SYMANTEC.™

Visual Cafe™ User's Guide

Symantec Visual Cafe™ User's Guide

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Language Addendum

Acknowledgements

Documentation team: Cheryl Chambers, Orlando Cordero, Jim Douglas, Erfert Fenton, Guy Haas, Kimberly Haeger, Lauren Katzive, Landon Ott, Cheryl Potter, and Jonathan Simonoff.

Special thanks to: David Allcott, Eric Anderson, Andrei Arefiev, John Bealkowski, Erik Bruchez, Tim Bumgarner, David Bustin, Carlos Chang, Craig Conner, John Dixon, Dennis Dunham, David Ewing, Bob Foster, Mac Foster, Steve Frehe, Tatiana Golovina, Tim Hanna, Naoki Hamajimi, Ling Hao, Michael Hopkins, Warren Horowitz, Dawn Hutton, Lubomir Litchev, Tim Lopez, Rod Magnuson, Andrew McFarland, Terry Okamoto, Dan Small, D'Arcy Smith, Glenn Taylor, Craig Yara, and the rest of the Visual Cafe 3.0 team.

Extra special thanks go to: Erfert the Wonder Editor and Horse Wrangler.

Heartfelt thanks goes to C. Rand McKinney, Mom, and Dad.

C O N T E N T S

Section I The Essentials

Chapter 4 Working with Source Code

Chapter 6 Debugging Your Program

Section II Using Components

Chapter 7 Working with Components

I

The Essentials

CHAPTER 1

Welcome to Visual Cafe

Symantec's Visual Cafe family of products is the first visual **Rapid Application Development** (**RAD**) tool designed exclusively for the Java programming language. Visual Cafe is a complete form-based development environment that provides a rich set of What-You-See-Is-What-You-Get (WYSIWYG) tools and components that enable you to develop, debug, and deploy high-performance Web applets and stand-alone Java applications. Additional tools such as JavaBeans and Java component libraries, graphics libraries, and templates provide the complete solution for the Java developer or sophisticated Web developer.

Visual Cafe is available in three editions:

- ◆ Visual Cafe Standard Edition (SE) provides a Java development environment that's suitable for the Web developer with technically sophisticated needs. This edition is also a good development environment for programmers who are new to Java.
- ◆ Visual Cafe Professional Edition (PE) is for developers who need the latest and most powerful Java features in their development environment. Visual Cafe Professional Edition includes all the features available in the Visual Cafe Standard Edition, as well as many additional features that enhance its capabilities.
- ◆ Visual Cafe Database Edition (DE) is for database application developers who want full database connectivity. Visual Cafe Database Edition includes all the features available in the Visual Cafe Professional Edition, as well as additional features that relate to database connectivity.

Visual Cafe features

This manual describes the features that are common to each edition of the Visual Cafe family of products: Visual Cafe Standard Edition, Visual Cafe Professional Edition, and Visual Cafe Database Edition. Each edition includes a core set of tools that enable you to create Java applets, servlets, and applications. The Professional Edition includes all the features found in the Standard Edition, plus a set of additional features that includes enhanced debugging capabilities, version control integration, a wizard that helps you create servlets, native compiling, and a tool that simplifies the process of program localization. If you're using the Visual Cafe Standard Edition, consult the following table to see which features that edition supports. All the features that are in the Visual Cafe Professional Edition are also in the Visual Cafe Database Edition, which also includes a specialized set of features for adding database connectivity to your programs. An additional manual, the *Visual Cafe Database Developer's Guide*, is included with the Database Edition; that manual covers just the features that are specific to the Database Edition.

Included with the Professional and Database Editions is a Web-page designer and publisher called **Visual Page.** Visual Page provides a WYSIWYG environment that includes a visual designer, a source code editor, and a publishing utility. Visual Page comes with its own documentation and online help. For further information, see the *Visual Page User's Guide.*

The features of Visual Cafe Standard Edition (SE) and Visual Cafe Professional and Database Editions (PE/DE) are shown in the following table:

Feature	SE	PE/DE
JFC (Swing) component support	X	X
JFC (Swing) MenuDesigner support	X	X
Improved Interaction Wizard and interaction editing features	X	X
JDK 1.1.7a and 1.2 support	X	X
Javadoc Editor and Viewer	X	X
Improved deployment features	X	X
Improved AutoJAR tool and JAR Viewer	X	X

Feature	SE	PE/DE
Rapid JavaBeans development	X	X
Code Helper	X	X
Syntax Checker	X	X
Improved GridBagLayout support	X	X
Improved version control support		X
Localization tool		X
Servlet project template wizard		X
Native x86 compiler		X
Improved customizable user interface (MDI)		X
Incremental debugging		X
Remote debugging		X
Web browser debugging		X
Visual Page integration		X

What's new in Visual Cafe version 3.0

In this section you'll find a brief description of each of the features that are new or improved in Visual Cafe version 3.0. The features that pertain to all editions of Visual Cafe — Standard, Professional, and Database — are described first. Next, the new or improved features that apply only to the Professional and Database Editions are described.

New or improved features in all editions

Here you'll find a short description of the new or improved features that are found in all three editions of Visual Cafe version 3.0.

JDK 1.1.7a and 1.2 support

Visual Cafe supports the latest in Java technology. You can create, compile, and deploy Java code that complies with the JDK (Java Development Kit) 1.1.7a.

You can also choose to compile and debug your code using the latest beta version of the JDK 1.2. JDK 1.1.7a is an integral part of the Visual Cafe environment, while JDK 1.2 is not. If you use JDK 1.2, you won't be able to see your 1.2 code represented in the Form Designer or menu designers. To use JDK 1.2, you need to download it from the Sun Microsystems Web site (`http://java.sun.com`)and install it for Visual Cafe by creating an internal Java Virtual Machine (Java VM) for it. For more information about using different VMs, see "Using different Java virtual machines in Visual Cafe" on page 5-22.

Improved Just-in-Time compiler (JIT)

Symantec's Just-in-Time compiler, or JIT, is the fastest around. It's so fast that it has been included in Sun Microsystems' Java Development Kit since JDK 1.1.6. A fast JIT will keep your Java programs running quickly on end users' machines. For more information about the JIT compiler, see "Symantec's Just-in-Time compiler" on page 2-15. For details on compiling your programs, see Chapter 5, "Compiling and Deploying Your Project."

Swing support

Visual Cafe version 3.0 includes full support for Swing components. *Swing* is the new set of Java visual components that you can use to create user interface elements such as buttons, tables, lists, text fields, windows, dialog boxes, and so on, as well as applets and applications. For more information on Swing components, see Chapter 8, "Working with JFC/ Swing Components."

The Swing Menu Designer

Visual Cafe now includes a Swing Menu Designer that makes it easy to create or change menus. The Swing Menu Designer lets you type in menu names, insert separators, insert menu objects, attach menu items to events, and otherwise construct menus. For more information, see "Working with Swing menus" on page 8-22.

Improved interaction features

You can create and manage interactions more easily than ever with Visual Cafe’s improved Interaction Wizard. You can select actions, methods, properties, arguments, and strings associated with interactions in a wizard that takes you step by step through the process of creating interactions.

Interactions display in the Form Designer as arrows between or within objects, and you can choose to see some or all interactions. Deleting interactions is as easy as clicking an interaction arrow in the Form Designer and pressing the DELETE key.

