility if you have tried something and you are not happy with the result. Simply select “Undo” from the Edit menu directly after an unsuccessful operation. You may also use wish to the undo feature to experiment along the way. Making frequent backups also allows you to revert to a saved image at any time.



Figure 8-2 Image Editor Tutorial - Final Image

1. **In the Image Editor, select “New” from the File menu.**
This displays the Image Size dialog.
2. **In the Size dialog, specify a width of 320 and a height of 140. Press OK.**
3. **Choose a color - something bright like pink - from the Swatches color palette.**
Do this by clicking over the color square of your choice. The Swatches window is shown in Figure 8-9.

When a colour is selected from the color palette, it becomes the foreground color and is displayed in the foreground square in the bottom left corner of the Image Editor window. We are going to make this color the background color.

4. **To switch over the foreground and background colors, click over the double-headed arrow above the foreground/background squares.**
This is shown in Figure 8-1 and in more detail in Figure 8-8.
5. **Click over another color from the palette in order to set the foreground color.**
Choose a good contrast - green, for example.

- 6. In order to achieve an interesting effect on the background, select the gradient tool from the tool palette.**

If you are not sure which one this is, look it up in Table 2 on page 118 or pass the cursor over each tool and read the Tool Tip which is displayed as you do so.

- 7. With the gradient tool selected, choose “Tool Properties” from the Edit menu.**

A shortcut way of displaying this dialog is to double-click over the tool in the palette.

- 8. In the gradient tool Properties dialog, shown in Figure 8-15, make sure that the “Use Colormap” toggle is off and select “Conical Symmetric” from the Type list.**

The gradient tool is described in detail in the *Gradient Tool* section on page 128.

- 9. Making sure that the gradient tool is still selected on the tool palette, click in the centre of your image window and drag a line outwards to the right edge.**

Of course, you can drag any length of line and in any direction to create different effects, but for the purposes of this tutorial you need to ensure that you have the foreground color on the right of the image. This color will be used by the gradient tool later in the tutorial.

You should now have an interesting “conical” effect in your new image with the foreground and then the background colors appearing to “wrap around” the window. We shall now add some text.

- 10. Save your image by selecting “Save” from the File menu. You are prompted for the file type. Choose one which suits you best - the Image Editor can read back in any of the formats listed.**

It is important to save your work from time to time.

- 11. Select the text tool from the tool palette.**

If you are not sure which one this is, look it up in Table 2 on page 118 or use the Tool Tips, displayed when you pass the cursor over the tools.

12. Click near the left side of your image.

You need to specify first where your text is going to go. When the text is placed in the image it is selected so you can move it around. Once you have clicked in your image, the text properties dialog appears, as shown in Figure 8-3.



Figure 8-3 Text Tool Properties Dialog

- 13. In the text tool properties dialog, specify a large font size, such as 90, choose a font that you like and type “Imagine” into the text field.** Although the text is displayed the correct size, the dialog may be too small to display it all. In this case, simply resize the dialog.
- 14. When you are happy with the size and appearance of the text in the properties dialog, press “OK”.**

The text is inserted into the image using the foreground color - the color is not important at this point as we shall be changing it using one of the filters. At this point, you can move the text by clicking *inside* the text and dragging it to the desired location.

It is important to remember at this point that the text is added as a floating selection (see the *Floating Selections* section on page 123 if you are not sure what this is). We need to keep it as a floating selection until we have finished with it, so do not do any other type of drawing or selecting.

Another point to remember is that if you are not happy with the text when it is added to the image, press “Undo” in the Edit menu and start again with the text by clicking in the image to display the text properties dialog. This applies to any drawing or filter - you can always “Undo” them and try again.

15. Save your image.
16. With your text in the image and still selected (surrounded by a red band), go to the **Filters** menu and choose “**Shapeburst**” from the **Stylize pullright** menu.
The Shapeburst dialog is displayed, as shown in Figure 8-4.

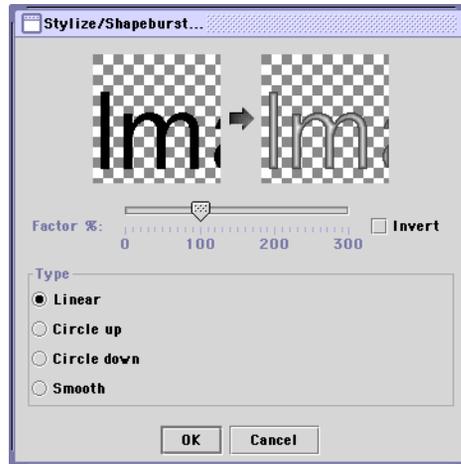


Figure 8-4 Shapeburst Filter Dialog

17. You do not need to change any of the default settings in the **Shapeburst** dialog, so press “**OK**”.
Your text is now given a rounded appearance using a gray color. The Shapeburst filter is useful as a preprocessor to the Embossing filter, which we shall use next.
18. With the text still selected, choose “**Emboss**” from the **Stylize pullright** menu in the **Filters** menu.
The Emboss dialog is displayed, as shown in Figure 8-5.

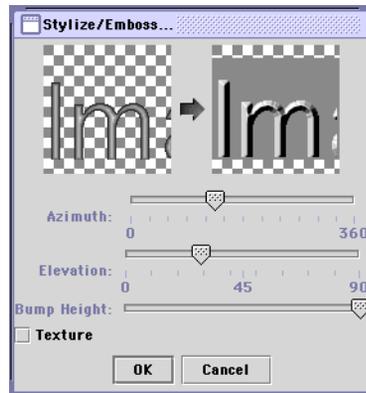


Figure 8-5 Emboss Filter Dialog

19. **Once again you do not need to change the defaults in this dialog, simply press the “OK” button.**

Your text now has an embossed look.

Note – Remember that if you apply a filter and then wish to change it, you can select “Undo” from the Edit window and try again.

20. **Select “Drop shadow” from the Stylize pullright menu in the Filters menu.**

This displays the drop shadow dialog, shown in Figure 8-6.

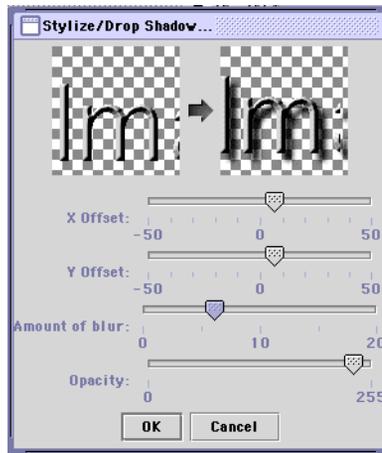


Figure 8-6 Drop Shadow Filter Dialog

21. **For this filter, you may wish to adjust some of the default settings, Try turning down the “Blur” to make a deeper shadow and turning up the X and Y offsets very slightly to make the text stand away from the shadow a little more.**

The filter dialog shows you an example of the changes you are making. Sometimes these dialogs are slow in redrawing because of the complexity of the mathematical operations they are performing. Please be patient.

22. **When you have selected settings you like, press “OK”.**
Your text now has a drop shadow, making it appear to jump away from the image. Save your image.

For the final stage of the tutorial we are going to create and use a gradient. More detailed information on gradients is given in the *Gradients* section on page 125.

23. **In the gradient list window (shown in Figure 8-13) select “New” from the File menu.**

A new gradient is added showing a black to white gradient.

- 24. Select the new gradient by clicking over it in the list and then choose “Edit” from the File menu.**

The gradient editor is displayed, as shown in Figure 8-14. More information on the gradient editor is provided in the *Gradient Editor* section on page 126.

The gradient editor displays two small arrowheads underneath the actual gradient. In order to create a “fading out” effect, we need to set the gradient to range from transparent on the left (to allow the text to show through) to the colour of the background pattern on the right. The applied gradient will then show the text gradually blending in to the background.

- 25. Select the dropper tool from the tool palette. This tool allows you to “pick up” colors from an image.**

If you are not sure which one this is, look it up in Table 2 on page 118.

- 26. Click over the color in your background pattern at the far right of your image. Figure 8-7 shows you where.**

This will set the color picked up with the dropper tool to be the foreground color.



Figure 8-7 Using the Dropper Tool

- 27. Back in the gradient editor, click directly above the arrowhead on the right.**

This sets the foreground color for that end of the gradient. If this does not happen on the first attempt, keep trying until the appropriate color appears.

- 28. Back in the Image Editor (with the gradient editor still displayed), click over the foreground color square.**

This sets it to be transparent and is indicated by a checkerboard pattern.

29. **In the gradient editor, click in the color directly above the arrowhead on the left.**
This sets transparency for the other end of the gradient.
30. **From the Type menu in the gradient editor, choose “Circle Up”.**
This should give us the effect we require. Remember that you can experiment with other types by using the “Undo” button in the Edit menu after using the gradient tool.
31. **Close the gradient editor.**
Your new gradient is displayed in the gradient list.
32. **Select your new semi-transparent gradient in the gradient list.**
33. **Select the gradient tool from the tool palette and display its properties dialog again.**
If you cannot remember how to do this, refer back to Step 7 on page 110.
34. **This time, set the “Use colormap” toggle in the properties dialog and choose “Linear” as the Type.**
35. **Choose “Select none” from the Selection menu.**
This is to make sure that subsequent operations affect the whole image.
36. **In your image, click halfway down on the far left side and drag a line right across to the right side.**
Make sure that you have drawn the line right across the whole image.

You should now have the effect shown in Figure 8-2 except that in color the effect is much more pronounced and interesting.
37. **Save your image.**
Now that you have a completed image, you can use it in a Visaj design or you can print it directly from the Image Editor. Choose “Print” from the File menu to do this.

8.3 Image Files

Open an existing image by selecting “Open...” from the File menu or by pressing the Open button on the toolbar. The image is placed in its own window within the Image Editor.

Selecting “New” from the File menu, or pressing the corresponding toolbar button, displays a dialog prompting you for the size of the new image. You can change the size of it later by selecting “Image Size” from the Image menu. Once you have pressed “Ok” in the Image Size dialog, you are given an empty editing window in the Image Editor.

Because the Image Editor can contain any number of image files, each in its own window, there is a *current* image. The current image is displayed on top of any others. All operations affect only the current image. The “Close” item in the File menu closes the current image only.

8.3.1 Saving Files

Choose “Save” (or “Save As...”) from the File menu or the Save button on the toolbar to write out the current image. Once saved, the image can then be used in Visaj as an object’s image property.

The Image Editor can read and write files in JPEG¹ and PNG formats and Swing ImageIcons.

Before saving, the Image Editor prompts you for the file type.

8.3.2 View Menu

The items in the View menu affect only how you see the image in the Image Editor. Nothing in this menu changes the saved image. From this menu you can:

1. Zoom in.
2. Zoom out.
3. Change the display ratio. 1:1 is the “real size” of the image.
4. Turn the grid on and off. The grid is automatically *off* for 1:1 or 2:1 displays.

1. The JpegEncoder and its associated classes are Copyright (c) 1998, James R. Weeks and Bio-ElectroMech. This software is based in part on the work of the Independent JPEG Group.

8.4 Help

For all buttons in the Image Editor, tool tips are displayed when the mouse is passed over them. Tool tips do not appear until the pointer has been positioned over a button for one or two seconds.

The status line at the bottom of the Editor displays information on the currently selected tool. Keyboard modifiers are included in this information.

8.5 Tool Palette

The tool palette on the left of the Image Editor allows you to change between methods of editing. Table 2 lists all the palette buttons along with a brief description of their function.

Table 2: Palette Tools

Button	Function
	Pointer Tool. Use this to move the current selection.
	Rectangle Selection Tool. Make rectangular selections with this tool. Holding down the Shift key adds the new selection to any existing selections. Holding down the Control key subtracts the new selection. To select irregular areas, use the Magic Wand tool.
	Panning Tool. With this tool, pan the image left/right and up/down.
	Zoom Tool. Zooms the image in. Holding down the Control key zooms out.

Table 2: Palette Tools

Button	Function
	Ink Dropper Tool. Use this to “pick up” colors in the image. Any color selected with this tool, becomes the foreground color.
	Magic Wand Tool. Click over one pixel with this tool and the selection spreads in each direction until a different color is found. This allows you to select non-rectangular areas. See the <i>Tolerance</i> section on page 120 for information on refining this process. Holding down the Shift key adds to an existing selection, holding down the Control key subtracts from it.
	Fill Tool. Use this tool to flood an area with the foreground color. The area to flood is all pixels with the same color as the one clicked over until a different color in each direction is reached. See the <i>Tolerance</i> section on page 120 for information on refining this process. Hold down the Control key to use the background color.
	Pencil Tool. Colors individual pixels using the foreground color. Hold down the Control key to use the background color.
	Line Tool. Draws lines in the foreground color. Hold down the Control key to use the background color.
	Rectangle Tool. Draws outline rectangles in the foreground color. Holding down the Shift key draws a filled rectangle. Hold down the Control key to use the background color.

Table 2: Palette Tools

Button	Function
	Circle Tool. Draws outline circles in the foreground color. Holding down the Shift key draws a filled circle. Hold down the Control key to use the background color.
	Text Tool. Use this to draw text into your image. Select a start location in the image first. A dialog is displayed for you to enter the text and change the font, size and style before it is added to the image. When added to the image, the text is a “floating” selection - see the <i>Floating Selections</i> section on page 123 for more details,
	Dodge Tool. Use this to lighten parts of your image. Hold down the mouse button and drag across an area to lighten it.
	Burn Tool. Use this to darken parts of your image. Hold down the mouse button and drag across an area to darken it.
	Gradient Tool. Use this to paint gradients into the image. This tool and its properties dialog is described in the <i>Gradient Tool</i> section on page 128.

8.5.1 Tolerance

You can set the tolerance of the magic wand and fill tools in order to select or fill an area with similar (as opposed to identical) colors. Select “Tool Properties...” from the Edit menu to display the Tolerance dialog. This dialog contains a slider which can be set to any value between 0 and 255. 0 (the default) indicates no tolerance - only exact color matches are selected or filled. A tolerance of 255 will, effectively, match every color. The effect of

any setting in this dialog is immediate. This means that you can try different tolerance values easily until you find the one that matches your criteria.

8.6 Colors

The Image Editor works with a background and a foreground color. These are displayed on the left of the Editor window, as shown in Figure 8-8.

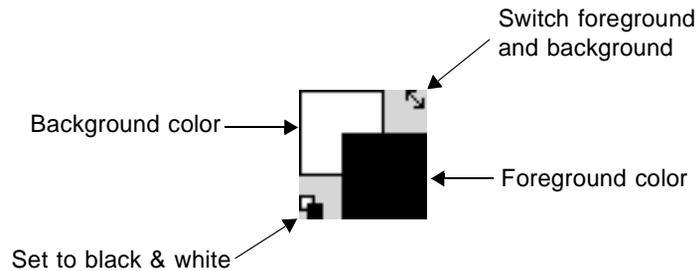


Figure 8-8 Foreground and Background Colors

8.6.1 Setting the Foreground and Background Colors

Any color selected from either the Color chooser or the Color palette, both of which are shown in Figure 8-9, sets the foreground color.

Any of the tools which paint color onto the image use the foreground color. Holding down the Control key forces the Image Editor to use the background color. The background color is used when, for example, a selection is moved, thereby leaving a “hole”. The gradient tool and some of the filters also use the current foreground and background colors.

As shown in Figure 8-8, you can switch the background and foreground colors by pressing the double-headed arrow. This is how the background color is set.

A shortcut to change the foreground to black and the background to white is provided. Press the black and white squares, also shown in Figure 8-8.

8.6.2 Transparency

Clicking over the background or foreground color square toggles between that color and *transparent*. The color square itself becomes transparent with a cross over it. In the image, transparent pixels are drawn with a check pattern. Transparency is useful for images such as toolbar button icons where the color of the toolbar should show through so that they are consistent on different platforms.