For more information, see Chapter 9, “Working with Events and Interactions.”

Javadoc support

You can now easily create and edit Javadoc comments and quickly generate the associated HTML files. Use the Javadoc Editor to create and edit your Javadoc comments, and use the Javadoc Viewer to browse Javadoc comments. For more information, see “Using the Javadoc Editor” on page 4-61 and “Using the Javadoc Viewer” on page 4-66.

Rapid JavaBeans development

If you’re creating a Bean, but you haven’t added it to the Component Library, now you can also quickly update it so that it can be used in other projects. For more information, see “Packaging your Bean for distribution” on page 10-21.

As you work on a JavaBeans component, you can automatically update the Bean in the Component Library and all instances of that Bean in one step. For more information, see “Automatically updating Beans in the Component Library” on page 10-19.

Improved AutoJAR tool and JAR Viewer

Use AutoJAR to quickly put your Bean into the Component Library. Any open projects that use that Bean will then use the updated version. Now when you’re creating JAR files, your settings will persist, so you won’t have to reset your JAR options each time you update your files. Use the JAR Viewer to easily view JAR files much like you do ZIP files in the Windows environment. You can also use the JAR Viewer to view a JAR file’s manifest file. For more information about using JAR files in Visual Cafe, see “Using JAR files” on page 5-55.

One-step deployment

Not only can you deploy to JAR files, but now you can deploy to ZIP files, CAB files, and directories. You can also deploy directly to an FTP server. If you want to deploy to a different operating system or deploy outside of the Visual Cafe environment, you can use the new command-line deployment utility. The JAR command on the Project menu has now been replaced with the Deploy command (also on the Project menu). For more information about Visual Cafe’s deployment features, see “Deploying your project” on page 5-31.

Code Helper

As you type in the Source window, you can use the Code Helper to suggest Java language keywords, depending on the context. You can use the Code Helper consistently, or you can use it just when you need it. For more information, see “Using the Code Helper” on page 4-49.

Syntax Checker

When you have the Syntax Checker enabled, errors in your source code will be highlighted. You can quickly find typing and syntax errors, which will help you avoid problems later. For more information, see “Using the Syntax Checker” on page 4-51.

Improved GridBagLayout support

When you’re using a component layout of `GridBagLayout`, you can use the new GridBagConstraints Editor to easily manage the contraints of this complex layout manager. For more information, see “Arranging components in GridBagLayout” on page 7-45.

New Java macro system

Visual Cafe provides an extensive macro capability, which you can use to automate common, repetitive, or lengthy editing tasks. Visual Cafe macros are now programmed in Java. Visual Cafe macros can perform most Visual Cafe commands, so you can use them to automate many Visual Cafe operations. For more information on using the Visual Cafe macro system, see the *Macro Reference* in the Online Help.

New features in the Professional and Database Editions

In this section you'll find a short description of the features that are new to Visual Cafe Professional Edition and Visual Cafe Database Edition. For detailed information about features that are new to just the Visual Cafe Database Edition, see the *Visual Cafe Database Developer's Guide*.

Localization tool

If you're developing programs for non-US English systems, you can use the Localization tool to quickly and easily localize your programs. Now you can create resource bundles for your files so that they can be localized for another language or region. You can also add and edit locales with Visual Cafe. For more information, see Chapter 13, "Localizing Your Java Programs."

Version control integration

You can now access your version control software from within the Visual Cafe environment. Check in, check out files without having to exit your work in Visual Cafe. For more information, see Chapter 12, "Using Version Control with Visual Cafe."

Servlet support

You can easily create a servlet by using Visual Cafe's new Servlet project template wizard, which is found with the project templates. Visual Cafe also provides the tools you need to debug your servlets. For more information, see "Creating a servlet" on page 3-30.

Improved customizable user interface

You can now customize your workspace even further by using dockable windows. You can position windows so that they "dock" against other windows or the edges of the environment, or you can have them float undocked so that you're free to move them. For more information, see "Setting environment variables in the sc.ini file" on page 3-72.

New features in the Database edition

If you want to include database connectivity in your programs, you'll be interested in the new features of the Visual Cafe Database Edition. You'll want to see the *Visual Cafe Database Developer's Guide* for details about these features, but here's a short description of each of them:

- Full design-time support for most JDBC drivers including JDBC/ODBC Bridge, Oracle Lite driver, and dbANYWHERE middleware.
- Seamless databinding to standard JFC/Swing components.
- Extensive stored-procedure support, including Beans, wizards, customizers, and test tools. This reduces programming time and enhances application functionality.
- Robust support for database functionality:
  - Query-By-Example (QBE)
  - Dynamic list binding (show one value, store another)
  - Data-validation adapters that ensure client-side integrity. You can use the validation functions that are provided or develop your own and spawn validation calls to databases, application servers, or middleware.
  - Calculation adapters and functions generate your own client-side values using functions such as addition and multiplication. You can use the calculation functions that are provided or import your own.
  - AWT databound Mask and Currency fields
- The event system provides an audit trail of events, SQL, and transactions, which simplifies tracing user and database activity.
- Customizable application components:
  - A status bar (`DBStatusBar`) provides interactive information about record status, form status, and transactions.
  - A database toolbar (`DBToolBar`) provides an efficient way to access functions such as record navigation, queries, updates, and the like.
- Live data during development allows you to make changes in real-time, without the need to recompile.
- Customizer and data binding for the standard JFC/Swing `JTable` component is provided:

  - Multi-column sorting
  - Easy customization of `JTable`'s visual properties, including fonts, background colors, column widths, and so on.
  - The ability to embed complex column editors within your `JTable`, including mask fields, currency fields, combo-boxes, checkboxes, or your own JFC/Swing components.
  - `JTable` databinding through JFC Model architecture.
  - Single-click addition of row number and row state columns to your `JTtable`.

- Database-related wizards and components have been enhanced to be more robust. Now there are more wizards, more property editors, more Bean customizers, and extensive smart properties to greatly increase productivity.

Version compatibility

In general, any applet or application that ran in JDK 1.0.2 should run correctly in Visual Cafe version 3.0. Incompatibilities may exist where functionality has changed between component versions.

Applets that depend on any new JDK 1.1 APIs will not work in browsers that support only 1.0.2, such as Internet Explorer 3.0, Netscape Navigator 3.0, and the alpha and pre-beta1 versions of the HotJava browser. In general, however, applets that rely only on APIs defined in 1.0.2 (but compiled with the JDK 1.1 compiler) will run on 1.0.2 browsers.

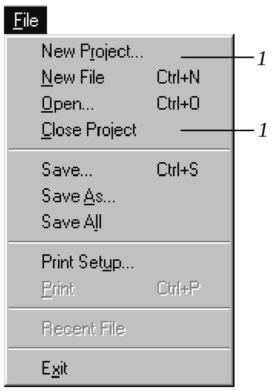
Visual Cafe automatically converts your Visual Cafe 2.x projects to Visual Cafe version 3.0 projects. During this conversion process, all source code generated by Visual Cafe 2.0 is converted from the JDK 1.0.x event model to the JDK 1.1.x event model. For more information about converting code from JDK 1.0 to 1.1, see "Migrating Java source files from JDK 1.0 to JDK 1.1" on page 3-38.

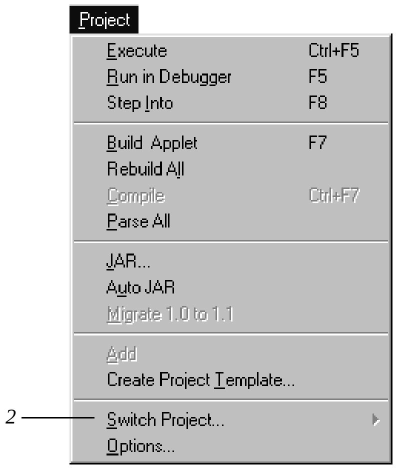
Menu rearrangement

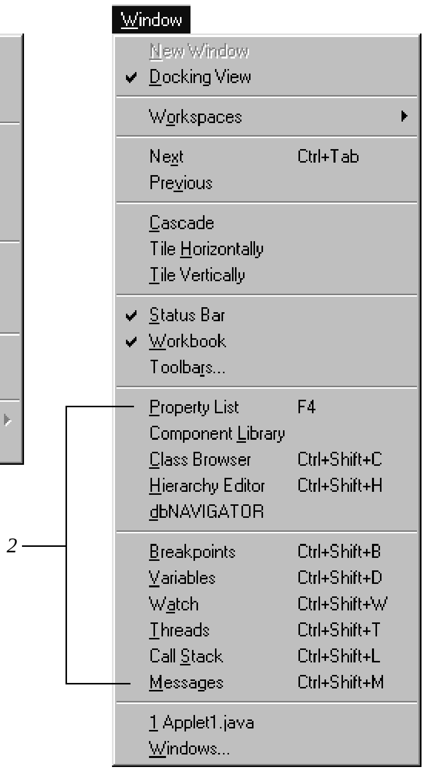
If the most recent version of Visual Cafe you're accustomed to using is before version 2.5, you'll notice that some of the menu items have changed in version 3.0. This may affect your work, especially if you used macros in pre-2.5 versions of Visual Cafe.

Visual Cafe versions prior to 2.5 did not have a View menu, so some items from the pre-2.5 Window menu were moved to the View menu in later versions. In addition, some items in the File menu were rearranged or added, and one item in the Project menu was moved to the new View menu.

Here's an example of how these menus in Visual Cafe have changed (only the File, Project, and Window menus are affected):



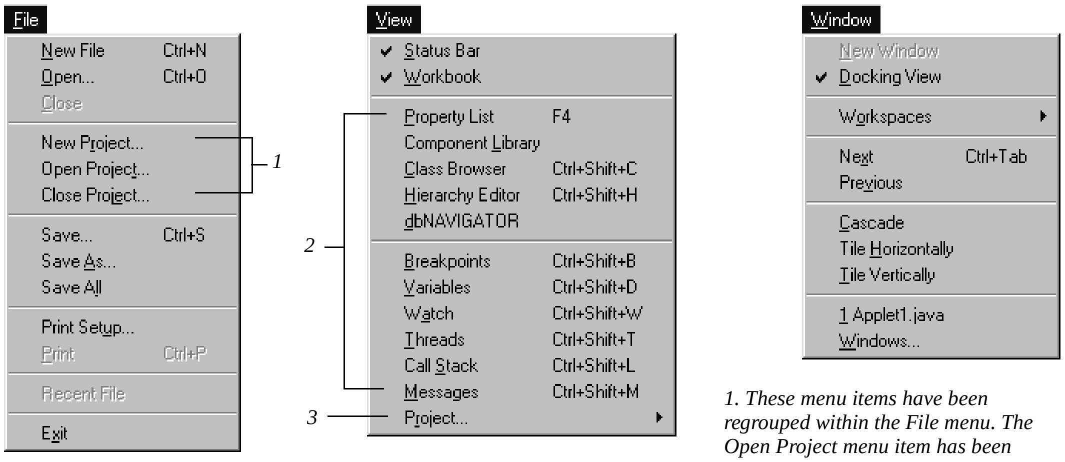




1. These menu items were rearranged within the File menu in versions 2.5 and later.

2. These menu items were moved to the View menu in versions 2.5 and later.

Here's an example of what the menus look like in versions after 2.5:



1. These menu items have been regrouped within the File menu. The Open Project menu item has been added.

2. These items have been moved from the Window menu to the View menu.

3. This menu item has been moved from the Project menu to the View menu.

Updating Visual Cafe

From time to time, Symantec provides updates to Visual Cafe. You can download updated versions from Symantec's Web site by using a utility called LiveUpdate. See Appendix A, “Updating Visual Cafe with LiveUpdate,” for information on using LiveUpdate.

Visual Cafe documentation

Visual Cafe comes with extensive documentation and online help to assist you in the process of developing applets and applications. These documents are described in this section.

Visual Cafe Getting Started

The *Visual Cafe Getting Started Guide* consists of a tour of the main features of Visual Cafe.

You may want to work through the tour, which takes less than an hour, before beginning work in Visual Cafe. The tour is designed to familiarize you with the main features of Visual Cafe by guiding you through the process of building a working Java applet. It requires no previous experience with Java, and is a quick way to learn Visual Cafe.

Visual Cafe Sourcebook

The *Visual Cafe Sourcebook* presents coding samples, in the form of applications and servlets, that were developed in Visual Cafe, and describes them in detail so you can understand how they work and how they were built. You can modify the code in this book to build your own applications, applets, and servlets.

Online Help

Visual Cafe has extensive online help that describes all of the procedures for building Java applets and applications. To access Visual Cafe's Help, choose Help Topics from the Help menu. The online help is also context-sensitive, which means that you can press F1 in most areas of Visual Cafe and receive information that pertains to your current activity.

You can also access information about Visual Cafe's components and review the Java API documentation from Sun Microsystems. You can access information about an individual component by typing the name of the component in the Index tab of the Help window, choosing Components Reference from the Help window's Contents tab, or selecting a component in the Component Library and pressing F1.

Visual Cafe also includes online help for the Java macro system. To access this help topic, choose Macro Reference from the Help menu.

ReadMe file

The ReadMe file is the first document you should read before using Visual Cafe. It contains late-breaking news, work-arounds, and known issues. This file is available for viewing at the end of the installation process, as well as from the Windows Start menu.

User's Guide

This manual, the *Visual Cafe User's Guide*, may be the document you turn to most frequently as you work with Visual Cafe. It contains both conceptual information and step-by-step procedures. This manual is divided into three parts: Part One, The Essentials; Part Two, Using Components; and Part Three, Professional Features.

Part One introduces you to Visual Cafe and takes you through the process of creating an application or applet. This section describes Visual Cafe's development environment and covers the basics of project development, with step-by-step instructions for creating, compiling, running, debugging, and deploying a project.

Part Two provides information on using various types of components, including AWT-based components, Swing components, and JavaBeans. In this section you'll also learn how to create interactions between components.

Part Three provides information on features that are not in Visual Cafe Standard Edition. These include native Win32 support, version control, and localization features.

Part Four consists of appendix information, such as using LiveUpdate to get the latest version of Visual Cafe, and troubleshooting information.