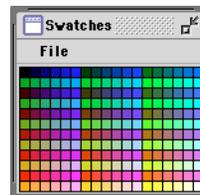
8.6.3 Color Chooser and Color Palette (Swatches)

The Color chooser contains two areas. The lower area shows the full color spectrum, or the *hue*, changing in *brightness* top to bottom. Selecting in here changes the upper area to display the *saturation* (left to right) and *brightness* (top to bottom) of the selected hue.

The Color palette (or Swatches) is a grid of small color squares representing a color cube. The colors of the default swatch are “Web safe”; they are the ones that browsers try to allocate. Keeping to these colors will ensure that your images display well on a Web page. You can, however, load in your own color palettes using the “Load” item in the File menu of the Color palette. Similarly you can save color palettes using the “Save” item.



Color Chooser



Color Palette

Figure 8-9 Color Chooser and Palette

You can choose not to display the Color chooser or Color palette window by selecting the appropriate item in the Window menu. Selecting the item again redisplay the window.

8.7 Selection

Selection is a key feature of the Image Editor. All of the filters and all items in the Image menu affect the currently selected area. Where there is no current selection, the Image Editor assumes that the whole image is selected.

A red line around the selected area indicates a floating selection. A yellow line indicates a non-floating selection.

8.7.1 Floating Selections

Selected areas can be moved around the image and even altered by one of the filters without affecting the image underneath. This is particularly useful for parts of an image such as text which you may need to move around to find a place that is right for it. This ability for selections to move or be altered without affecting the image underneath is referred to as “floating selection”. A selected area is automatically floated whenever it is moved by the Pointer tool or if it is passed through a filter which changes its shape (makes it bigger or smaller). There is a “Float Selection” item in the Selection menu but you will not normally need to use this.

Floating selections are automatically dropped back into the image when you select and use a tool from the Tool palette other than the Pointer tool. You can force a selection to drop back into the image by selecting “Drop Selection” from the Selection menu.

If a selection is not floating (it has a yellow line around it) and you move this selection, a “hole” appears in the image where the selection used to be. The hole shows the background color. Once the selection has been moved, it is then floating and can be moved without leaving a “hole”. Of course, making “holes” in this way can create an interesting effect.

If you have a floating selection, you can change the way it is painted over the image by selecting the appropriate item from the “Paint modes” pullright menu in the Selection menu. This mode only lasts as long as the current selection. Changing the selection reverts to the default paint mode.

8.7.2 Making a Selection

The Rectangle Selection Tool and the Magic Wand Tool, both shown in Figure 8-10, are available on the Tool palette. They provide two ways of making a selection.



Rectangle Selection Tool



Magic Wand Tool

Figure 8-10 Selection Tools from the Tool Palette

As its name suggests, the Rectangle Selection Tool allows you to make rectangular selections. Hold down mouse button 1 and drag a rectangle around the area you wish to select.

The Magic Wand Tool allows you to select non-rectangular areas. Click over one pixel to spread the selection in each direction until a different color is found. This type of selection is suitable for selecting blocks of the same color.

Adding to Selection

Increase the selected area by holding down the Shift key while selecting. This will add the new selection to any already present.

Subtracting From Selection

To subtract areas out of a selection, hold down the Control key while selecting the area to subtract.

8.7.3 Selection Menu

The items in the Selection menu help you to refine and alter your selected area. These items are:

1. Select All. Selects the whole image. This is identical to the toolbar button shown in Figure 8-11.



Figure 8-11 Select All Toolbar Button

2. Select None. Deselects the whole image. This is identical to the toolbar button shown in Figure 8-12.



Figure 8-12 Select None Toolbar Button

3. Select Foreground. This item selects all pixels in the image which are the same color as the current foreground. Set the foreground to one of the colors in the image by using the Dropper Tool. See Table 2, "Palette Tools," on page 118.
4. Invert. Changes the current selection to the whole image *except* the previously selected area.
5. Grow. Increases the selected area one pixel in each direction.
6. Shrink. Decreases the selected area one pixel in each direction.

8.8 Gradients

The window entitled "Gradients" in the Image Editor shows your list of defined gradients, or colormaps, as shown in Figure 8-13.

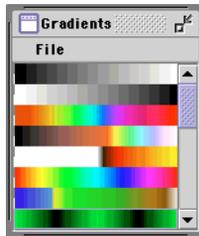


Figure 8-13 Gradient List

A gradient is a way of creating color sets. There is always a “start” and an “end” color and the Image Editor makes a smooth transition between the two. You may have any number of subsets, each making a color transition, within one gradient and you may also specify the way in which the color transition is made. All of this is performed in the Gradient Editor which is described below.

The Gradient List appears with a default set of gradients. From the Gradient List’s own File menu you can remove a gradient, create a new one or duplicate an existing one, save the list of gradients and load a saved set of colormaps. Place a saved list of gradients into the plugins directory to make the Image Editor load it automatically when it starts up. If you wish to do this, make sure that the list of gradients is saved into a file with a `.ser` extension. The plugins directory is `<VISAJROOT>/lib/plugins`, where `<VISAJROOT>` is the install directory of your Visaj.

Choosing “Edit” from the File menu displays the Gradient Editor, allowing you to edit the currently selected gradient. Click over a gradient in the list to select it. The currently selected gradient is used by the Gradient Tool and by those filters with a “Use Colormap” toggle.

Selecting “Colormaps” from the Window menu hides the Gradient list. Selecting this item again redisplayes the window.

8.8.1 Gradient Editor

When you select “New” from the Gradient List’s File menu, a new gradient showing one smooth transition from black to white appears in the list. To edit this new gradient, select it and then choose “Edit” from the File menu. The Gradient Editor, shown in Figure 8-14, is displayed.

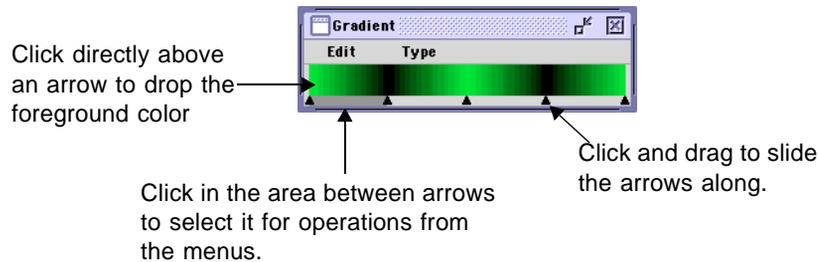


Figure 8-14 Gradient Editor

In the Gradient Editor, there is a line of color representing the gradient, an area underneath where the gradient can be segmented and a menubar containing two menus at the top.

The segments within a gradient allow you to add more color and color transitions to your gradient. A new gradient contains one segment. To add new segments, select “Split segment” from the Edit menu. To remove segments select “Remove segment” from the Edit menu. To select a segment, select the area between the segment arrows. The operations available from the two menus apply to the currently selected segment. Click in the color directly above a segment arrow to “drop” the foreground color at that point. The Image Editor then makes a smooth transition between the colors of adjacent segment arrows.

The Type menu contains items which affect the way the transition is made between the colors at either end of a segment. There are two sections in this menu. The top section refers to the interpolation type and the lower section to the color type. For the top section, your choices are:

1. **Constant.** This fills the segment with solid color using the color on the left of the segment. Use this with multiple segments to create sharp stripes.
2. **Linear.** This gives a consistent spread of color change across the segment.
3. **Spline.** This applies the color change as a “wave” and therefore makes segments join together more smoothly.

4. **Circle Up** and **Circle Down**. These transformations give a smoother, more rounded effect when used to fill an image.

For the lower part of the menu, you can choose between the following:

1. **RGB**. This uses RGB values to move from one color to another.
2. **Clockwise Hue** and **Counter-clockwise Hue**. These show the relevant section of a color wheel as defined by the two colors of the segment. You can ask for a clockwise or counter-clockwise journey between the two colors.

Your edits in the Gradient Editor are not applied until the Gradient Editor is closed.

You may use transparency in gradients to create an interesting effect. If you choose transparent as the color at one end of a gradient, the image appears to gradually “fade” into the background.

8.8.2 Gradient Tool

The gradient tool, available from the Tool Palette, allows you to draw gradients into your image. To use the gradient tool, drag a line in your image along which you wish to paint a gradient. By default this tool uses the foreground and background colors as the start and end points of the gradient. Change this behavior in the properties dialog which is displayed by either double-clicking over the tool or by selecting it and choosing “Tool properties” from the Edit menu. The dialog is shown in Figure 8-15.

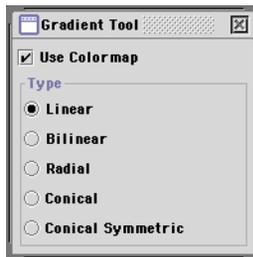


Figure 8-15 Gradient Tool Properties Dialog

The “Use Colormap” toggle in the properties dialog tells the Image Editor to use the currently selected gradient from the Gradient List instead of simply using the foreground and background colors. The rest of the properties dialog refers to the way the gradients are drawn. The following are short descriptions of each type of drawing style. By far the best way to understand them, however, is to try them:

1. **Linear.** This draws the gradient straight along the line you drag across the image.
2. **Bilinear.** This draws the gradient twice along the dragged line - from one end to the other and then back again.
3. **Radial.** This draws outwards in a radial pattern from the start of the dragged line to the end.
4. **Conical.** This draws the gradient *around* the start point with the colors starting from the end point.
5. **Conical Symmetric.** This is the same as Conical but drawing the gradient twice - once forward and once backward.

Use the Shift key when dragging a gradient line to “cycle” the gradient. This causes the gradient to be repeated across the image. The length of the line determines the size of the repeating pattern.

8.9 *Editing the Image*

The Edit menu contains items to Cut, Copy, Paste and Clear the current selection. All operations which modify the image can be undone by selecting “Undo” in this menu.

The Image menu contains extra editing functions. These can be divided into the following sub-sections.

8.9.1 *Changing the Image Size*

There are two operations in the Image menu which directly affect the size of the image:

1. Crop
2. Image Size

Crop cuts the image down to the size of the bounding box which contains the selection. If you have a rectangular selection, the cropped image is the same as the selection.

Selecting “Image Size” displays the dialog shown in Figure 8-16. Change the width and height of the image in this dialog.

If you have the “Keep aspect ratio” checkbox selected, the width and height fields are automatically kept to the same ratio.



Figure 8-16 Image Resize Dialog

The Resize dialog allows you some control over what happens when the image is resized. Choose one of the following:

1. Don't rescale image. The image itself stays the same. It is cropped if the new size is smaller and the extra area is filled with background color if the image is made larger.
2. Replicate pixels. The image is scaled by simple replication.
3. Average pixels. The image is scaled using an algorithm which produces smoother results than simple replication. Pixels are blended with adjacent pixels to achieve the effect of the same size ratio.
4. Tile image. The image is repeated to fill the new area. This only applies when an image increases in size.

8.9.2 Flipping

The image, or the selected part of it, can be flipped across three axes:

1. Horizontal
2. Vertical
3. Diagonal

Because flipping an image causes its size to change, the Image Editor applies the following rules:

- If there is no selection, the Image Editor assumes that the whole image should be flipped. The image is then resized to display the flipped area.
- If there is any selection (even if this is the whole image), the image is not resized and the selection is left *floating* after being flipped. See the *Floating Selections* section on page 123 for information on this.

Flipping the whole image diagonally results in a “blank” area being exposed. This area is left transparent.

8.9.3 Rotating

The whole image or a selected part of it can be rotated through 90°, 180° by selecting the appropriate item or through any other angle by selecting “Rotate...” and typing the angle in the Rotate dialog. The Rotate dialog gives you a preview of the rotated image as you type.

If you are rotating an image which is not square, its size changes. The Image Editor applies the following rules when this happens:

- If there is no selection, the Image Editor assumes that the whole image should be rotated. The image is then resized according to its new shape.
- If there is any selection (even if this is the whole image), the image is not resized and is left *floating* after being rotated. See the *Floating Selections* section on page 123 for information on this.

8.9.4 Filling

The “Fill Selection” item in the Image menu floods the selected area with the foreground color. If there is no selection, the whole image is flooded. This item is identical to the toolbar button shown in Figure 8-17.



Figure 8-17 Fill Selection Toolbar Button

8.10 Filters

Your image, or any selected part of it can be altered by using one of the filters provided in the Filters menu. The filters are divided into pullright sub-menus according to the type of change they effect.

The following applies to all filters:

1. If any part of the image is selected, only that area is filtered.
2. If there is no selection, the whole image is filtered.
3. If the size of the selection has changed as a result of being filtered, the selection is floated. See the *Floating Selections* section on page 123 for more information on this.
4. All filters may be undone using “Undo” from the Edit menu.

Some of the filters need to be customized. If this is the case, a dialog is displayed. A preview of the effect of the filter is provided in this dialog. Most of the items in the Filter menu are pullright menus which group various types of filter. The first two items are commands which are only selectable once a filter has been used. These menu options are described in the following sub-sections. Then follows a brief description of each filter, grouped according to the pullright menu title.

8.10.1 Repeat Last Filter

This item simply repeats the last filter you used. The same settings are used - you are not prompted to change anything.

8.10.2 *Show Last Filter*

This displays the dialog pertaining to the last filter used, allowing you to change the settings if you wish.

8.10.3 *Distort*

1. **Offset.** Moves the image by the X and Y offsets specified in the dialog. The image wraps round. This is useful for creating “seamless” images for tiling into a larger image.
2. **Mesh Warp.** Warps an image using a mesh warp algorithm. You supply two warp grids: a source grid and a destination grid. The filter will warp the image so that pixels on source grid points will move to destination grid points. You could produce animations of warping by repeatedly using the warp filter.
3. **Ripple.** Produces a rippled effect. The dialog allows you to specify the wavelength and amplitude of the ripple.
4. **Sphere.** Applies a fisheye lens type effect to an image. Pixels are displaced according to their distance from the centre of the image.
5. **Twirl.** This filter will distort your image by twisting it around the centre. You can change the angle and direction of twist.
6. **Border.** Adds a border to the edges of the selected area or the whole image if there is no selection. The resulting image is larger than the original.
7. **Water Ripples.** This filter will produce a water ripple effect on your image. You can change the wavelength, phase and amplitude of the ripples and turn antialiasing on or off.

8.10.4 *Colors*

1. **Transparency.** Changes the opacity of the image. Use the dialog to specify how opaque the result should be.
2. **Gamma.** Changes the gamma (brightness) of the image.

3. **Gray Out.** Grays out the image (or selected part). This is useful for creating the icons to use for insensitive buttons.
4. **Adjust RGB.** Alters the RGB (Red, Green, Blue) values of the colors used in the image according to the settings given in the dialog.
5. **Adjust HSB.** Alters the HSB (Hue, Saturation, Brightness) values of the colors in the image according to the settings given in the dialog.
6. **Grayscale.** Converts the colors in the image to grayscale.
7. **Invert.** Changes each color to its complement.
8. **Dither.** Dither the colors according to the selected options in the dialog. You can create a dithered black and white image or selects any one of a number of color dithering algorithms.
9. **Diffusion Dither.** This filter converts an image to a specified number of colors. You can choose straight or serpentine dithering.
10. **Reduce Colors.** Specify the number of red, green and blue colors to use in the image.
11. **Contrast.** Allows you to change the contrast and brightness of the image.
12. **Lookup.** This filter recolors an image by converting it to a grayscale image and passing it through a color lookup table.
13. **Solarize.** Uses a V-shaped transfer curve to convert the colors in the image.

8.10.5 Stylize

1. **Threshold.** Changes each color in the image to black or white depending on the specified threshold. Colors below the threshold are coerced into black, those above are forced to be white.
2. **Mosaic.** Converts the image into mosaic blocks. Use the dialog to specify the size of the blocks. This effect is often used on TV to hide a person's face.