Portable Document Format

Portable Document Format (PDF) versions of the books are included with your copy of Visual Cafe. These documents require that Adobe Acrobat Reader be installed. Adobe Acrobat Reader is included on the installation CD-ROM of your Visual Cafe product. It's also freely available from Adobe Systems at `http://www.adobe.com`.

How much programming do I need to know?

In Visual Cafe it's possible to develop Java programs without writing a single line of source code. There are many ways to obtain Java programs, such as books, the Internet, and your friends and colleagues. You can drop these programs into Visual Cafe and have an applet, application, or Bean that's ready to use. However, sometimes these programs need modifications, and that's where real challenges to your programming expertise occur.

This manual assumes that you're familiar with the Java programming language or are learning how to program in Java. It's beyond the scope of this manual to teach Java programming, although you can learn more about Java by using Visual Cafe. To learn how to program in Java, you can consult one of the many excellent books that extensively explore the Java language. You can also search the Internet and find many well-written Java tutorials and summaries, as well as abundant resources to guide you on your way to becoming a Java developer. You might want to participate in a Java programmers' special interest group (SIG) in your area. You can find user groups on the Web or through Usenet newsgroups.

To use Visual Cafe, it helps to have a basic understanding of object-oriented programming languages, such as C++. Many of the principles and concepts of Java are based on those found in C++, although you should note that there are also some vast differences between Java and C++.

You should also have a basic understanding of cross-platform operating system concepts. This knowledge is useful, for example, when you're developing multithreaded Java programs, because all platforms handle threading differently.

If you're using Visual Cafe Professional Edition, you need to have basic Microsoft Windows programming skills in order to develop native 32-bit applications and libraries. Chapter 11, "Creating Native Win32 Java Applications," shows you how to create native Win32 applications.

If you're using the Database Edition, a basic understanding of Structured Query Language (SQL) and client-server models is necessary if you want to build complex databound Java programs.

Finally, a basic understanding of Hypertext Markup Language (HTML) is helpful so you can develop relationships between your Java applets and Web pages.

Conventions used in this manual

This manual uses the following typographic conventions:

- Names of files, resources, classes, methods, and variables, as well as code fragments and information you type, appear in the `code typeface`. Metanames appear in *italic*. A metaname is a descriptive placeholder for a real name. For example, when referring to a project's `.vep` file, we say *projectname*`.vep`, rather than specifying a specific project name.
- Terms that appear in the glossary appear in **bold** type when they're defined in the text.
- Names of menus, menu items, buttons, and other user interface elements appear in this typeface.
- Keys you press at the same time are shown as follows: CTRL-G (press the CTRL and G keys simultaneously). Please note that even though the letter keys are listed in uppercase, you should not hold down the SHIFT key when executing these key combinations unless the SHIFT key is listed as part of the combination.
- We use the word "program" to refer to whatever you're creating with Visual Cafe, whether it's an applet, application, library, or servlet.
- Wherever possible, we use the term "folder" rather than "directory" in accordance with standard Windows style, except in cases where the Visual Cafe interface uses "directory." Since Windows also uses the DOS system (which primarily uses the term "directory"), and Visual Cafe makes use of this DOS–Windows relationship, some areas of the product deal with "directories."

What's next?

Before you begin developing your Java applet or application, take a look at Chapter 2, "Developing in Visual Cafe," which provides information on the basic features of Visual Cafe and includes an overview of creating an applet and an application. Remember, if you're new to Java development, the tour in the *Visual Cafe Getting Started Guide* is a great place to start.

CHAPTER 2

Developing in Visual Cafe

Visual Cafe simplifies Java programming by providing an **Integrated Development and Debugging Environment** (**IDDE**). In this environment it's possible to design, develop, and build an applet or application without having to write a single line of source code. Visual Cafe helps you reference the rules of the Java language, automatically create and update source code, find and fix development problems, and optimize the entire process of creating Java applets and applications. In other words, Visual Cafe's tools provide everything you need to develop a Java program.

When a development environment is *integrated,* this means that the tools in the environment work together. For example, the compiler might find and display an error in the source code. With a double-click on the highlighted error, Visual Cafe opens the source file in the Source window and jumps to the line in the source code where the error occurred. Once at the error, Visual Cafe then displays another window with reference information about how the code should be formatted.

A large part of applet and application development involves adding and arranging components in forms. Visual Cafe provides tools to make designing forms a simple process.

This chapter introduces the basic features of Visual Cafe and provides a brief overview of the steps involved in creating a Java program. These features and procedures are described in detail in subsequent chapters.

The Visual Cafe environment

The Visual Cafe development environment provides windows, toolbars, editors, and wizards that allow you to create your Java applet or application in an easy-to-use, visually-oriented work area. These features are introduced in this section.

Windows

Visual Cafe **windows** are the areas on your screen that you use to develop your programs. Some windows allow you to enter and edit information, while others help you monitor the status of your project. These windows include:

- Form Designer – You can drag and drop components into this primary development area and arrange them for easy access.
- Project window – You can work with the different parts of your project in this tabbed window, which offers three views: Objects, Packages, and Files.
- Class Browser – A three-pane window that lists all the classes, methods, and data items contained in your program.
- Component Library – The repository for all components, including your custom components.
- Property List – Lists component properties and lets you control them. You can resize a component, name it, change its visibility, assign values, and change its colors and fonts.
- Source window – Use this window to add and customize source code for your project at any phase of the development cycle.
- Breakpoints window – Lets you set breakpoints while you're debugging to validate parameters and states.
- Call Stack window – Displays a list of active methods and variables while you're debugging.
- Variables window – Use this window to monitor and change variables in your expressions as you debug your code.
- Watch window – Here you can watch the debugger evaluate selected expressions.

- Threads window – Allows you to monitor and debug threads in your program.
- Messages window – Collects and displays information from Visual Cafe as you're running or debugging your project.
- Javadoc Viewer – Where you can browse your Javadoc comments in generated HTML files.
- JAR Viewer – Lets you see the contents of JAR (Java Archive) and ZIP files. You can quickly locate documentation for files, packages, JavaBeans in the Component Library, classes and methods in the Source Editor, and more.

Toolbars

Visual Cafe provides an extensive set of toolbars. Toolbars can be docked in the Visual Cafe window or floated on the screen; docked toolbars offer easy access to your favorite components.If you're more comfortable using menu commands, you can hide your toolbars for a clutter-free design environment.

The following toolbars are available in the Visual Cafe Project window:

- Standard – Contains buttons for working with files, printing, and copying and pasting.
- Component Palette – Contains buttons for adding components to a form.
- Layout – Contains buttons for arranging the placement of components on a form.
- Views – Contains buttons for debugging views.
- Debug – Contains buttons for debugging actions.
- Workspace – Lets you select the debugging or editing workspace.

When you switch from one environment to another in Visual Cafe, different functions become available. For example, when you work in the Class Browser or Hierarchy Editor, the Classes menu is enabled to assist you in working with classes. When you work in the debugging environment, various debugging windows, menus, and tools appear.

Editors

Visual Cafe has three editors for creating and managing Java projects. You can use these editors to control various aspects of project development, such as editing source code and manipulating classes.