3. **Add Noise.** Changes the color of each pixel by a small random amount which is anything between zero and the number you give in the dialog. Use this to “roughen” a smooth image.
4. **Emboss.** Looks at each pixel in the image and calculates the gradient of brightness according to the pixel’s location relative to a light source. This results in an “embossed” effect.
5. **Oil.** Gives an “oil-painting” effect. This filter can be very slow, especially with large images.
6. **Sparkle.** Draws a sparkle or sunburst effect on an image. You can change the number of rays in the sparkle, the randomness of the ray lengths and the radius of the centre of the sparkle. The sparkle is drawn into the centre of the selected area (or the centre of the image if there is no selection). You can use this filter multiple times to create sparkles on corners in images.
7. **Drop Shadow.** This filter produces drop shadows for images. It uses the shape of the selection as the shape of the shadow. You can change the offset of the shadow in the X and Y directions and the fuzziness and opacity of the shadow. The output of this filter is an image which is larger than the input by the offsets and the shadow blur radius.
8. **Shapeburst.** This filter applies a “shapeburst” gradient to an image. It uses the shape of the selection to determine the shape and then shades from the outside of the shape inwards. You can change the shape of the gradient between linear, circle up, circle down and a smooth transition and you can change the rate at which the gradient changes. By default, the gradient will shade from black at the edges to white in the centre of the shape, but you can also get the filter to invert this. This filter is particularly useful for creating bump maps to enhance the emboss filter.
9. **Marble.** creates a marbled effect. Use the dialog to specify the width, height and turbulence to use for the marbling.

8.10.6 *Blur*

1. **Simple Blur.** Blurs the image by the specified amount.

2. **Gaussian Blur.** This applies a Gaussian blur to the source image. Blurring images is useful for special effects. You can specify the radius of the Gaussian convolution kernel; the larger the radius, the more blur. This filter can be slow if you specify a large radius.
3. **Sharpen.** Gives clearer definition to blocks of color.
4. **Bumps.** The opposite of blur, the bumps filter applies an 'embossing' convolution kernel to the image which sharpens edges, producing a bumpy embossed effect.
5. **Motion Blur.** Blurs the image in one direction, giving the effect of speed.
6. **Detect Edges.** This filter detects edges in an image by applying two Sobel gradient operators and subtracting the results. This produces a 'neon' type effect which can be a good starting point for interesting textures.
7. **Maximum.** and **Minimum.** These filters replace each pixel by the maximum and minimum of the input pixel and its eight neighbors. Each of the RGB channels is considered separately. You can achieve some interesting effects on some images by repeatedly applying these filters.
8. **Median.** Applies a color median operator to the image. The output pixel is the median of the input pixel and its eight neighbours. You can use this filter to remove shot noise from an image. This filter may be slow.

8.10.7 Texture

1. **Texture.** Create a gray texture using the parameters specified in the dialog.
2. **Weave.** This filter simulates woven cloth. You can specify the width of the threads, the gap between them, the pattern of crossings, flat or round threads and whether shading is done at the crossings. You can give the threads constant colours or use the colours from the input image. It can be effective to emboss or use as a bump map the output of this filter.

3. **Checkerboard.** Creates checkerboard patterns. You can change the size of the squares in the X and Y directions and the angle at which they are drawn. You can also specify a “fuzziness” for the edges.
4. **Plasma.** This filter produces “plasma” clouds using a midpoint displacement algorithm. Random colours are assigned to pixels at the corners of the image and then recursively averaged and displaced to produce pixels at the midpoints. You can specify the turbulence (graininess) of the results and can provide a colormap. This filter can be useful for producing stone-like textures.

8.10.8 Binary

1. **Erode and Dilate.** These filters perform binary erosion and dilation, removing or adding black pixels from or to the edges of black areas. You can specify a threshold for the number of neighbors needed to flip pixels.
2. **Outline.** Removes black pixels from the centre of black areas, leaving just the outline.
3. **Life.** Performs one turn of John Conway's game of Life on the image.

Resource Bundle Editor

9

In order to allow the text in your application (button labels etc.) to be easily translated for other language users, you will need to define resource bundles. Instead of entering a string label for a Button or Label component in the Property Sheet, a *key* is used. Keys can then be linked to strings in any number of languages by using the Resource Bundle Editor.

The Resource Bundle Editor, shown in Figure 9-1, is displayed when you open a Resource Bundle save file, select “New Resource Bundle” from the Class Editor File menu, double-click over a Resource Bundle save file in the Visaj project window or when you press the “New Resource Bundle” button on the project window toolbar.



Figure 9-1 Resource Bundle Editor

The Resource Bundle Editor is arranged as a table, with one column per language and one key per line. To add a string into the table, select the table cell and enter the string into the text field at the top of the window.

Press the Return key to put the string into the selected cell. When a cell is selected, the locale of the text field is set to match the language of that cell. If your system supports that language, you will be able to type using characters of the selected language.

9.1 File Menu

The File menu in the Resource Bundle Editor contains items to import and export text as well as the standard file operations for opening, closing and saving files.

“Import language” and “Export language” operate on **one** language at a time, using the currently selected language only.

9.1.1 Export Language

The export command provides a means of listing all keys, in the currently selected language, which require translation. When “Export language...” is selected from the File menu, a dialog appears prompting you for a character encoding for the export file. This is provided so that you can be sure the file you are exporting can be read by whoever will be reading the file. Different languages require different encodings and, indeed, different editors will expect different encodings too. A scrolling list of all commonly used character encodings is provided. Simply select one of these and press “OK”. Once you have specified the encoding, the Resource Bundle Editor prompts you for the name of the file to be exported.

The exported file contains one line for each key. Each line is of the format:

```
<key>=<value>
```

Note – *Only the currently selected language is exported. If you have nothing selected, nothing is exported.*

The selected encoding is saved with the selected language so that you can import back into the same language later.

9.1.2 Import Language

The import commands allow you to read in newly translated data. There is a pullright menu, available from the “Import language” item in the File menu, which allows you to choose whether the imported data should merge with or append to the data already present.

Import operates on the currently selected language only. Make sure that you select the correct language before importing.

Each language is exported using a specified character encoding. The Resource Bundle Editor remembers which encoding was used. If, however, you are not using the same save file which was used for the export command, you will be prompted for the character encoding which was used to export the language initially.

Once it has found the correct character encoding to use, the Resource Bundle Editor prompts you for the name of the file to be imported.

The Resource Bundle Editor expects an import file to have one line for each key. Each line must be of the format:

```
<key>=<value>
```

Note – *The Resource Bundle Editor imports into the currently selected language. If there is no language selected, nothing is imported.*

9.1.3 Save and Open

When a Resource Bundle is saved, the table is serialized into the specified file. By convention, Visaj expects Resource Bundle save files to have the filename extension “vrb”. When a file with such an extension is opened, Visaj will automatically open it in the Resource Bundle Editor.

9.2 Edit Menu

The Edit menu contains Cut, Copy, Paste, Clear, Undo and Redo. These apply to the text in a cell. There are also some extra items allowing you to add and delete languages and keys (which correspond to columns and rows in the table).

9.2.1 New Language

To add new languages to the Resource Bundle Editor, select “Add language...” from the Edit menu. A dialog appears which asks for the new language and country, as shown in Figure 9-2.



Figure 9-2 Add Language in Resource Bundle Editor

There is a small arrow button next to each of these which, when pressed, displays a popup menu containing those languages or countries which are known to Java. When a selection is made from one of these popup menus, the two-letter ISO Language Code (ISO-639) (or two-letter ISO Country Code (ISO-3166)) is displayed in the text area. You may enter these codes directly if you wish. You do not have to enter the country code, but if you do the Resource Bundle Editor tries to find an appropriate flag to display in the table. This is for display purposes only.

If you wish to see a list of the language and country codes, try the following web sites:

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt> (for the ISO Language Codes)

http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html (for the ISO Country Codes)

9.2.2 New Key

To add new keys to your Resource Bundle Editor, press the “new Key” button on the toolbar or choose “Add key” from the Edit menu. A new, blank line appears at the bottom of the table.

9.3 Generating Code

The Generate menu contains two items: “Generate all files” and “Set properties...”. The former item generates a class file for each of the languages in the table - including the default. The filenames of these generated files are constructed by using the class name followed by the two letter ISO code for the language and the two letter ISO code for the country, if provided. The language and country codes are preceded by the under bar (_) character. The “Default” is given no language or country code. Subclassing occurs as follows:

- The “Default” class subclasses ListResourceBundle from the java.util package.
- A class with a language code only, extends the “Default” class.
- A class with a language and country code extends the class with the language code only, if provided. If this is not provided, it extends the “Default”.

The filename is always given the “.java” suffix. The example resource bundle shown in Figure 9-1 contains the French language of the country France. Assuming the class name is “Greetings”, the class file for that language would be:

```
GreetingsResourceBundle_fr_FR.java
```

Here is the contents of that file:

```
/*
** French (France) resource bundle for Greetings
*/

public class GreetingsResourceBundle_fr_FR extends

GreetingsResourceBundle {
    public Object[][] getContents() {
        return contents;
    }

    static final Object[][] contents = {
        {"Yes", "Oui"},
        {"No", "Non"},
    }
}
```

```
        {"Hello", "Bonjour"},  
    };  
}
```

The first time you generate code, you will be prompted for a class name and package name. You do not have to provide a package or the generated files, but you are required to provide a class name. This is also used as the filename of the generated code file. If you wish to change the class or package name before generating again, choose “Set properties...” from the Generate menu.

9.4 Using Resource Bundles

You may create resource bundles using the Resource Bundle Editor, save them and generate the appropriate class files. In order to use them in the Class Editor you will have to do two things:

1. Set a Code Expression for the String property.
2. Add the resource bundle to the class in the Class Editor.

9.4.1 Setting the String Value

You will need to use code expressions in order to make the generated code read the value from the resource bundle. To do this:

1. **Select the component in the containment hierarchy which has a String property.**
A Button or Label, for example.
2. **Display the Property Sheet and select the “text” property.**
3. **Choose “Code expression” from the option menu at the bottom of the Property Sheet.**
4. **Type the following into the Code expression text box:**

```
resources.getString("key")
```

where “resources” is the variable name of the resource bundle and “key” is the key in the resource bundle. Remember to type Return at the end of the line.

9.4.2 Adding a Resource Bundle to the Class

To add a resource bundle to a class which is using its keys, do the following:

1. **In the Class Editor, select the ResourceBundle object from the palette.**
2. **This appears in the invisible beans area.**
3. **Change the variable name of the resource bundle to the class name you provided in the Resource Bundle Editor.**
4. **Display the Property Sheet for the resource bundle and change the Object Initialization property to use the code expression:**

```
ResourceBundle.getBundle("ClassName")
```

where "ClassName" is the name of the class you have generated.

Resources fetched from a resource bundle do not affect the dynamic display. They do display correctly when the generated code is run.

The Project Window

10

10.1 Introduction

The project window, shown in Figure 10-1, provides an area where you can specify the files you wish to work on.

Note – *If you are running Visaj from within an IDE (Integrated Development Environment), the project window may not be accessible from Visaj as the IDE may wish to control file handling. The ability to make and edit projects should then be available from the IDE.*

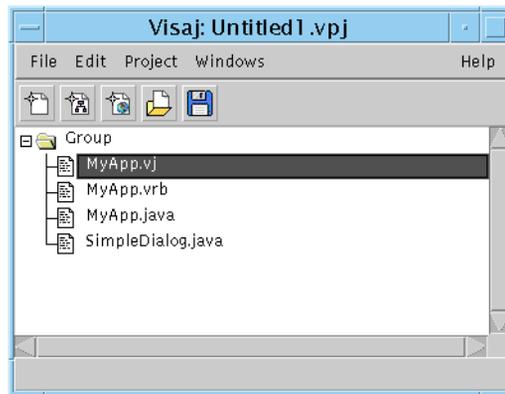


Figure 10-1 The Project Window

You may group these files together in a way which will help you develop your project; for example, you may wish to put all the files which control the user interface together in one group and all the files which handle

output in another. In this way, the files are easier to find and access. The project window is supplied as an extra level of convenience, but you do not have to use it. It is displayed by either selecting “New project” from the File menu of the Class Editor or by opening a project save file. Here is a list of the functions which can be performed from this window:

1. Create, save and open project files
2. Add and remove files
3. Add and remove groups
4. Create new classes
5. Create new resource bundles
6. Edit a class or a resource bundle

Each of these functions is described in the following sections.

See Appendix B, “Quick Reference”, starting on page 199, for a diagram and brief description of the toolbar buttons and menu items in each of Visaj’s windows, including the project window.

10.2 Creating, Saving and Opening Projects

You may create a list of filenames, arbitrarily grouped, by following the instructions in the sections below entitled Adding, Removing and Renaming Groups and Adding and Removing Files. This list of filenames constitutes your project. Save your project by choosing “Save” (or “Save As...”) from the File menu or by pressing the Save button on the toolbar. You will be prompted for a filename - by convention, Visaj project files have the suffix “vpj”. This is important because, when you open a file, Visaj uses the suffix to know what sort of file it is opening. Open a project file by choosing the “Open” item from the File menu or by pressing the Open icon on the toolbar.

10.3 Adding, Removing and Renaming Groups

The central area of the Visaj project window shows a tree of filenames. The filenames can be grouped together in any way which suits your way of working. This is purely for your convenience and has no impact on the

way Visaj's tools work. To add a group, select "Add Group" from the Edit menu. The group is added with the default name "Group". To change the group name, either select "Group name" from the Edit menu or double-click over the group name in the tree area. A dialog appears prompting you for the new name. Press "Ok" when you have entered the new name. To remove a group from your project, select "Remove group" from the Edit menu.

10.4 Adding and Removing Files

Add files to your project by selecting "Add file" from the Edit menu. You will be prompted for the filename. The filename is added to the currently selected group. Remove a file by selecting it and choosing "Remove file" from the Edit menu. This only removes the filename from your project, the file itself is not removed from its place on your disk.

10.5 Creating Files

You may create any of the following types of file from the project window:

1. Another project file
2. A resource bundle file
3. A class file

Create these files by either selecting the appropriate option from the pullright menu under "New" in the File menu or by selecting the relevant toolbar button.

10.6 Editing Files

Double-clicking over a filename in the tree area (or selecting a filename and then choosing "Edit" from the Edit menu) displays the editor appropriate for the selected file. The following editing tools are available:

1. **The Class Editor.** This is the most powerful part of Visaj, allowing you to create complete classes with methods defining your user interface, add functionality to the user interface and generate source code. The Class Editor is described in Chapter 4, "The Class Editor", starting on page 37.

2. **The Resource Bundle Editor.** This tool allows you to create resource bundles containing keys linked to values in any number of languages. This is described in Chapter 9, “Resource Bundle Editor”, starting on page 139.

10.7 *The Windows Menu*

The Windows menu appears in the Visaj project window and in Visaj’s editing tool windows. This menu provides a means of cycling between all currently open windows.

11.1 Introduction

Code can be generated from the Class Editor and from the Resource Bundle Editor. This section describes code generation from the Class Editor. Generating code from the Resource Bundle Editor is described in the *Generating Code* section on page 143.

11.2 How to Generate Code

To generate code for your classes, choose “Generate java...” from the Generate menu in the Class Editor or press the Generate button on the toolbar. The Directory Selection dialog is displayed, as shown in Figure 11-1. This allows you to specify in which directory you wish the generated code to appear.