- Source window – A tool for editing source code. For more information about working with Java source code, see Chapter 4, “Working with Source Code.”
- Class Browser – A three-pane window that lists all of the classes, methods and data items contained in your program. For more information about using the Class Browser, see “About classes, members, and the Class Browser” on page 4-1.
- Hierarchy Editor – Provides a visual representation of the classes in your project and their inheritance relationships. For more information about using the Hierarchy Editor, see “About the Hierarchy Editor” on page 4-35.
- Javadoc Editor – While working in the Source window, you can use the Javadoc Editor to quickly add Javadoc tags that document your code. This documentation will appear in HTML files when you generate Javadoc for the file.

Wizards

To streamline project development, Visual Cafe includes a number of **wizards,** interactive help utilities that guide you through a complex task. In a wizard, you enter or select information in a sequence of screens, or pages, then click a Finish button to complete the task.

Some key wizards in Visual Cafe are:

- Bean Wizard – Inserts JavaBeans components or lets you create your own to insert into your projects. For more information, see Chapter 10, “Working with JavaBeans Components.”
- Insert Class Wizard – Helps you insert the right class in the right place in your project. See “Using the Insert Class Wizard” on page 4-15 for details.
- Interaction Wizard – Helps you build relationships between components, or between a component and itself. These relationships

designate the actions to take when an event is triggered on a component. For details, see “Creating an interaction with the Interaction Wizard” on page 9-8.

- Servlet project template wizard – When creating a new project from a template, use the this wizard to quickly create a servlet.

How Visual Cafe keeps work synchronized

As you work in the Project window and Form Designer, Visual Cafe automatically adds the source code files for your forms to your project and generates the Java source code for components. As you modify a component, the underlying Java code changes.

When you’re editing the source code for a component directly in the Source window, any change that you make is interpreted and the source change is reflected in the Form Designer.

The power behind Visual Cafe component coding is its source-parsing technology. The Visual Cafe parser reads your Java source code in the background as the code is added to your project. This reading results in a symbolic object map of your source code. This map is used by the Class Browser and Hierarchy Editor to allow you to navigate and edit your Java classes. The Visual Cafe source parser is independent of the compiler and frees you from having to compile your code before obtaining class information.

When you edit source files outside of the Visual Cafe environment, the changes are reflected the next time you open the project in Visual Cafe. See “Adding custom code to a source file” on page 4-48 for important guidelines.

Understanding Visual Cafe components

In Visual Cafe, you add components to your forms to assemble applets and applications. *Components* are reusable objects that you can store in a library and add to one or more projects. Components can accept input from a user and perform specific actions (for example, a user could click a button that caused an animation to play). Components can also be used to display the results of an action (for example, clicking a button could display the results of a mathematical operation).

You can use standard graphical user interface (GUI) techniques for dragging and dropping components into and among other components. Visual Cafe also has unique drag-and-drop behavior in the Project and Source windows.

For more information about components, see Chapter 7, "Working with Components."

AWT Components

A set of *AWT (Abstract Windowing Toolkit)* components is included with Visual Cafe. These components include user-interface elements such as windows, dialog boxes, text-entry fields, and buttons. These components can be placed into your project as is, or customized to meet your needs.

For more information on AWT components, see Chapter 7, "Working with Components."

Swing components

Swing is the name given to the new set of Java visual components, which you can use to create user interface elements such as buttons, tables, lists, text fields, and so on, as well as applets, applications, windows, and dialog boxes. There is a large set of Swing components that includes one component for each AWT component plus many variations.

Swing components, which are part of the *Java Foundation Classes (JFC)*, have the following advantages over the older AWT components:

- They are "lightweight" components that require fewer system resources (see "About lightweight and heavyweight components" on page 7-6 for information).
- You can control the look and feel (appearance and behavior) of Swing components.
- A number of the components allow you to place an image icon on them.
- The Swing set of components includes types of components that are not included in AWT, such as a scrolling pane.

Swing components are 100% pure Java, which means you can easily subclass Swing components to create your own components. In addition, the fact that Swing components are written entirely in Java means that the

Java code determines what they look like, so that you can control their final appearance.

Swing also includes an applet component, a frame component for creating applications, a window component, a dialog component, and many other containers. Visual Cafe includes project templates for Swing applets and applications, and includes a Swing menu editor that lets you easily design, build, and change Swing menus.

For more information about Swing, see Chapter 8, "Working with JFC/ Swing Components."

JavaBeans components

You can also use JavaBeans components in Visual Cafe. *JavaBeans* is a portable, platform-independent Java component model. A JavaBeans component, or *Bean,* is a reusable component that can be manipulated visually in a builder tool such as Visual Cafe. Beans can be combined to create applications or applets.

In Chapter 10, "Working with JavaBeans Components," you'll learn how to create JavaBeans components and add them to your projects, as well as how to test your components and package them for distribution.

Forms hold your Java program together

Forms are containers for components that allow a user to interact with your program. A form might contain a series of buttons for a user to click, a field where a user enters text, or a menu from which a user selects commands.

You use *frames,* another type of form, when creating a stand-alone application. An example of an application that uses a frame is the JavaPad application, which is included on your Visual Cafe CD. Frames generally include toolbars, menu bars, and other windows.

Visual Cafe provides several tools to help you create forms, including the Form Designer.

For more information, see Chapter 7, "Working with Components."

Projects keep your work together

When developing an applet or application in Visual Cafe, you work mainly with projects. A *project* is a collection of files that make up your Java applet or application. You create a project to manage and organize these files.

When you open a new Visual Cafe project, all the default files required to begin development are created and included as part of your project. In addition, project elements may include HTML files, frames, and forms.

When you save your work, Visual Cafe saves the entire project to the Project folder as a `.vep` file, along with all the changes and other files you may have added to your project.

Chapter 3, "Working with Projects," describes projects in detail.

Using workspaces to customize your work environment

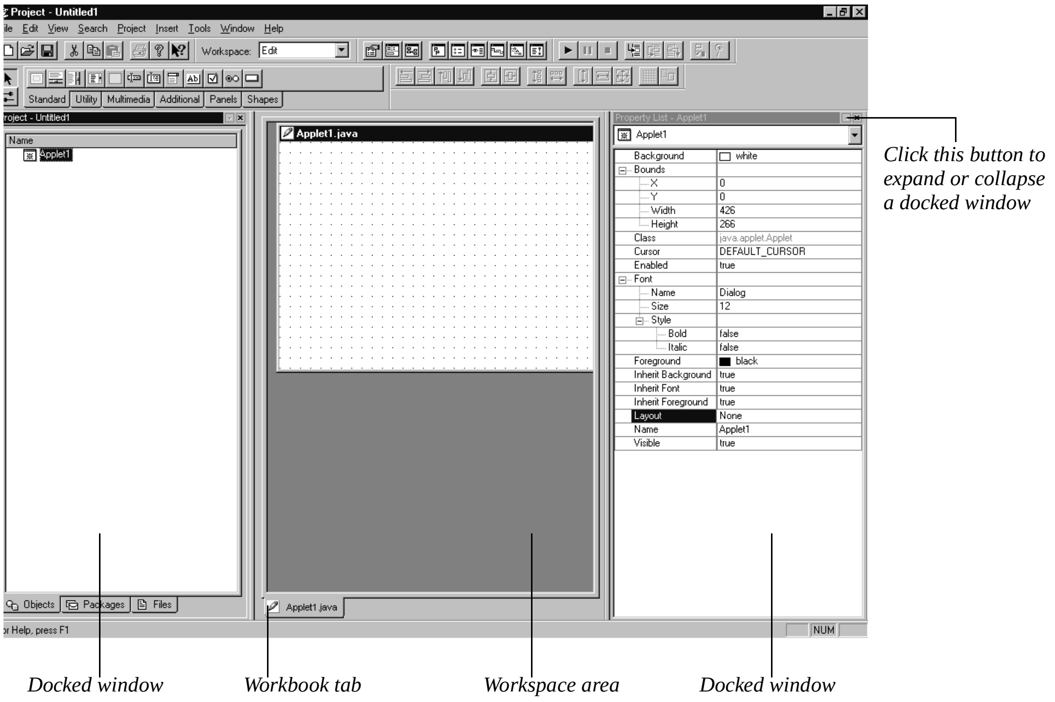
A *workspace* is a saved arrangement of windows. Because the various tools in Visual Cafe are displayed in many individual windows, workspaces are used to group windows that have related functions. For example, when you edit source code you can use a workspace that displays the Source window, the Messages window, and the Project window. When you're debugging your program, you can use another workspace that displays windows for the different debugging tools.