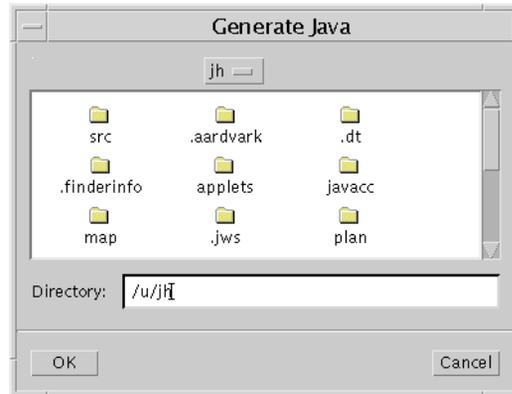


Figure 11-1 Directory Selection dialog

You should note that Visaj adheres to the requirements of the Java language by using the name of the class as the filename for the Java source file. If, therefore, you generate a Java source file, change the name of the class in Visaj and then regenerate, a Java source file is created with the new filename reflecting the new class name. In addition, if you have the “Update existing files” toggle in the Generate dialog selected, the previous Java source file is removed. This is to ensure that the design and the generated code are always in step. If you wish to retain the previously generated Java source file, make sure that the “Update existing files” toggle is not set before generating. In this case, however, any changes you may have made to the generated source are lost.

11.3 What is Generated

For each class you have created in Visaj, a java file is generated. Each class is placed in a file of its own using the name of the class with a “.java” suffix. Note that you can only specify one class per Class Editor save file so one “.vcl” file corresponds to one Java class file.

In addition, if you have specified an object initialization type of “Deserialization” in the property editor of any object, the file specified there is generated too. Object initialization is described in more detail in

the *Object Initialization* section on page 63. The file should have the suffix “.ser” to indicate that it is a serialized bean file. You are prompted to do this in the property editor.

11.3.1 Packing Frames

You should note that if you *do not* explicitly set the size property on a Frame (or JFrame from the Swing set of components), then the generated code includes a call to the pack() method. If you *do* set size property, then no call to the pack() method is generated.

11.4 Adding Your Own Code - Subclassing

Although you can edit the generated code directly to add your own code (this is explained in the following section), you may wish to consider subclassing the generated class file. This means that any code you add is in a separate file. It is easier to read than the generated code, which is sprinkled with protective comments. There is also no danger that the code you add will be accidentally lost when code is regenerated from Visaj.

Your new class would look something like this:

```
public NewClass extends VisajGeneratedClass {
    public void newMethod() {
        super.newMethod();
    }
}
```

There are a number of points to keep in mind when subclassing the generated code:

- The class defined in Visaj must be accessible - see the *Editing Properties of the Class* section on page 40 for information on this.
- If you have provided a package statement for the generated class, you should also provide one in your new class.
- If the generated class is in a different package from your new subclass, you will have to import the generated class.
- Remember to add the call to overridden methods, as in the example above, if the overridden method contains code.

- If you have wish to use a non-default constructor which you have provided in the superclass, you will have to add a constructor to your new class too.

11.5 Editing the Code

You may edit the generated code - for example, to add an interface to existing software. Make sure, when you do this, that you *do not* edit anything in sections which begin with the special comment `//v j+` and end with the special comment `//v j-`. Lines on their own which must not be changed or removed have the special comment `//v j=`. If your additions are made outside of these comments, they will be retained if you need to generate the code again.

11.6 Regenerating Code - Using the Update Toggle

When you regenerate your code, the Generate dialog contains a toggle labelled “Update existing file”, as shown in Figure 11-2. This will ensure that any changes you have made to the generated code will be retained.

If you have generated source code and then changed the class name, when you next generate code the “Update existing file” toggle has a more dramatic effect. If this toggle is set, the file is updated to use the new class name as the source code filename. It then appears as though a new file has been created and the old one removed. If you wish to retain the previously generated source file, make sure that the “Update existing file” toggle is not set before you generate. Of course, in this case you would lose any changes you may have made to the source file.

If you use the toolbar Generate button to regenerate code, the Generate dialog does not appear. Instead, code is regenerated immediately, retaining any changes you may have made to it.

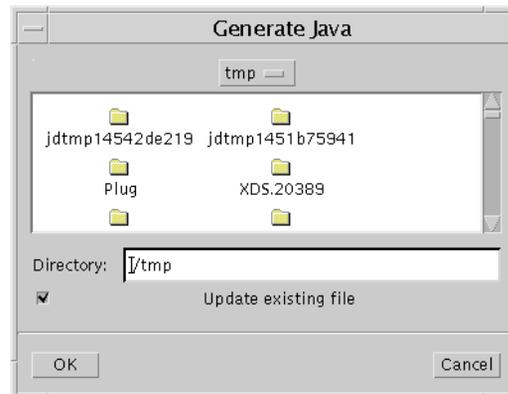


Figure 11-2 Directory Selection Dialog with Update Option

11.7 Example Code

The following is an example of code generated from a simple design, the containment hierarchy of which is shown in Figure 11-3. One event binding has been added to set the text of the textfield to “hello world” when button1 is pressed. The constructor has been designated “main method”. Otherwise, all defaults have been retained.

This code compiles and runs. Because it uses defaults, the layouts and sizes would need to be changed to create a better appearance.

```
//vj+ <VJ-PackageName>
//vj- <VJ-PackageName>

//vj+ <VJ-PackageInclude>
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Frame;
import java.awt.Button;
import java.awt.Panel;
import java.awt.Label;
import java.awt.TextField;
import java.awt.MenuBar;
import java.awt.Menu;
```

```
import java.awt.MenuItem;
//vj- <VJ-PackageInclude>

//vj= <VJ-DefineClasses>
//vj+ <VJ-BeginClassDef>
public class MyClass extends Object {
//vj- <VJ-BeginClassDef>

    //vj+ <VJ-DataMembers>
    protected Frame frame1 ;
    protected Button button1 ;
    protected Button button2 ;
    protected Panel panel1 ;
    protected Label label1 ;
    protected TextField textField1 ;
    protected MenuBar menuBar1 ;
    protected Menu menu1 ;
    protected MenuItem menuItem1 ;
    protected MenuItem menuItem2 ;
    protected Menu menu2 ;
    protected MenuItem menuItem3 ;
    //vj- <VJ-DataMembers>

    //vj= <VJ-Methods>

    //vj+ <VJ-BeginMethodDef>
    // Method# 1
    public MyClass() {
    //vj- <VJ-BeginMethodDef>

        //vj= <VJ-MethodCode>
        //vj+ <VJ-DefineAWTMembers>
        frame1 = new Frame();
        frame1.setTitle( "frame1" );
        button1 = new Button();
        button1.setLabel( "button1" );
        button2 = new Button();
        button2.setLabel( "button2" );
        panel1 = new Panel();
```

```
label1 = new Label();
label1.setText( "label1" );
textField1 = new TextField();
panell1.add(label1, null, -1);
panell1.add(textField1, null, -1);
menuBar1 = new MenuBar();
menu1 = new Menu();
menu1.setLabel( "menu1" );
menuItem1 = new MenuItem();
menuItem1.setLabel( "menuItem1" );
menuItem2 = new MenuItem();
menuItem2.setLabel( "menuItem2" );
menu1.add(menuItem1);
menu1.add(menuItem2);
menu2 = new Menu();
menu2.setLabel( "menu2" );
menuItem3 = new MenuItem();
menuItem3.setLabel( "menuItem3" );
menu2.add(menuItem3);
menuBar1.add(menu1);
menuBar1.add(menu2);
{
    String strConstraint;
    strConstraint = "Center";
    frame1.add(button1, strConstraint, -1);
    strConstraint = "North";
    frame1.add(button2, strConstraint, -1);
    strConstraint = "South";
    frame1.add(panell1, strConstraint, -1);
}
frame1.setMenuBar(menuBar1);
frame1.pack();
frame1.show();
//vj- <VJ-DefineAWTMembers>

//vj+ <VJ-EndAWT>
//vj- <VJ-EndAWT>
```

```
//vj+ <VJ-EventListenerClass>
class ActionListenerAdapter implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        if ( e.getSource().equals( button1 ) ) {
            textField1.setText( "Hello world" );
            return;
        }
    }
}
//vj- <VJ-EventListenerClass>

//vj+ <VJ-AddEventListeners>
button1.addActionListener(new ActionListenerAdapter());
//vj- <VJ-AddEventListeners>

//vj= <VJ-Classes>

//vj+ <VJ-EndMethodDef>
}
//vj- <VJ-EndMethodDef>

//vj= <VJ-Classes>

//vj+ <VJ-EndClassDef>
public static void main (String args[] ) {
    try {
        MyClass myclass = new MyClass();
    } catch ( Exception ex ) {
        ex.printStackTrace();
    }
}
//vj- <VJ-EndClassDef>
```

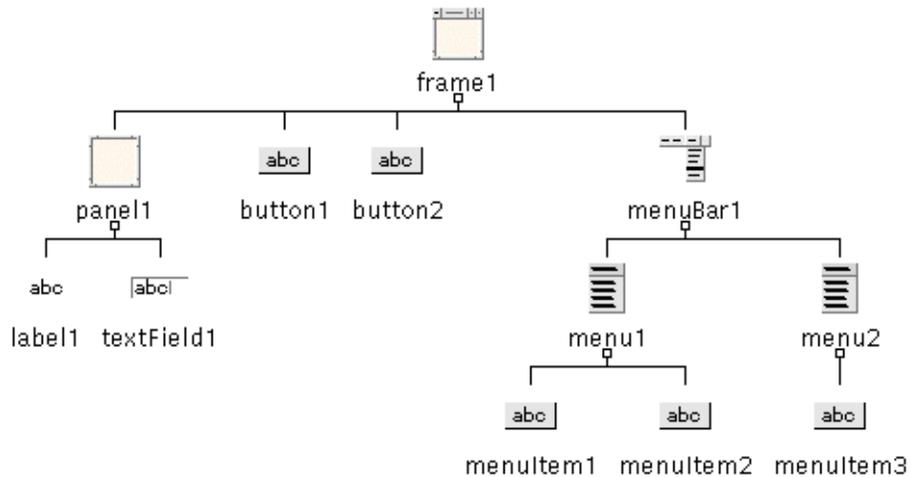


Figure 11-3 Hierarchy for Sample Code

11.8 Using the Diamond Components

If you have used any of the Diamond components in your application, you will have to make sure that the library or directory containing the Diamond class files is in your CLASSPATH. For the current release, the Diamond class files are in the file `diamonds.jar`, which is found in the Visaj install directory.

11.9 File Types on Apple Macintoshes

If you intend to run the generated code on an Apple Macintosh, you will need to set the file type of the generated code files. This is to enable them to be recognized by your chosen IDE. Visaj uses the following system property to decide which file type to set:

```
vj.macJavaFileType
```

This defaults to “CWIE”, which is the type expected by the IDE Code Warrior.

12.1 Integration with an IDE

To integrate Visaj with Java Workshop 2.0, use the Java Workshop integration package available from the same source as your Visaj¹. Once integrated, a Visaj icon appears on the Java Workshop toolbar allowing you to create new Visaj Class Editor save files (“.vcl”) or edit existing ones. When a Visaj Class Editor save file is added to a Java Workshop project, Java Workshop will automatically cause the appropriate Java source code files to be generated when you build the project.

Note – *If you have any problems with the Java Workshop integration, you may wish to check whether any extra information is listed in the README file supplied with the integration package.*

To use other IDEs, refer to the documentation supplied by the IDE vendor or simply add the generated “.java” source code files to the project. For details of the latest integrations with IDEs, visit the following website:

<http://www.ist.co.uk>

1. The release and installation notes supplied with Visaj provide details.

12.2 Palette File

If you wish to add components for use in Visaj, you can do so by defining a palette file or by loading a JAR file. Loading JARs is described in the *Loading JAR Files* section on page 60. The palette file is a text file containing the list of components to add, as full package names. You will also need to make sure that your CLASSPATH environment variable¹ points to the location of the new components. Both of these are described below.

Note – *Visaj is supplied with several pre-defined palette files. These can be found in the palettes directory beneath your Visaj install directory.*

This is an example palette file:

```
palette {
  title = "AWT"
  jar = "/components/jars/klg/jcchart200.jar"
  group {
    title = "KLG Chart"
    item {
      title = "JCChartComponent"
      class = "jclass.chart.JCChartComponent"
      help = "Create a JCChartComponent"
    }
  }
  jar = "/components/jars/jscape/java1/jars/Animator.jar"
  group {
    title = "JScape"
    item {
      title = "Animator"
      class = "COM.jscape.widgets.Animator"
    }
    item {
      title = "BaseTabbedPanel"
```

1. Environment variables are available on UNIX and Microsoft Windows. Other platforms, such as the Apple macintosh, use their own method of setting a CLASSPATH. Please refer to the relevant Java documentation for more information.

```

        class = "COM.jscape.widgets.BaseTabbedPanel"
    }
}
group {
    title = "TeaSet"
    item {
        title = "AnimatedButton"
        class = "tea.set.AnimatedButton"
    }
    item {
        title = "Animator"
        class = "tea.set.Animator"
    }
    item {
        title = "ArrowButton"
        class = "tea.set.ArrowButton"
    }
}
}
}
group {
    title = "KL"
    item {
        title = "DemoFrame"
        class = "jclass.chart.demos.DemoFrame"
    }
    item {
        title = "MyShape"
        class = "jclass.chart.demos.basic.MyShape"
    }
}
}

```

There are four levels in the palette file:

1. The name of the palette. For the current release, this will always be "AWT", as shown above.
2. The name and location of the jar file to load for some or all of the components in the palette. This is optional. If no jar file is specified, Visaj checks your CLASSPATH.

3. The “group”. In the above example, there are three groups: TeaSet, KL and JScale. All items inside the group will appear together and labelled appropriately on the component palette.
4. The “item”. This level identifies the individual component.

There are two further fields at the “item” level relating to components: “icon” and “help”. “icon” is the name of the icon to display on the palette (a “.gif” suffix is assumed) and “help” is the text to display in the status line. These fields can be seen in the following extract from the built-in palette file for Visaj:

```
    item {
        title = "Panel"
        class = "java.awt.Panel"
        icon = "Panel"
        help = "Create a Panel"
    }
    item {
        title = "ScrollPane"
        class = "java.awt.ScrollPane"
        icon = "ScrollPane"
        help = "Create a ScrollPane"
    }
}
```

12.2.1 Using VISAJOPTS

When you have created your palette file, you need to tell Visaj where it is. There are three ways of doing this:

- Using the VISAJOPTS environment variable
- Using the “Merge palette...” item from the Palette menu in the Class Editor
- From the command line

Here is an example of how the VISAJOPTS environment variable is used:

```
setenv VISAJOPTS -Dvj.AWTPalette.file=/u/me/paletteFile
```

where `/u/me/paletteFile` is the full pathname of your palette file. The syntax of the line will depend on the shell or operating system you are using.

Microsoft Windows NT

If you are using Microsoft Windows NT, make sure that you use the backslash character ('\') instead of the forward slash character ('/') when specifying a pathname, as in the example above.

Microsoft Windows 95

It is not possible to use the VISAJOPTS environment variable on Microsoft Windows 95. If you wish to set up Visaj so that a particular palette is always loaded on that system, you will need to create a batch file containing a command line configured as described in the following section.

12.2.2 Using the Command Line

You can tell Visaj that you wish to use merge your own palette file into the default palette by using the following system property either from the command line or in a startup script. Here is an example:

```
visaj -Dvj.AWTPalette.file=/u/me/paletteFile
```

This merges the palette file `/u/me/paletteFile` into the palette you have selected for starting Visaj - AWT or Swing. Use `vj.CommonPalette.file` to merge a palette file into both the AWT and Swing palettes.

12.2.3 CLASSPATH - Locating Classes

The CLASSPATH environment variable¹ must contain the full pathnames of the package containing the class files of any new components you wish to have on your component palette. This may be a ".jar", a ".zip" file or the full pathname of a directory containing the class files. By default, Visaj automatically loads the Java AWT components and the IST Diamonds.

1. Environment variables are available on UNIX and Microsoft Windows. Other platforms, such as the Apple macintosh, use their own method of setting a CLASSPATH. Please refer to the relevant Java documentation for more information.

12.3 Pre-defined Palette Files

Supplied with Visaj are some pre-defined palette files which load various components from different vendors. These reside in the palettes directory which can be found in the Visaj install directory. The README file in the same directory lists which components are loaded by each of the palette files supplied. To use these palette files you will need to set your CLASSPATH environment variable to point to the class files (or jar file) supplied by the component vendor.

12.4 Merging Palette Files

You may merge a palette file into your existing Class Editor window by selecting “Merge palette...” from the Palette menu. Specify the name of the palette file where prompted. You must make sure that your CLASSPATH environment variable contains the locations of the class or jar files for the classes in the new palette file. See also the *Loading JAR Files* section on page 60 for another way of adding objects to your palette.

12.5 Use Swing Palette

Type

```
visaj -swing
```

to have the Swing palette supplied with Visaj loaded at start-up. You must make sure that the Swing jar file is in your CLASSPATH before starting Visaj. See the *Loading Swing Components* section on page 90 for more information.

12.6 Visaj Options

The following sections detail the Visaj options (or system properties) which alter some aspect of Visaj’s appearance or behavior. The sections are grouped according to the type of change effected by the option.

These options are passed as arguments to Visaj using “-D” and then the property (with no spaces), followed by any arguments which may be required. This is shown in the following example:

```
visaj -Dvj.menuFont=Helvetica-italic-
```

Some of the options do require an argument, such as a filename or a value of true or false. Others are simply switches and cause a change in behavior just by being passed to Visaj.

If you pass in a badly formatted string or a type of property which does not exist it is ignored. No error message is displayed.

12.6.1 Fonts

The following system properties allow you to change the fonts Visaj uses:

1. `vj.menuFont`
2. `vj.windowFont`

`vj.menuFont` allows you to specify the font that should be used for menus. `vj.windowFont` lets you change the font used elsewhere in the windows. On some platforms, the default fonts are not very suitable.

The font should be specified in one of the following ways:

1. `<fontname>-<style>-<pointsize>`
2. `<fontname>--<pointsize>`
3. `<fontname>-<style>-`
4. `<fontname>`

Note - *The trailing hyphen (-) in number 4 is required.*

12.6.2 Palettes

Use `vj.AWTPalette.file` to specify a palette file to merge into either the AWT or Swing palette - whichever you have selected to appear on startup. Use `vj.CommonPalette.file` to specify a palette file to merge into *both* the AWT and Swing palettes.

12.6.3 Design Time Flag

To set the design time flag for beans, use `vj.beanDesignTime`, setting this to "true", like this:

```
visaj -Dvj.beanDesignTime=true
```

This will cause all beans to operate in design mode.

12.6.4 *Temporary Directory*

By default, Visaj uses `/tmp` as the directory in which to compile event bindings so that they can be used in the dynamic display. If you wish to change this, use `vj.tmpDir`, setting it to your preferred directory. For example:

```
visaj -Dvj.tmpDir=/u/me
```

12.6.5 *Current Directory for Code Generation*

By default, Visaj shows the current working directory when the Generate dialog is displayed. You can override this by using the `vj.cwd` property.

12.6.6 *Swing Conversion*

The option `vj.convertPackages=true` allows you to convert swing package names from the older form of `com.sun.java.swing.<class>` to the newer `javax.swing.<class>`. Since the default behavior is the same as passing “false” using this option, you do not

12.6.7 *Plug In File*

The option `vj.pluginFile` tells Visaj where to find a plug-in. By default, Visaj looks for plug-ins in the `PlugIns` directory in the Visaj installation directory.

12.6.8 *Migration from Java WorkShop*

If you wish to use Visaj with your `.gui` files created with Java WorkShop, start Visaj with the option `vj.JWS=true`. An extra menu is displayed in Visaj’s menubar, labelled “Java WorkShop”. This menu contains the commands required to convert a `.gui` file.

Remember that the shadow runtime needs to be in the classpath.

12.6.9 Scope of Beans

The `vj.variablesDefaultToInstanceVars` option controls whether beans added to a design are local variables or instance variables. The default value for this option is true, which means that beans are normally added as instance variables.

12.6.10 Use System Colors

The option `DTCOLORS` causes Visaj to use the system colors.

12.6.11 Switch off JIT

Passing the `java.compiler=none` option via Visaj switches off the JIT compiler. This produces an error message which can be safely ignored.

13.1 Introduction

This chapter incorporates some points which you may find useful when using Visaj along with some issues which could be termed “troubleshooting”. If you become “stuck” or confused, scan this chapter to see if any relevant information is provided. Check here also for advice on using Visaj. The tips and hints are divided into the following categories:

- User Interface
- Event Bindings
- Loading X-Designer Save Files
- Palette Configuration
- Layout
- Generated Code

13.2 User Interface

How do I make a “main” method?

Tell Visaj which method should be the “main” method of your application by selecting the method in the Class Editor and then choosing “Main method” from the Method menu. A small star next to the method reminds you which has been designated “main”. You do not have to mark any of the methods as “main”, but remember to add one somewhere in your application!

Further information:

- The *Class Structure View* section on page 38 for information on seeing and editing the methods in your class.
- The *Main Method* section on page 49 for more information on making a method “main”.

☞ *The hierarchy has disappeared after looking at the Event bindings or method signature pages.*

The area of the Class Editor window which contains editors for the methods in the classes is organized as a *tabbed panel*. This is like a stack of cards with a tab on each card showing its name (or function in this case). Selecting a tab brings the associated “card” to the top. If you have been looking at the Event bindings or method signature “card”, you will have to select the “Bean creation” tab to bring the hierarchy building “card” to the front again.

Further information:

- Chapter 5, “Beans View”, starting on page 55, explains all the method editors.
- Chapter 3, “Visaj Tutorial”, starting on page 13, leads you through a real working example to familiarize you with Visaj.

☞ *I want to reparent some child objects*

By using a combination of Cut/Clear/Paste you can reparent whole groups of objects. Here is an example:

1. **You have created a hierarchy of Frame-Splitter-{Button, Toggle, TextField}, as shown in Figure 13-1.**

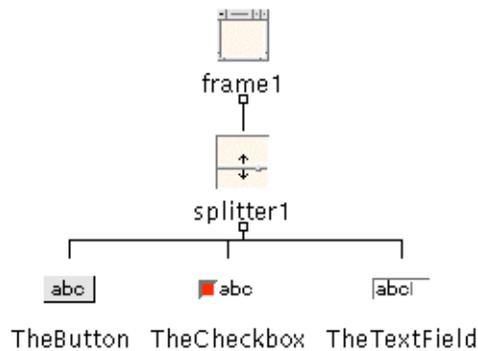


Figure 13-1 First Parenting

- 2. Select the Button, Toggle and TextField.**
- 3. Press the Cut button on the toolbar (or select from the Edit menu).**
- 4. Select the Splitter in the hierarchy.**
This is a Diamond component.
- 5. Press the Clear button on the toolbar (or select from the Edit menu).**
- 6. Make sure the Frame is selected and press the Paste button on the toolbar (or select from the Edit menu).**

7. You have now changed to a structure of Frame-Panel-{Button, Toggle, TextField}, keeping the same child objects, as shown in Figure 13-2.

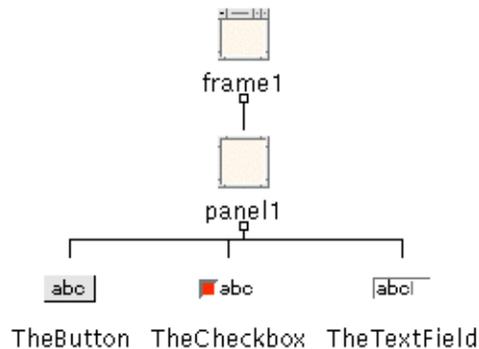


Figure 13-2 Second Parenting

Further information:

- The *Building Hierarchies* section on page 58 describes how to create and edit containment hierarchies.
- Chapter 3, “Visaj Tutorial”, starting on page 13 leads you through a working example to introduce the major features of Visaj.
- See the *Splitter* section on page 195 for information on the Splitter Diamond component.

☞ *How do I manage lots of classes for my application?*

The Class Editor works with one class at a time. The chances are that your application will have more than one class in it. To keep track of them all and to make them easily available for editing, do one of the following:

1. Use the Visaj project window. Using this you can keep a list of all the classes in your application and simply double-click over them to edit them in the Class Editor
2. Use an Integrated Development Environment (IDE). IDEs are commercial packages for organizing, creating, compiling and debugging Java application. Some IDEs can be configured to work with Visaj's save files.

Further information:

- Chapter 10, “The Project Window”, starting on page 147, describes the Visaj project window.
- The *Integration with an IDE* section on page 161 explains how to use an IDE in conjunction with Visaj.

☞ *How do I use CheckboxGroups?*

The CheckboxGroup, whose palette icon is shown in Figure 13-3, is an invisible bean. As with all invisible beans, you must have nothing selected in the design area to add it to your design. It is then placed at the top of the design area, alongside the roots of hierarchies.



Figure 13-3 CheckboxGroup Palette Icon

The CheckboxGroup provides a means of *grouping* Checkboxes and giving them *radio button behavior* (only one in the group can be “set”). To link one CheckboxGroup to a group of Checkboxes, do the following, assuming that you have already added some Checkboxes and a CheckboxGroup to your design:

1. **Select the Checkboxes and display the Property Sheet.**
2. **Select the “checkboxgroup” property.**
3. **In the property editor at the bottom of the Property Sheet, select the “Variable name” option.**
4. **Type the variable name of your CheckboxGroup into the text field.**

Although you will not see the radio behavior in the dynamic display, the generated code will display such behavior.

Further information:

- See the *Using CheckboxGroups* section on page 76 for another description of this issue.

- See the *Properties* section on page 60 for a description of object properties, including how to use the Property Sheet.

☞ ***I can't see the objects in my design properly, particularly when I add invisible beans***

The Class Editor window is divided into several areas. You can alter the amount of window space used by the containment hierarchy and by the class structure editor by:

- Using the splitter to resize the panels. Do this by positioning the pointer over the bar between the panels, pressing the mouse button and moving the divider. A diagram is shown in Figure 13-4.
- Pressing the “Show Method Editor only”/”Show both” buttons on the toolbar.

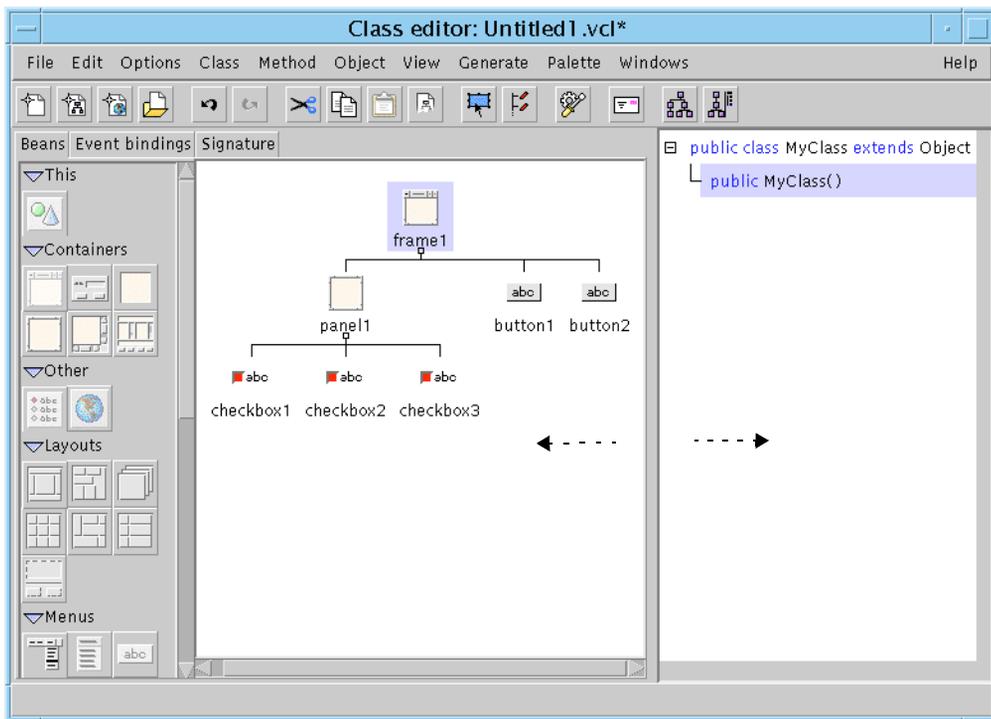


Figure 13-4 Demonstration of Splitter Panel Divider

Further information:

- Chapter 4, “The Class Editor”, starting on page 37, describes the Class Editor window and functions.
- The *Class Editor Toolbar Buttons* section on page 199 tells you which buttons are which on the toolbar.

 ***I can't find the other windows in my Visaj session***

The Windows menu, available from all of Visaj's editors, lists all open windows in your current Visaj session. Selecting one of these menu items brings the window to the front.

Note – *On some window systems, if the selected window is iconized the icon is brought to the front but the window is not opened.*

Further information:

- The *Windows Menu* section on page 52 describes the whole Windows menu.

 ***I have added a file selection dialog to my design but it does not appear in the dynamic display***

By default, Visaj sets the “visible” resource for file selection dialogs to “false” because, being modal, they cause confusion when they appear as they have to be dismissed. you can make a file selection dialog appear either by explicitly setting the “visible” resource or by setting up an event binding which causes it to appear.

Further information:

- Chapter 6, “Event Bindings”, starting on page 79 describes event bindings and how to create them.
- The *Properties* section on page 60

☞ ***How do I add components to the base component?***

To define methods which add components to the class, add “this” to the method design when the class is a subclass of `java.awt.Component` or `MenuComponent`. “this”, in such a case, refers to the class itself so any components added to it are added to the base component (the class itself).

Further information:

- See the “*this*” section on page 42 for more information on using “this”.
- See the *Editing Properties of the Class* section on page 40 for more information on changing the superclass.

☞ ***Visaj won't exit***

If you try exiting visaj and nothing happens, it may be because you have some iconized class editors that have put up a dialog asking if you want to save changes, but the dialog cannot be seen because the editor is iconized.

☞ ***Visaj is not responding/behaving strangely. Can I get any feedback on what is happening?***

Select “Java Console...” from the Options menu. If any exceptions have been generated, they are displayed in this window.

13.3 Event Bindings

☞ ***How can I set up an action to be performed when the Close button in a Frame (or Dialog) is pressed?***

To set up an event binding on the close button of a Frame or Dialog, display the Event Binding editor and select the Frame or Dialog as the “Source”. Next, select “windowClosing” from the list of “window” actions in the “Type” list.

To exit the application, select the class (default “MyClass”) as the “Object” and a class method previously defined by you. In the generated code, you would then add the following line to your method:

```
System.exit(1);
```

To hide the window, select the frame (or the dialog) as the “Object” and then the “hide” method.

Dragging a line between two components doesn't display the Event Bindings Editor

Holding down the control key while dragging between two components causes the Event Bindings Editor to be displayed, primed with the two components. If, however, the control key is released at any time *before* the mouse button is released, the Editor is not displayed. Instead, the second component is simply selected in the design hierarchy,

Further information:

- *Displaying the Editor* section on page 80. This section describes the different ways of displaying the Event Binding Editor including dragging across the containment hierarchy.

Adding an event binding causes Exceptions and does not work

If you are trying to set up an event binding using Beans from a compressed JAR file, you may find that the event binding is not created and Exceptions are printed in a terminal window. This is because the compiler which Visaj is using “behind the scenes” cannot access compressed listener or event objects.

Further information:

- Chapter 6, “Event Bindings” for information on creating event bindings.