Workspaces provide a convenient way to switch from one screen layout to another. Workspaces are task-oriented, as opposed to project-oriented; you create workspaces for different tasks, such as editing or debugging.

Visual Cafe supports the extended mouse functions of the Windows 95, 98, and NT interface. Right-clicking on various components of the windows opens context-sensitive pop-up menus. As you work with Visual Cafe, try right-clicking on different parts of the screen. You may discover some useful shortcuts.

Visual Cafe has a customizable user interface. You can use **Single Document Interface (SDI)** or **Multiple Document Interface (MDI)** for your development environment. SDI was used exclusively in versions of Visual Cafe prior to version 2.5.

Here's an example of what the MDI environment looks like:



Background	◻ white
Bounds	
X	0
Y	0
Width	426
Height	266
Class	java.applet.Applet
Cursor	DEFAULT_CURSOR
Enabled	true
Font	
Name	Dialog
Size	12
Style	
Bold	false
Italic	false
Foreground	■ black
Inherit Background	true
Inherit Font	true
Inherit Foreground	true
Layout	None
Name	Applet1
Visible	true

While using MDI, you work with regular windows, which appear in the workspace area, and dockable windows, which can be docked along the edges of your Visual Cafe workspace or can be floating in the workspace area.

When using MDI mode, some window positions are saved with the workspace and the project. For more information on setting your workspace options, see "Setting environment variables in the sc.ini file" on page 3-72.

About applets, applications, servlets, and libraries

The program coding techniques used to create applets, applications, servlets, and libraries are fundamentally similar. Both applets and applications are created using the same basic programming concepts. Because a Web browser is responsible for running an applet, program instructions for applets have a different organization than the instructions for applications.

Applets

An **applet** is a Java program that runs on a Web page in a Java-enabled Web browser. Unlike an application (see the following section), an applet cannot run on its own; it must be run in a browser.

Applets, like all Java programs, are made up of source code that's compiled into a class file. A reference to the class file is placed in a Web page. The Web page is downloaded across the Internet using a Java-capable Web browser. As the bytecode contained in the class file is read, the Web browser's Java interpreter converts the bytecode into machine-specific instructions, executes the program, and hosts the running applet in a Web page.

With earlier versions of Java, applets always adopted the look and feel of the client machine's operating system and native interface controls. A Java applet would look and feel like it was written for the Macintosh when it ran in the Macintosh environment, look and feel like it was written for Windows when it ran in the Windows environment, and reflect the look of the various flavors of UNIX when run in a UNIX environment. However, with the new Swing components (see "Swing components" on page 2-6), you can decide whether your applet has a user interface that looks and acts the same way on all platforms, if the applet takes on the look and feel of the local system, or if the user controls the look and feel of the applet.

In order to provide security for users running applets from Web pages, applets can't read from or write to a user's local hard drive, and also have other restrictions. See "Limitations of applets" on page 2-12 for more information.

When you create an applet with Visual Cafe, you create a subclass of the class `Applet` or, for JFC/Swing applets, `JApplet`. These applet classes allow your applet to work within a Web browser and to use the capabilities of the AWT and JFC/Swing components, which are user interface elements that display to the screen and handle mouse and keyboard events. Although your applet can use as many classes as it needs, the main `Applet` or `JApplet` class triggers the execution of the applet. Its signature is as follows.

For a JFC/Swing applet:

```
public class JApplet1 extends com.sun.java.swing.JApplet{
    . . .
}
```

For an AWT applet:

```
public class myClass extends java.applet.Applet {
    . . .
}
```

Java requires that your `Applet` subclass be declared as `public`. This is true only of the main `Applet` class; all other classes may be declared `public` or `private` as you wish.

Browser versions needed to run Visual Cafe 3.0 applets

In order to run Visual Cafe version 3.0 applets in a Web browser, you need to have browsers that support JDK 1.1.

If you're running Netscape Navigator, you'll need:

- Netscape version 4.0 to 4.05 and the JDK 1.1 patch. You can download the patch from `http://developer.netscape.com`.
- Netscape version 4.06 or newer.

If you're running Microsoft's Internet Explorer, you'll need to have version 4.0 or newer.

If you're running Sun Microsystems' HotJava, you'll need version 1.1 or newer.

If you're going to be signing applets, you'll also need the Java plug-in 1.0 or newer. You can download the plug-in from `http://www.javasoft.com/products`.

Advantages of applets

Java applets have a significant advantage over applications because the Web browser software handles many of the functions that are required to make the applet run. Because the browser software automatically provides this functionality for applets, applets are an attractive type of Java program to work with.

Instead of running in a Web browser, applications run in a **Java Virtual Machine (Java VM)**, which contains a bytecode translator that converts a Java file into instructions the client machine can execute. Stand-alone applications have more overhead than applets; a stand-alone program must be able to start and stop, perform memory management, and handle display functions.

Limitations of applets

Sun designed Java to restrict the kinds of operations applets can perform. Limitations are imposed on applets so that a destructive program from a remote computer can't steal information from or cause damage to your system. To prevent applets from being destructive, Java enforces the following limitations:

- Applets can't read from or write to the file system of the computer viewing the applet. This prevents damage to files and the spread of viruses.
- Applets can't run any programs (or parts of programs like shared libraries or files) on the viewer's computer. This prevents an applet from calling destructive programs that don't have the limitations of the applet.
- Applets can establish connections only between the server computer where the applet is stored and the client's computer. This restriction prevents the applet from connecting the client's computer to another server without the viewer's knowledge.

Note that applets can run programs, access data, and write files on the server where they are stored; security considerations limit applet activities on the user's system rather than the applet's home system.

Java applications don't have these limitations and can be used to build fully functional programs. In addition, applications don't depend on the presence of a Web browser to run.

Applications

An **application** is a Java program that runs with a Java Virtual Machine that's installed on the client system. An application is not displayed on a Web page. Java applications can be built easily and quickly because the Java language includes useful features that are not standard in other languages. For example, Java comes with libraries of program code that make networking and graphics operations easy to write.