13.4 Loading X-Designer Save Files

☞ *When I try to load my design, one of the file dialogs in it appears, but then Visaj freezes up.*

File selection dialogs are modal, therefore if you have set the “visible” resource for the file selection dialog to “true”, Visaj will freeze when it displays the dialog in its dynamic display. This happens on import as well as with normal loading. If you cancel the file dialog when it appears, Visaj will continue loading. By default, the “visible” resource is set to “false”.

Further information:

- The *Importing X-Designer Save Files* section on page 50 describes how to load files created in X-Designer into Visaj.

☞ *Visaj seems to have confused the title of an MWT FramedPanel with its contents - or - Visaj is not displaying the correct heading ('error', 'information', etc.) on my dialog template.*

You can specify which of a FramedPanel’s children is its title child by using a property, and you can specify an IconMessagePanel’s icon in a similar manner. However, properties of type Component can only be specified using a code expression property setting in Visaj.

These settings do not take effect in the dynamic display. The dynamic display, therefore, may use the wrong child as title or icon when displaying the FramedPanel or IconMessagePanel. Generated code, however, will use the correct child.

Further information:

- The *Importing X-Designer Save Files* section on page 50 describes how to load files created in X-Designer into Visaj.

☞ *My Motif XmRadioBox has disappeared on Visaj*

Motif designs which use XmRadioBoxes, when brought over to Visaj, will use Checkboxes inside Panels. You will have to edit your design to add radio-box behavior.

Further information:

- The *Using CheckboxGroups* section on page 76 describes how to add Checkboxes to CheckboxGroups.
- The *ButtonGroup* section on page 102 describes how to use the Swing ButtonGroup component to add radio-box behavior to your application.

☞ *A message “Unknown class <name>_user” is displayed when loading the X-Designer generated file.*

You may see such a message if your X-Designer design was captured (using XD/Capture) in Java mode. In some circumstances XD/Capture overrides some of the classes in the design. You should replace the class mentioned in the error message with <name>_c.

☞ *The design seems to have loaded, but nothing is displayed.*

Check whether the visible property on the topmost frame has been set to false. This sometimes occurs in designs captured with XD/Capture.

13.5 Palette Configuration

☞ *How do I add components onto the Visaj palette?*

Visaj loads the Java AWT components and the Diamond components by default onto the component palette. If you wish to add components from other vendors or your own homemade components, you can do one of the following:

- Select “Load Jar file...” from the Palette menu to load extra components contained in a JAR file.

- Select “Merge palette file...” from the Palette menu to load extra components described in a palette file.

Merging in a palette file also requires that you set your CLASSPATH environment variable so that Java can locate the new classes.

Further information:

- Step 2 on page 14 describes how to add the extra components required for the tutorial.
- The *Loading JAR Files* section on page 60 explains how to load beans from JAR files.
- The *Opening Other Palette Files* section on page 59 describes the merging of palette files.
- The *CLASSPATH - Locating Classes* section on page 165 explains more about setting the CLASSPATH environment variable.

 ***When I do “Merge Palette”, no components are added to my palette***

If a message is displayed saying that Visaj cannot handle the classes in the palette file you are trying to merge in, check that you have set your CLASSPATH environment variable so that Java can locate the classes mentioned in the palette file. You can set up the CLASSPATH to point to a JAR file containing the classes or to a directory with the classes in it.

Further information:

- The *Opening Other Palette Files* section on page 59 gives more information on merging palette files.
- The *CLASSPATH - Locating Classes* section on page 165 explains some more about the CLASSPATH environment variable.

 ***No Beans have been added to my palette after loading a JAR file***

The following are some possible reasons why Java Beans have not loaded from a JAR file:

1. There is no manifest file in the JAR.
2. There is a manifest file, but no mention of the Bean to be loaded.

3. There is a manifest file which refers to the Bean, but the entry for that Bean is missing the line:

```
Java-Bean: true
```

Further information:

- The *Loading JAR Files* section on page 60 gives some more information on the loading of JAR files.
- Step 2 on page 14 from explains how to load a JAR file for the tutorial.
- Look at the books mentioned in the *Books on Java* section on page 209 to find out more about JAR files.

☞ ***On Microsoft Windows, my specified palette file has been ignored or cannot be found***

If you have specified a palette file either on the command line or using VISAJOPTS, make sure that there are no spaces in the path. This is especially noticeable on Microsoft Windows where you may be referring to files in the “Program files” directory. In such a case, try substituting the DOS-style “Program~1”, as in the following example:

```
-Dvj.AWTPalette.file=D:\Program~1\Visaj\palettes\JFC11.palette
```

13.6 Layout

☞ ***How do I specify no layout at all for a container?***

In Visaj, containers are always given a default layout. You can choose to have no layout for a container by choosing “null” for the container’s layout property in the Property Sheet. You may still use the Layout Editor, which will allow you to position the container’s children absolutely (i.e. with no constraints or resize behavior). The children can also be resized in the Layout Editor in this case.

Further information:

- The *Null Layout (Absolute Positioning)* section on page 71 also covers this issue.

☞ ***When using the Null layout, the child objects have disappeared***

When you have chosen to have no layout for a container (choosing “null” for the layout property), the Layout Editor allows you to move and resize the children but cannot show the insets of the container. The inset is a border around the edge. This means that the child components can “disappear” underneath the inset in the dynamic display window. Move the components in the Layout Editor so that they are visible in the dynamic display window, which updates immediately.

☞ ***(Motif only) When using the Null layout, the generated application takes up the whole screen***

If, having set no layout (“null”) on the root window of your application, the generated and compiled application takes up the whole screen, specify an explicit size for the frame by setting the “size” property in the Property Sheet for the frame.

Note – *This only applies to applications running under Motif.*

☞ ***How do I simply place the children of a container at a particular location?***

If it is important that you position the children of a container at an absolute location so that they stay there regardless of whether the container is resized, use the Null layout. Display the Property Sheet for the container, select the layout property and choose “null” from the option menu at the bottom.

The major usefulness of the Null layout is when the child components must fit in with a background image (a button over a picture for example).

Further information:

- The *Properties* section on page 60 describes the Property Sheets and how to use them.

Can I undo and redo actions in the Layout Editor?

You can. Use the appropriate buttons in the Class Editor window. The Class Editor will undo and redo the last actions regardless of whether they took place in the Layout Editor or directly in the Class Editor window.

Further information:

- The *Class Editor Toolbar Buttons* section on page 199 helps you to find the toolbar button you are looking for.
- The *Class Editor Menu Items* section on page 200 lists the function of each item in the menus.

How do I edit the “insets” of a container?

To edit the insets in a GridBagConstraints object, use the Layout Editor. The area on the right of the Editor window allows you to edit the GridBagConstraints for the child component which is selected in the area on the left. Enter values in the “left”, “Right”, “Top” and “Bottom” fields to change these insets.

There is no direct way of editing the insets of an AWT container. To do this, you would have to create a subclass of the container and then override the `getInsets()` method.

13.7 Generated Code

The generated application does not run

Make sure that you have a “main” method. If you have not designated a method in one of your classes as the “main” and you have not linked in your own “main” method, none of your code will be called.

Further information:

- See the *Main Method* section on page 49 for details of how to specify a “main” method from within Visaj.

☞ *I generated code using one class name and regenerated using another. Where is the first Java source file?*

Visaj always generates Java using the name of the class for the name of the Java source file. This is a requirement of the Java language. If you have changed the name of the class in Visaj, Visaj treats this as an update and generates to a new file using the new class name. If, however, the “Update existing files” toggle in the Generate dialog is set, the previously generated source file is removed. This is to ensure that the design and the generated code are always in step. If you wish to retain the previously generated file, unset the “Update existing file” toggle before generating. If you have made any changes to the generated source file, unsetting the “Update existing files” toggle and regenerating will lose them.

☞ *When I try to run my generated application, it stops with a `NullPointerException` at a line where my generated code is trying to access an image file*

In the generated code, image files are loaded as resources using the class loader. The string typed into the “Runtime resource path” text field is passed directly to the class loader method `getResource`. By default, Visaj generates code which assumes that the images you specify are in the same directory as the class file which is accessing them. If you have copied your class files to a directory matching your package, make sure that you have copied your image files too. Similarly, if you have any serialized object files (ending in “.ser”), you will also need to copy these.

Further information:

- See Appendix C, “Bibliography” for suggested books on Java. These provide more information on the class loader and its `getResource` method.

☞ *I have copied my generated Java files to another platform and recompiled but the compiled application won't run*

Make sure that you have also copied any serialized object files (ending in “.ser”). These files are generated by Visaj for objects which you wish to be initialized by deserialization and for objects which have been customized. This applies not only when copying to another platform but also to another directory on the same platform, such as when placing the class files into their package location. When moving your Java or class files around, make sure that you have also taken any image files used by your application.

Further information:

- The *Object Initialization* section on page 63 provides more information on this subject.
- The *Customizers* section on page 68 explains more about customizers and when you may wish to use them.

☞ *With Diamonds in my design, I can't compile the generated code*

If you are using any of the Diamond components in any part of your application, make sure that you have specified the Diamonds jar file in your CLASSPATH environment variable¹ before compiling. If you have not done so, a message saying that the class cannot be found in the import will be displayed.

Further information:

- See the *Using the Diamond Components* section on page 159 for more information.
- See the *CLASSPATH - Locating Classes* section on page 165 for information on CLASSPATH.

1. Environment variables are available on UNIX and Microsoft Windows. Other platforms, such as the Apple macintosh, use their own method of setting a CLASSPATH. Please refer to the relevant Java documentation for more information.

How do I add my own code to the generated code?

There are essentially two ways of adding your own code to the generated code:

1. Subclassing the generated class
2. Editing the generated code

The first option, subclassing the generated class, is by far the cleanest way and the easiest to maintain. Your new class would look something like this:

```
public class NewClass extends GeneratedClass {
    public void generatedMethod() {
        super.generatedMethod();
        ...
    }
}
```

Points to remember when doing this are:

- Make sure that the class you created in Visaj has public access, otherwise you will not be able to subclass it in a separate file.
- Remember to call the method in the superclass if you have added any code in there (it doesn't hurt if you haven't).
- If you have declared a package name in the generated class, you will have to do the same here.
- Import the generated class if it is in a different package.
- Remember to follow the Java rules for constructors if you have created a non-default constructor in your superclass.

If you decide to edit the generated code directly, make sure that you pay attention to the special comments put in by Visaj, otherwise your changes may be lost when code is regenerated.

Further information:

- See the *Adding Your Own Code - Subclassing* section on page 153 for more information on using subclasses to add your own code.
- See the *Editing the Code* section on page 154 for more information on the special comments in the generated code.
- See the *Adding Your Own Code* section on page 34 for an example of how to add your own code using a subclass.

- See the *Editing Properties of the Class* section on page 40 to find out how to change the access of a class.
- See the *Books on Java* section on page 209 for the details of books on Java.

Diamond Components

A

The IST Diamonds are a collection of useful components and layout classes that have been designed with the needs of the developer in mind. Each class has been made as lightweight as possible, while still delivering maximum functionality. They have been constructed to make them easy to re-use with considerable thought given to the methods provided to allow the developer to control the class. All classes adhere to the JavaBeans conventions. All classes are documented to give javadoc documentation.

A.1 The Diamond Components

IST's Diamonds components are:

- **Buttons and labels**
See the *Buttons* section on page 192.
- **FramedPanel**
See the *FramedPanel* section on page 194.
- **Separator**
See the *Separator* section on page 194.
- **Controllers: ProgressBar, Slider, Meter and Knob**
See the *Controllers: Knobs and Meters, Sliders, and Progress Bar* section on page 194.
- **Splitter**
See the *Splitter* section on page 195.
- **Statusbar**
See the *StatusBar* section on page 195.

- **SuperGrid**
See the *SuperGrid Layout* section on page 195.
- **DlogTemplateLayout**
See the *DlogTemplateLayout* section on page 195.
- **Book**
See the *Book (TabbedPanel)* section on page 196.
- **Toolbar**
See the *ToolBar* section on page 197.
- **Bezel Panel**
See the *Bezel Panel* section on page 197.

A.1.1 Buttons

Diamonds provide the following buttons:

- ArrowButton
- DrawnButton
- PolygonButton
- RoundButton
- FlexiButton

All buttons provide label, button, toggle or multi-state behaviour, and use the JDK 1.1 event model to fire action events when the button state changes. The events can be fired when the button is pressed or released. The buttons also support tips.

ArrowButton

The simplest button in the Diamonds set is the arrow button which just draws a triangle, representing an arrow. This can be set to point in one of eight different directions:

1. NORTH
2. SOUTH
3. EAST
4. WEST

5. NORTHEAST
6. NORTHWEST
7. SOUTHEAST
8. SOUTHWEST

DrawnButton



The `DrawnButton` is a button whose `paint` method can be overridden to put an image on the button. We provide some subclasses that have already done this. It can have shadows on any or all of its sides, and these can be round or square.

Note – *With the Diamonds, a label can be created by setting the behaviour of a `DrawnButton` to `LABEL`.*

PolygonButton



The `Polygon Button`, as the name suggests, allows the developer to produce a button in any polygon shape. It is a `DrawnButton` subclass, shaped as an arbitrary polygon. The `Diamonds` code works out all the appropriate shading to give the natural 3D appearance.

RoundButton



`RoundButton` is similar to a `PolygonButton`, but has code to use an `Oval` as its polygon.

FlexiButton



The `FlexiButton` class can display images, text, or both together. It also supports separate appearances for the pressed, released, disabled and enabled states. This is the button to use for tabs in a `Book` component, as detailed in the *Book (TabbedPanel)* section on page 196.

A.1.2 *FramedPanel*



The FramedPanel can be used to draw a frame around groups of related components, with an optional title. A variety of layouts and appearances are provided.

A.1.3 *Separator*



A simple but useful class, this lets you add separators to your visual layout. Full customization of the separator is supported.

A.1.4 *Controllers: Knobs and Meters, Sliders, and Progress Bar*

Controllers are UI elements which allow the developer to display numerical values in a graphical manner, and can allow the user to input data using the mouse. There are two types; sliders and AngleControllers.

Sliders



Sliders display the information as a thumb, which is placed somewhere along a track; the position of the thumb represents the value. There are two types of slider; the lightweight version, which is provided as a diamond, and the somewhat more complex MWT version, which mimics the Motif Scale widget in appearance.

Progress Bar



The ProgressBar is a subclass of Slider and is designed to show how far through a particular process a program might be; for example, a web browser might use it to show how far it has progressed with a download. The ProgressBar can also take input; its only true difference is in its appearance, which has been optimized for output.

AngleControllers (Knob and Meter)



AngleControllers show a value in terms of an angle; a Knob shows it in the form of a control which can be 'turned' by the mouse, much like the volume control on many music systems. A Meter is designed more for

output (although it can also be used for input); it takes the form of a needle which points to a value on a semicircular scale (rather like, say, the dial on an analogue ammeter).

A.1.5 Splitter

A splitter is a panel which lays out its children in a grid. Between the children are 'sashes', which can be dragged around. Dragging them allows the user to determine how much space is given over to each child. The cursor changes when over a sash to hint to the user that this can be done.

Maximum and minimum sizes can be set for the height of each row and for the width of each column. Splitters can be nested inside other splitters to allow quite complex user-resizable layouts.

A.1.6 StatusBar

The StatusBar is simply a Panel which draws a shadow around itself and around each of its children. The example below uses it in conjunction with a SuperGrid to provide a simple status bar.

A.1.7 SuperGrid Layout

The SuperGrid class provides the ideal layout control for most dialogs. What most developers need is a cross between the AWT Grid and the AWT GridBag layout manager. The SuperGrid provides the ability to layout simple tabular arrangement of components. Individual columns expand so that they are the width of the widest component in the column and individual rows expand so that they are the height of the tallest component in the row. SuperGrid provides fill and alignment properties so that simple dialogs containing, for example, labelled textfields can be laid out so that the labels are right aligned against the textfields and the textfields expand to fill unfilled areas.