Java applications, unlike applets, can read from and write to the client machine's hard drive. And rather than being tied to the structure of a Web page, applications can use their own frames, title bars, and menus.

Because applications are Java programs that run on their own, applications can be as large or as small as you want them to be. The primary class of the application needs to have a `main` method. When Java runs the application, it calls this `main` method, and your program takes over from there.

The signature for the `main` method always looks like this:

```
public static void main(String args[]) {. . .}
```

or

```
public static void main() {. . .}
```

The parts of this line of code have the following meanings:

- The `public` keyword means that the method can be "seen" or used by other classes and components. Your application's `main` method must be declared `public` for the application to run.
- The keyword `static` specifies a storage class.
- The keyword `void` tells the `main` method not to return anything.
- The `main` method takes one argument, which is an array of strings. This array is used for command-line arguments outside of Visual Cafe. For example, the body of the `main` method contains all the code your application needs to start executing. This includes code for variable initialization or component instantiation.

Libraries

The *libraries* program type allows a collection of classes to be stored for use as a class library. Visual Cafe includes libraries and DLLs (Dynamic Link Libraries) to support native application and DLL development.

For more information about libraries, see Chapter 11, "Creating Native Win32 Java Applications."

Servlets

A *servlet* is a Java program that can be thought of as a server-side applet. That is, a servlet extends the capabilities of a server in the way that an applet extends the capabilities of a browser.

More precisely, a servlet is a Java class based on the Java `Servlet` interface. A servlet runs on a Web server and, in general, does the kinds of things that you might do with CGI scripts, with these advantages:

◆ Servlets are written in Java.

◆ Where a CGI script needs a new process to handle each request, a servlet can handle many requests at a time. The fact that there is only one instance of a servlet also means that the servlet only needs to load once.

◆ A servlet can share information between users.

Since a servlet runs on a server, it has no graphical user interface. Servlets can extend server functionality in any way you want, but they generally serve Web pages to users and read and respond to user input on HTML forms. The sample servlet in this chapter serves preexisting Web pages, but many servlets create Web pages in response to user input.

Most Visual Cafe users will never write servlets. They are useful only if you are running a Web server and need to extend its capabilities.

If you have the Visual Cafe Professional or Database Edition, see the *Visual Cafe Sourcebook* for an example of creating a servlet.

Debugging with Visual Cafe

After you write and execute your program, you might encounter errors, such as compile errors (code construction or syntax errors, for example), run-time errors that occur after you start the program (dividing by zero or writing to a file that does not exist, for example), or logic errors (the program does not do what you want it to do).

One of the most powerful tools in the Visual Cafe environment is the integrated debugger. The debugger allows you to watch your programs execute line by line. As your program executes, you can observe the various components of the program to see how they are behaving. By using the debugger, you can monitor:

- The values stored in variables
- Which methods are being called
- The order in which program events occur

For more information about debugging, see Chapter 6, “Debugging Your Program.”

Compiler choices

When you first create your project, you specify the type of compiler you want to use. Visual Cafe provides two compilers: Symantec’s Java compiler and Sun Microsystems’ Java compiler.

For information on compiling your programs, see Chapter 5, “Compiling and Deploying Your Project.”

Symantec’s Just-in-Time compiler

The Symantec Just-in-Time compiler (JIT) that comes with Visual Cafe was written specifically for the Java language and is generally faster than Sun’s command-line compiler. Instead of typing a string of commands at the command prompt, you click on a single icon to compile and execute a program.

Visual Cafe displays information in the Windows environment instead of at the DOS command line. Visual Cafe also provides useful messages that indicate how the compiling process is progressing.

In addition, Visual Cafe provides many GUI-based development tools not found in Sun’s JDK (see the following section).

Sun Microsystems’ Javac Compiler and JDK

When Sun Microsystems developed Java, they also created a compiler to convert Java source code into `.class` files. The Java compiler developed by Sun is called `javac.exe`, and is run from a DOS command line. To compile a Java program you need to run `javac.exe`, passing the name of the source file and any other required parameters.

Sun's Java compiler also comes with the Java Development Kit, or JDK. The JDK is a collection of tools to help compile, debug, and test Java programs. The JDK, like Sun's compiler, uses a command-line interface.

Overview of creating a Java program

Developing a Visual Cafe applet or application happens in two stages: designing the program and developing it. In the first stage, you design, create, and implement the graphical user interface of your program. You also make a very simple arrangement of all the components you need. In the second stage, you bring your project to life by adding and modifying source code, debugging, redesigning, and testing your project. If you're creating databound applets or applications, you must perform several additional steps as part of the design stage (see the *Visual Cafe Database Developer's Guide* for information).

Overview of creating an applet

When you start up Visual Cafe, Visual Cafe creates all the project files required for an applet or application, including source files and the project file. Visual Cafe then loads these project files, and you can immediately begin working on your project.

To create an applet:

1. Create a project with an applet template. Visual Cafe provides several templates for your use.

   See Chapter 3, "Working with Projects."

2. Design the user interface by adding forms and components to your project, customize the component properties, and create component interactions.

   This process is described in detail in Chapter 7, "Working with Components."

3. If necessary, modify the Java source code.

   See Chapter 4, "Working with Source Code."

4. Set the project options.

   See Chapter 3, "Working with Projects."

5. Run the applet in the AppletViewer.

   This step is explained further in Chapter 5, "Compiling and Deploying Your Project."

6. If necessary, debug the applet.

   Chapter 6, "Debugging Your Program," contains detailed information on debugging in Visual Cafe.

7. Add the applet to your HTML page. You may also want to test-run the HTML page on your local machine, then across an intranet or the Internet.

   For details, see Chapter 5, "Compiling and Deploying Your Project."

8. Deploy the applet.

   Deploying is described in Chapter 5, "Compiling and Deploying Your Project."

Overview of creating an application

Creating an application in Visual Cafe is very similar to creating an applet.

To create an application:

1. Create a project with an application template. Visual Cafe provides several templates for your use.

   See Chapter 3, "Working with Projects."

2. Design the user interface by adding forms and components to your project, customize the component properties, and create component interactions.

   This process is described in detail in Chapter 7, "Working with Components." For information on interactions, see Chapter 9, "Working with Events and Interactions."

3. If necessary, modify the Java source code.

   See Chapter 4, "Working with Source Code."

4. Set the project options.

   See "Customizing a project" on page 3-55.

5. Test-run the application.

Compiling and running a project is explained in Chapter 5, "Compiling and Deploying Your Project."

6. If necessary, debug the application. You should also test the application outside of Visual Cafe once it's been debugged.

   Chapter 6, "Debugging Your Program," contains detailed information on debugging in Visual Cafe.

7. Deploy the application.

   Deploying is described in Chapter 5, "Compiling and Deploying Your Project."

Now that you've been introduced to the basic features of Visual Cafe, you're ready to start working on a project. The next chapter provides detailed, step-by-step instructions for creating a project.