A.1.8 DlogTemplateLayout

The DlogTemplateLayout gives you a default layout suitable for simple dialogs such as those used for displaying messages or errors. Any buttons added to a container with this type of layout are added in a horizontal line

at the bottom of the container. If you add another container, such as a Panel, it is put at the top. A separator is provided “free” and placed above the line of buttons. Figure 13-5 shows an example hierarchy using `DlogTemplateLayout` along with the associated dynamic display.

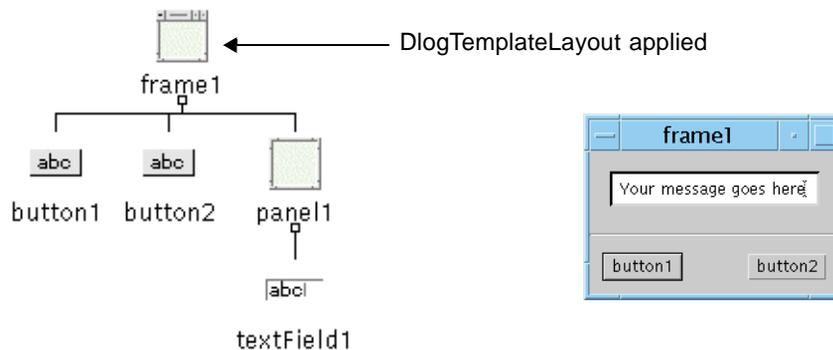


Figure 13-5 `DlogTemplateLayout` Example

Note – If you use the `DlogTemplateLayout` in your application, you will need to add the `mwt.jar` file to your `CLASSPATH`. This jar file is found in the Visaj install directory.

A.1.9 Book (`TabbedPanel`)



The `TabbedPanel` class provides an easy-to-use tab control component for Java. It allows you to dynamically add and remove tabs, position them along the top, sides or bottom, place images, text, or combined images and text on the tabs. The `TabbedPanel` treats `FlexiButtons` as tabs and anything else as pages. These can be added in any order - the n^{th} tab will match the n^{th} page. If there are more tabs than pages, all extra tabs will show the last page.

Note – The `Book` component expects to have `FlexiButtons` as tabs. If these are not used, the `Book` cannot function as described above. See the `Buttons` section on page 193 for more details on that component.

Possible uses:

- Preference dialogs
- General paged dialogs

A.1.10 *ToolBar*

The `ToolBar` is a panel which draws a border around itself. Future releases will have docking and drag-and-drop support.

A.1.11 *Bezel Panel*

`BezelPanel` is a container with a bezzelled frame. The shadow thickness and insets can be adjusted via the Property Sheet. You may also provide a pixmap image to use for the frame edging itself.

A.2 *Using the Diamond Components*

When you use the Diamond components in your application, make sure that the library or directory containing the Diamond class files is in your CLASSPATH. The Diamond class files are in the `diamonds.jar` file, which is found in the Visaj install directory. This jar file must be in your CLASSPATH.

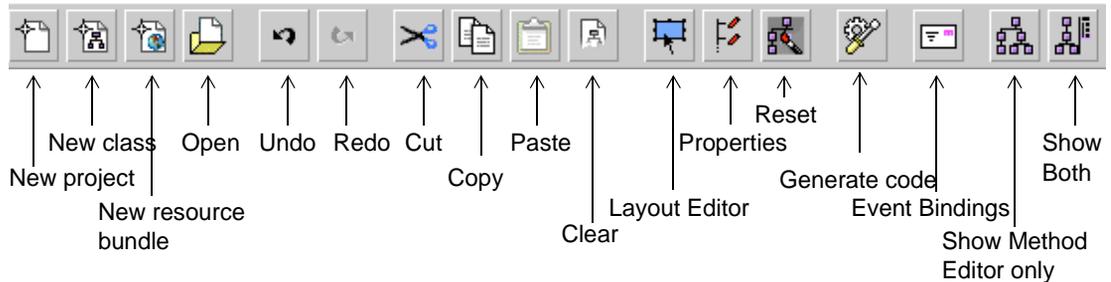
For `DlogTemplateLayout` you will need the `mwt.jar` file, which is also found in the Visaj install directory.

Quick Reference

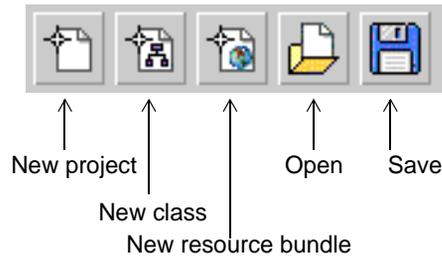
B

Note – In the SNIFF+ Visaj integration the Resource Bundle Editor, Image Editor and the Project Window are not supported and java code is automatically generated and stored in your SNIFF+ project directory so please ignore toolbar buttons and menu entries related to these tools and code generation.

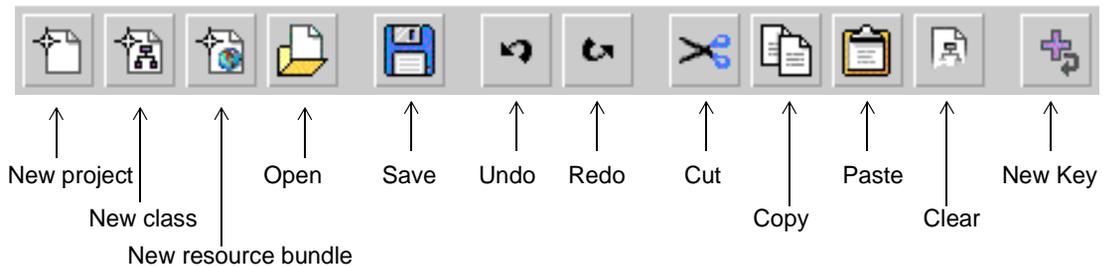
B.1 Class Editor Toolbar Buttons



B.2 Project Window Toolbar Buttons



B.3 Resource Bundle Editor Toolbar Buttons



B.4 Class Editor Menu Items

There are 10 menus in the Class Editor - “File”, “Edit”, “Class”, “Method”, “Object”, “View”, “Generate”, “Palette”, “Windows” and “Help”.

B.4.1 The File Menu

The first item in the File menu is “New”. This is a pullright menu, containing three further items.

New Project - Create a new, blank project window.

New Class - Display the Class Editor.

New Resource Bundle - Display the Resource Bundle Editor.

Open... - Open a saved document.

Save - Save the current class.

Save as - Save the current class using the specified filename.

Import - Pulls right to allow importing of class files generated from other tools. The current version supports:

X-Designer bridge file - A design created using X-Designer, the Motif builder.

Close - Close this Class Editor window.

Exit Visaj - Exits the application, closing all open windows.

B.4.2 The Edit Menu

Undo - Undo the last change.

Redo - Make the change again, after Undo.

Cut - Cut the selected item to the clipboard.

Copy - Copy the selected item to the clipboard.

Paste - Paste the contents of the clipboard.

Clear - Clears the currently selected item without saving it on the clipboard.

B.4.3 The Options Menu

Authentication... - Displays the Licensing dialog.

B.4.4 The Class Menu

Properties... - Displays a dialog for changing the class signature, the package and the list of imports.

B.4.5 The Method Menu

Add new method - Adds a new method to the class.

Add event binding - Displays the Event Binding Editor for the currently selected method.

Main method - Makes the currently selected method the “main” method.

Delete - Delete the currently selected method.

B.4.6 The Object Menu

Properties - Displays the Property Sheet for the currently selected item.

Customize... - Displays the Customizer for the currently selected item, if one has been provided by the component vendor.

Layout... - Displays the Layout Editor for the currently selected container.

Reset - Recreates the objects in the dynamic display.

B.4.7 The View Menu

Left justify tree - Display the hierarchy left justified.

Layout horizontally - Switch the tree view of the hierarchy so that it grows horizontally.

Collapse all composite components - Hide the children of Swing components which are composites.

Fold/Unfold nodes - Fold away or unfold the hierarchy below the currently selected node.

Collapse/Expand selected composites - Hide or show the children of the selected Swing composite component.

B.4.8 The Generate Menu

Generate java... - Displays the generate dialog for generating new Java code files or updating existing ones.

B.4.9 The Palette Menu

Show labels - A toggle to display or hide the component names along with their icons in the component palette.

Merge palette file... - Display a File Dialog prompting for the name of a palette to merge into the existing palette.

Load Jar file... - Display a File Dialog prompting for the name of a JAR file. Any beans in the file are added to the palette.

B.4.10 The Windows Menu

Color Selector - Display the Color Selector window.

Font Selector - Display the Font Selector window.

Image Editor - Display the Image Editor.

The Windows menu also lists all current Visaj windows, enabling you to switch easily between them.

B.4.11 The SNIFF+ Menu

Browse <class> - Displays the members of the class in the SNIFF+ Class Browser.

Show <class> in Hierarchy - Displays the inheritance relationships of the class in the SNIFF+ Hierarchy Browser.

Edit <generated java file> - Opens the SNIFF+ Source Editor and displays the generated Java file.

Find <selected method> - Displays all symbols matching the selected method name in the SNIFF+ Symbol Browser.

Show <selected symbol> - Opens the SNIFF+ Source Editor and is positioned at the selected symbol.

Retrieve <selected object> - Opens the SNIFF+ Retriever and retrieves the selected object from all files in the current project.

B.4.12 The Help Menu

Swing Compatibility - Display information on Swing support in Visaj.

Index - Display the online help main index for the Class Editor.

User Guide - Display the user guide in HTML format in a separate window.

About Visaj... - Displays the splash screen and licensing information.

B.5 Project Window Menu Items

There are 5 menus in the Visaj main window: “File”, “Edit”, “Project”, “Windows” and “Help”.

B.5.1 The File Menu

New - A pullright menu, containing the following three items:

New Project - Create a new, blank project window.

New Class - Display the Class Editor.

New Resource Bundle - Display the Resource Bundle Editor.

Open... - Open a saved document.

Save - Save the current project.

Save as - Save the current project using the specified filename.

Import - Pulls right to allow importing of class files generated from other tools. The current version supports:

X-Designer bridge file - A design created using X-Designer, the Motif builder.

Close - Close the current document.

Exit - Exit the application, closing all open windows.

B.5.2 The Edit Menu

Edit - Display the appropriate editor for the selected file.

B.5.3 The Options Menu

Authentication... - Displays the Licensing dialog.

B.5.4 The Project Menu

Add File... - Add a file to the project.

Add Group... - Create a new group.

Remove File - Remove the selected file from the project.

B.5.5 The Windows Menu

The Windows menu lists all current Visaj windows, enabling you to switch easily between them.

B.5.6 The Help Menu

Index - Display the online help main index for the Project window.

User Guide - Display the user guide in HTML format in a separate window.

About Visaj... - Displays the splash screen and licensing information.

B.6 Resource Bundle Editor Menu Items

There are 5 menus in the Resource bundle Editor - “File”, “Edit”, “Generate”, “Windows” and “Help”.

B.6.1 The File Menu

New - Displays a new Resource Bundle Editor window

Open... - open a saved document.

Save - Save the current project.

Save as - Save the current project using the specified filename.

Export language... - Export the currently selected language in a readable format.

Import language - A pullright menu containing the following two items:

Merge... - Import into currently selected language, merging with existing data.

Append... - Import into currently selected language, appending to existing data.

Close - Close the current document.

B.6.2 The Edit Menu

Undo - Undo the last change.

Redo - Make the change again, after Undo.

Cut - Cut the selected item to the clipboard.

Copy - Copy the selected item to the clipboard.

Paste - Paste the contents of the clipboard.

Clear - Clear the currently selected item without saving it on the clipboard.

Delete key - Delete the currently selected key (row).

Delete language - Delete the currently selected language (column).

Add key - Add a new key (row).

Add language... - Add a new language (column) - displays a dialog prompting for the ISO code of the new language.

B.6.3 The Generate Menu

Generate all files - Generate a Java source code file for each language (including "Default") in the table - displays the Class Name dialog if the class name has not been specified.

Set properties... - Display a dialog for changing the class name to be used for code generation and for adding a package name.

B.6.4 The Windows Menu

The Windows menu lists all current Visaj windows, enabling you to switch easily between them.

B.6.5 The Help Menu

Index - Display the online help main index for the Resource Bundle Editor.

User Guide - Display the user guide in HTML format in a separate window.

About Visaj... - Displays the splash screen and licensing information.

Bibliography

C

C.1 Introduction

This chapter supplies further details on the books which are referred to in this manual and others which we recommend for additional reading.

We list ISBN numbers but suggest you consult your book supplier for the latest editions.

C.2 Books on Java

Flanagan, David, *Java in a Nutshell*. O'Reilly & Associates, Inc., 1996. ISBN 1-56592-183-6

Arnold, Ken and Gosling, James, *The Java Programming Language*. Prentice Hall, 1996. ISBN 0-201-63455-4

Geary, David M, *Graphic Java 1.1*, Sun Microsystems Press, Prentice Hall, 1997. ISBN 0-13-863077-1

Harold, Elliotte Rusty, *JavaBeans*. IDG Books Worldwide, Inc., 1998. ISBN 0-7645-8052-3

Chan, Patrick, *The Java Developer's Almanac*. Addison Wesley, 1998. ISBN 0-201-37967-8

C.3 Books on Internationalization

O'Donnell, Sandra Martin, *Programming for the World: a guide to internationalization*. Prentice Hall, 1994. ISBN 0-13-722190-8

Lunde, Ken, *Understanding Japanese Information Processing*. O'Reilly & Associates Inc., 1993. ISBN 1-56592-043-0

C.4 Books on HTML

Graham, Ian S., *HTML Sourcebook*. John Wiley & Sons Inc., 1995. ISBN 0-471-11849-4

Musciano, Chuck and Kennedy, Bill, *HTML: The Definitive Guide*. O'Reilly & Associates Inc., 1996. ISBN 1-56592-175-5

Spainhour, Stephen and Cenercia, Valerie, *WebMaster in a Nutshell*. O'Reilly & Associates Inc., 1996. ISBN 1-56592-229-8

absolute positioning	In terms of the laying out of components, absolute positioning refers to the ability to specify the location of a component within its container without allowing the component to be resized or to move relative to anything else when the container is resized.
AWT	The set of portable Java components which were designed to look and feel like the native toolkit.
CLASSPATH	An environment variable which is used by the Java virtual machine to locate classes referred to in an application. CLASSPATH can contain directories or JAR files.
class structure	In Visaj, this is the tree on the right of the Class Editor window which shows the contents, order and signature of the Class under construction.
click over	To press and release the mouse button when the mouse pointer is above something.
code expression	In Visaj Property Sheets, these refer to any arbitrary piece of code which will be copied into the generated code for the selected property.
component	Strictly speaking, anything derived from <code>java.awt.Component</code> or <code>java.awt.MenuComponent</code> . Sometimes, the term is used more broadly to refer to objects derived from other Java classes.

constructor	The first method of a class which is called when a Class is created. The constructor is special in that the signature is unique (it has no return type) and it is always called for a new Class. If a subclass does not provide a constructor, the first such instance in the superclass hierarchy is called.
constraints	The properties of a component which control its size and positions. The constraints are dictated by the layout of the component's parent container.
container	Strictly speaking, anything derived from <code>java.awt.Container</code> .
containment hierarchy	In Visaj, the hierarchical view of the user interface which shows how objects relate to each other in terms of their parent->child relationship.
customizer	A dialog which sets properties on a Java bean. Not all beans have customizers; they have to be specially provided for a bean by the bean vendor.
design	In Visaj, the design is the user interface which is being created. Since any or all methods in a Class can have a user interface, the design may refer to individual methods or the whole class and includes the hierarchy of components plus any properties or variables.
double-click	Press and release the mouse button twice in quick succession.
drag and drop	The mechanism on window systems which allows information to be "carried" across the screen and moved or copied from one area to another by pressing a mouse button over the information, moving the mouse to another place and releasing the button. Different window systems have different ideas of which mouse button to use.
dynamic display	The window in Visaj which previews the user interface as it is being developed.

environment variable	UNIX and DOS shells run within an environment. Environment variables are variables which are set within a shell. Any applications run from that shell can use environment variables to set options and preferences. Java provides an interface for checking environment variables.
event bindings	In Visaj, the linking of objects using events. Event bindings provide dynamic functionality for your design.
events	In the Java model, events are actions, such as mouse clicks or list item selections. Objects which have registered themselves as listeners to other objects are notified of events occurring.
floating window	A window which is not attached to any other. Such a window may be moved and iconized independently of the window which caused it to be displayed. In Visaj, the dynamic display is an example of a floating window.
GUI builder	Graphical User Interface builder, as its name suggests, refers to a tool which allows you to build a user interface graphically.
IDE	An IDE (Integrated Development Environment) is an application which provides complete support for the building of an application. This would include a tool for defining the user interface, a tool for defining the files which make up the application, a compiler and a debugger. Some IDEs may include other tools too.
inset	The top, left, right and bottom margins of a container or component.
install directory	The root directory where your copy of Visaj was installed. You may need to know where this is in order to load example files for the tutorial. Any pathname for such files is relative to the directory where Visaj was installed.

internationalization	The setting of an application's strings (and other text) such that it may be altered easily to display in another language. Fetching strings from a pool, for example, means that only the strings in the pool need to be changed and not the application itself.
invisible bean	A Java bean which is not visible. All components (in the strict sense of the word) are capable of being displayed on the screen, but objects derived from other classes may not be.
JAR file	A Java archive file is a file which groups classes and resources so that they may be used by another application. A JAR file may be compressed. A JAR file also contains a <i>manifest file</i> which is a plain text file listing the contents of the archive.
Java Beans	Java Beans are reusable software components with a pre-defined API.
JFC	The JFC (Java Foundation Classes) are an extension to the AWT (Abstract Windowing Toolkit). They include a comprehensive set of graphical user interface class libraries, pluggable look and feel and the Accessibility API.
method design	All aspects of the design of a method, which includes the hierarchy of user interface objects, its signature and any event bindings. The left area of the Class Editor window contains all the editors for the design of a method.
method editors	The left area of the Class Editor window which contains tools for building the hierarchy of user interface objects, changing the method signature and setting up event bindings.
multiple selection	The selection of more than one object, specifically in the containment hierarchy of the Class Editor window.

mwt	Stands for Motif Widget Set. These widgets are provided with Visaj like the Diamonds. They are Java emulations of Motif widgets not available in the AWT set. They are described in Appendix A, “Diamond Components”, starting on page 191.
native methods	This is a Java term which refers to methods which are specific to one platform only and which are, therefore, not portable.
object	In the Java language, an object is the base class of all components. The term objects is used in Visaj to denote the basic building blocks of a user interface. See object palette.
object palette	The palette on the left of the Class Editor window containing the elements available for building a user interface.
package	A reserved word in the Java language, the package is the group to which a Class belongs. All classes belong to a package. The package name corresponds to the directory where the Class files reside. For example the package name <code>java.awt.event</code> corresponds to the directory <code>java/awt/event</code> ¹
portable code	Code which may be moved without difficulty from one platform to another and run.
properties	In Java, objects have properties which control all aspects of their appearance and behavior. Properties may be set and fetched by an application.
project	The “whole” application. A project includes the idea of all source files, image files and any extra data.

1. This is the UNIX directory format. Windows programmers should replace the forward slash (“/”) with the backward slash (“\”).

Pure Java	100% Pure Java is a trademark of Sun Microsystems. It is a standard which describes applications which are written completely in the Java language. To qualify for this description, applications must not use any native methods or import any classes which are not documented parts of the Java API. 100% Pure Java applications are guaranteed to be platform independent.
radio button behavior	The behavior of buttons when only one in a group can be selected. In Java, this applies to checkBoxes within a checkBoxGroup. The selected checkBox will stay set until another is selected.
resource bundle	A resource bundle contains the strings an application uses. Each string is identified by a <i>key</i> . Applications fetch strings from the resource bundle using the key as an indirection. With resource bundles for different languages, an application can far more easily be internationalized.
signature	In terms of a class or method in the Java language, the signature is the name, accessibility, scope and inheritance of a class or method. It also includes the parameters of a method. For example: <pre>public Class MyFrame extends Frame private boolean MyMethod(String s)</pre>
superclass	Also known as “base class”, the superclass is the class from which a given class is derived. In the Java language, it is the class which follows the word “extends” in the class definition.
Swing	The GUI components written in the Java language, without window-system-specific code, which form part of the JFC.
tabbed panel	A Diamond component which is like a stack of cards. Each card has a tab with the name of the card on it. Pressing the tab brings the associated card to the front. The method editors in the Class Editor window are arranged within a tabbed panel.

this	In the Java language, “this” is a reserved word and refers to the current class.
tree view	In Visaj there are two types of tree view: the containment hierarchy, which shows the relation between objects in terms of containments and the class structure, which provides a convenient way of grouping the elements of a Class.
variable name	In Visaj every object which makes up the user interface being created has a name. This name is generated into the code and provides a means of referring to the object. Use variable names which are meaningful to you if you are thinking of accessing objects.
variable scope	The <i>scope</i> of an object in Visaj defines how accessible it is - its visibility to other classes. An object (in Property Sheets they are also referred to as “variables”) can be “public” - meaning that it is visible everywhere, “protected” - meaning that it is visible within its own class, all subclasses and all classes in the same package, “private” - meaning that it is only visible within its class or the default (when no keyword is given) - meaning that it is visible within its own class and package only.

Index

A

- absolute positioning 71
- add noise filter 135
- adding code 153
- adding items to Choice components 68
- adding items to Lists 68
- adjust colors in Image Editor 134
- alignment button in layout editor 71
- Apple Macintosh
 - file types on 159
- applet 51
- ArrowButton Diamond 192
- assignment, in event binding editor 81
- Authentication 54
- AWT to Swing conversion 90

B

- Bean Creator
 - description 55
- beans, creating your own 76
- Beans, invisible 74, 214
- BevelBorder 101
- Bezel Panel Diamond 197
- blur filters 135

- Book Diamond 196
- books on HTML 210
- books on internationalization 209
- books on Java 209
- border filter 133
- Border layout editor 70
- borders, editing property 100
- Box Layout Editor 96
- BoxLayout 96
- bumps filter 136
- burn tool in Image Editor 120

C

- change palette 60
- changing event binding compilation directory 168
- changing the menu font 167
- changing the window font 167
- CheckboxGroups 76
- checkerboard filter 137
- Choice component, populating 68
- circle tool in Image Editor 120
- Class Editor
 - editing event bindings 79

- properties 60
- Class structure
 - how to create 32
- classes
 - default constructor 40
 - editing properties of 40
 - interface methods 41
 - methods added automatically 41
- CLASSPATH environment variable 165
- collapse components 94
- color properties 65
- color selector 53
- command line options 166
- component hierarchies
 - to create 58
- components
 - object initialization 63
 - using third party 162
 - variable name 62
 - variable scope 62
- components with no parent 58
- configuration 162
- constraints in layouts 73
- containment hierarchy 55
 - dragging in 80
- Controller Diamonds 194
- converting .gui files 168
- converting designs to javax 93
- creating a JAR for beans 77
- creating reusable components 76
- creating your own beans 76
- crop, in Image Editor 130
- cross in event parameter dialog 83
- customizer 68

D

- default constructor 40
- Default variable scope 62

- design time flag for beans 167
- detect edges filter 136
- Diamonds
 - ArrowButton 192
 - Bezel Panel 197
 - Book 196
 - Controllers 194
 - description 191
 - DlogTemplateLayout 195
 - DrawnButton 193
 - FlexiButton 193
 - FramedPanel 194
 - in CLASSPATH 197
 - in generated code 159
 - Knob 194
 - Meter 194
 - PolygonButton 193
 - Progress Bar 194
 - RoundButton 193
 - Separator 194
 - Slider 194
 - Splitter 195
 - StatusBar 195
 - SuperGrid 195
 - ToolBar 197
 - using in generated code 197
- dilate filter 137
- dither colors in Image Editor 134
- DlogTemplateLayout 195
- dodge tool in Image Editor 120
- Down button in event binding list 87
- dragging between components 80
- DrawnButton Diamond 193
- drop shadow filter 135
- DTCColors 169
- dummy frames 58
- dynamic design window 57
- dynamic display
 - components not highlighting 95
 - recreating 58

E

Edit menu

- in Class Editor 201
- in project window 205
- in Resource Bundle Editor 206

emboss filter 135

EmptyBorder 101

environment variables

- CLASSPATH 165
- VISAJOPTS 164

erode filter 137

EtchedBorder 102

Event Binding Editor 79

- cross in dialog 83
- tick in dialog 83

event bindings

- '=' in editor 81
- creating 80
- directory compiled in 168
- dragging between components 80
- editing 80
- Exceptions occurring 179
- invalid 87
- reordering 87

Event Bindings List 86

exception handlers 41

exceptions, where generated 178

F

File menu

- in Class Editor 200
- in project window 204
- in Resource Bundle Editor 205

file selection dialog, not visible 177

file type on Apple Macintosh 159

fill tool in Image Editor 119

filters in Image Editor 132–137

- add noise 135
- adjust colors 134

blur 135

border 133

bumps 136

checkerboard 137

detect edges 136

dilate 137

dither colors 134

drop shadow 135

emboss 135

erode 137

gamma 133

gray out 134

marble 135

maximum/minimum/median 136

mesh warp 133

mosaic 134

offset 133

oil 135

outline 137

plasma 137

ripple 133

shapeburst 135

sharpen 136

solarize 134

sparkle 135

sphere 133

texture 136

threshold 134

transparency 133

twirl 133

water ripples 133

weave 136

FlexiButton Diamond 193

flip, in Image Editor 131

Flow layout editor 70

font properties 65

font selector 52

Frame, pack() 153

FramedPanel Diamond 194

G

- gamma filter 133
- generated code
 - example 155
 - previous file disappears 186
 - starting directory for dialog 168
 - using Diamonds 197
- gradient editor in Image Editor 126
- gradient list in Image Editor 126
- gradient tool in Image Editor 120
- gray out filter 134
- GridBag layout editor 72

H

- help, online 5
- hierarchies of objects 55
- HTML
 - reading list 210

I

- IDE, integration with 161
- Image Editor 107–136
 - transparency 122
- image properties 66
- ImageIcon failing to load 186
- importing from other tools 50
- ink dropper tool in Image Editor 119
- instance variable 62
- integration with an IDE 161
- interface methods 41
- internationalization
 - reading list 209
 - using resource bundles 139
- invisible Beans
 - demonstration 29
 - description 74, 214
- ISO Country Codes 142
- ISO Language Codes 142

J

- JApplet 51, 105
- JAR files
 - loading 60
- Java
 - invisible Beans 74, 214
 - reading list 209
- Java Beans
 - customizers 68
 - demonstration of invisible 29
- Java Console 54
- Java Foundation Classes 89
- Java Workshop 161
- java.compiler=none 169
- javax, converting designs to 93
- JFC 89
- JFrame
 - pack() 153
- JIT compiler, switching off 169
- JLayeredPane 106
- JList 98
- JPEG 117
- JScrollPane components 164
- JTabbedPane 106
- JTable 99

K

- KL Group components 164
- Knob Diamond 194

L

- Layout Editor
 - alignment buttons 71
 - how to use 20
- layout editors
 - description 69
- layout properties 68
- layouts

as invisible beans 74
Border 70
DlogTemplateLayout 195
Flow 70
GridBag 72
SuperGrid 70, 195
line tool in Image Editor 119
LineBorder 102
List component, populating 68
loading JAR files 60
local variables 62

M

Macintosh
 file types on 159
magic wand tool in Image Editor 119
manifest file for beans 77
marble filter 135
MatteBorder 102
maximum/minimum/median filters 136
mesh warp filter 133
Meter Diamond 194
methods
 added automatically 41
 adding to class 41
 editing signature of 48
 exception handlers 41
 interface 41
 multiple hierarchies in 58
Microsoft Windows 95 165
Microsoft Windows NT 165
mosaic filter 134
Motif, importing legacy designs 50
multiple hierarchies 58
multiple selection 67
mwt 196, 197
mwt, definition 215

N

new document and window 200, 201, 204
null layout 71
NullPointerException from generated code 186

O

object initialization 63
ObjectInputStream error 187
offset filter 133
oil filter 135
online help 5
online user guide 7
options to pass to Visaj 166
order of event bindings 87
outline filter 137

P

pack method call for frames 153
palette file 162
 system property 167
palette, changing 60
panning tool in Image Editor 118
parameters in event bindings 82
pencil tool in Image Editor 119
plasma filter 137
plug-in file location 168
pointer tool in Image Editor 118
PolygonButton Diamonds 193
populating Choice components 68
populating List components 68
prerequisites 7
private variable scope 62
problem after copying files 187
Progress Bar Diamond 194
project 147, 215
properties 60
 border 100

- image runtime resource path 67
- images 66
- property editing 61
- property inheritance 61
- Property Sheet
 - color and font properties 65
 - how to use 61
 - layout properties 68
 - multiple selection 67
- protected variable scope 62
- public variable scope 62

R

- radio button behavior 76
- recreating the dynamic display 58
- rectangle selection tool in Image Editor 118
- rectangle tool in Image Editor 119
- reset 58
- resource bundles 139
- re-usable component hierarchies 58
- reusable components, creating 76
- ripple filter 133
- rotate, in Image Editor 131
- RoundButton Diamond 193
- Runtime resource path for images 67

S

- saving images 117
- scope of beans 169
- scope of components 62
- Separator Diamond 194
- shapeburst filter 135
- sharpen filter 136
- signature
 - class 40
 - method 48
- Slider Diamonds 194
- SNiFF+

- adding a new Visaj project 10
- adding an existing Visaj project 11
- integration with Visaj 9
- menu 203
- SoftBevelBorder 101
- solarize filter 134
- sparkle filter 135
- sphere filter 133
- Splitter Diamond 195
- StatusBar Diamond 195
- strings, internationalizing 139
- subclassing the generated class 153
- SuperGrid 195
- SuperGrid layout editor 70
- Swing
 - component tips 105
 - converting designs to 90
 - hiding composites 94
 - how to load 90
- Swing conversion of packages 168
- switching off JIT compiler 169
- system colors usage 169
- system properties 166

T

- TeaSet components 164
- text tool in Image Editor 120
- texture filter 136
- The 203
- third party components 162
- this (the class) 178
- threshold filter 134
- tick in event parameter dialog 83
- TitledBorder 102
- ToolBar Diamond 197
- tools in Image Editor 118
- transparency 122
- transparency filter 133

tutorial 13
twirl filter 133

U

Unknown class, error 181
Up button in event bindings list 87
using third party components 162

V

variable name 62
variable scope 62
Visaj freezes on import 180
Visaj not responding 178
VISAJOPTS environment variable 164
vj.AWTPalette.file 167
vj.beanDesignTime 167
vj.CommonPalette.file 167
vj.convertPackages 168
vj.cwd 168
vj.JWS 168
vj.menuFont 167
vj.pluginFile 168
vj.tmpDir 168
vj.variablesDefaultToInstanceVars 169
vj.windowFont 167

W

water ripples filter 133
weave filter 136
Workshop Visual, importing save files 50
writing manifest file 77

X

X-Designer save files, loading 180
X-Designer, importing save files 50

Z

zoom tool in Image Editor 118