CHAPTER 3

Working with Projects

This chapter shows you how to create a project in Visual Cafe. It describes the elements that make up a Visual Cafe project and includes step-by step instructions for the following tasks:

- Creating a project
- Creating a subproject
- Using project templates
- Adding files to a project
- Sharing files among projects
- Using HTML files
- Customizing a project
- Customizing the Visual Cafe environment

About projects

Visual Cafe provides an easy-to-use yet sophisticated system for managing the many files that make up a program. The files needed to create a Java program are part of a Visual Cafe project. A **project** is an organized collection of related files that are used to construct a Java program, which can be an applet, application, JavaBeans component, or library. The project is the core of the Visual Cafe development environment; it holds all the elements of the program you're developing. For example, in a project could contain an animation applet, a word processing application, or the Web pages and applets that make up an entire Web site. Projects provide vital organization for your programs, particularly as they grow more complex.

Within a project, you'll set up an arrangement of windows called a workspace. Because the various tools in Visual Cafe are displayed in many individual windows, workspaces are used to group windows that have related functions, such as editing or debugging. For more information about workspaces, see "About workspaces" on page 3-18.

In a typical project there are source code files, class files, and documentation files. A single project generally is used to create a single **target,** such as the application or applet that Visual Cafe builds.

About the Project window

A project is the starting point of every Java applet and application created in Visual Cafe. The **Project window** shows each item in a project. Visual Cafe lets you choose among three ways to view your project:

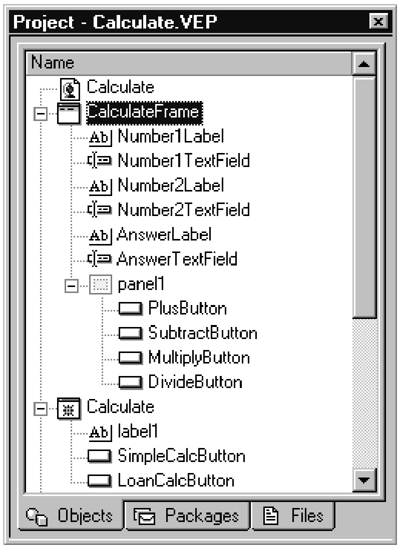
- as a list of objects (and, optionally, HTML files)
- as a list of packages
- as a list of files

The Project window is divided into three views: Objects, Packages, and Files. These views are represented by tabs at the bottom of the window. By clicking on a tab, you can organize the information in the Project window according to your needs. If you wish, you can remove a tab or change the default tab. See "Changing the Project window's tab display" on page 3-8 for information on modifying tabs.

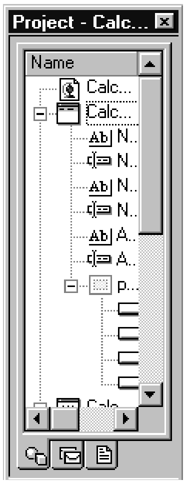
Before you can construct an applet or application with Visual Cafe, you need to create a new project, using one of Visual Cafe's predefined project templates (see "About project templates" on page 3-25 for information on templates). Then the Project window opens with the project name in the title bar.

When you first open Visual Cafe a new project is automatically created, based on the default project template, AWT Applet. Each new project is named "Untitled" until you save it with a name. If you've already created some projects, you can set Visual Cafe to display the last project you viewed when you open Visual Cafe (see"Creating a new project" on page 3-36).

Here's an example of a Project window, with the Objects view displayed:



You'll notice that when you lessen the width of the Project window past a certain point, the text no longer appears on the tabs, but the symbols do, as shown here:



As in Windows Explorer, you can click the column header to sort by that column. Clicking the column header again reverses the sort direction.

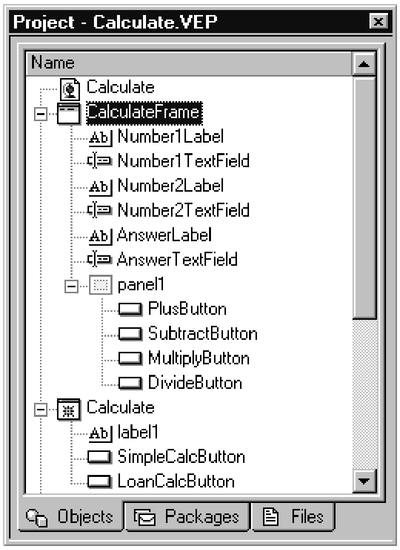
About the Project window's views

This section describes the three views that are found in the Project window: Objects, Files, and Packages.

The Objects view

When you click the Project window's Objects tab, the Objects view appears. In this view you see the components in your project and any HTML files that you've added to the project. Java **components** are user interface elements such as windows, menus, buttons, lists, and so on. For information on components, see Chapter 7, "Working with Components."

Here's what the Objects view looks like:



```
Calculate
CalculateFrame
    Number1Label
    Number1TextField
    Number2Label
    Number2TextField
    AnswerLabel
    AnswerTextField
    panel1
        PlusButton
        SubtractButton
        MultiplyButton
        DivideButton
Calculate
    label1
    SimpleCalcButton
    LoanCalcButton
```

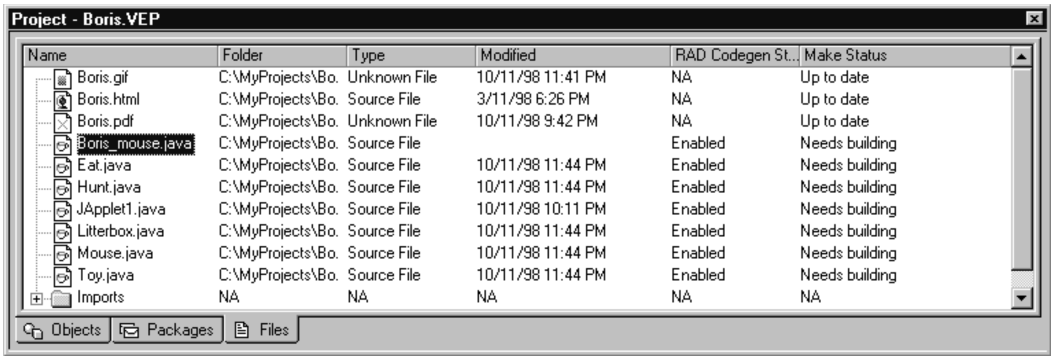
Some components can contain other components, such as an application window that contains a button. A component that contains other components is called a *container*. In the Project window, the components in a container appear subordinate to the container, like a file system display that shows files subordinate to their folders. The containers at the top level are separate Java files in your project (called Visual Cafe *forms*), while the components in the containers are Java code within the container Java file.

All Visual Cafe components are organized in the Visual Cafe *Component Library*. You can use the *Component Palette* to display your most frequently used components. For more information, see Chapter 7, “Working with Components.”

The Objects view is the view many people use most often during program development. With it, you can keep track of the components your project contains, as well as open a variety of editors. In addition, one way to add a standard Java component or a Visual Cafe component to your project is to drag it from the Component Palette to the Project window. When you do so, an instance of the component class called an **object** appears. See “Viewing the components and HTML files in a project” on page 3-48, “Opening editors from the Project window” on page 3-10 and “About files in a project” on page 3-42 for more information.

The Files view

When you click the Project window’s Files tab, the Files view appears. The Files view lists all the files in your project. At the top level are the files for the top-level components in your project, as well as any HTML files or other files you’ve added.



Project - Boris.VEP

Name	Folder	Type	Modified	RAD Codegen St...	Make Status
Boris.gif	C:\MyProjects\Bo.	Unknown File	10/11/98 11:41 PM	NA	Up to date
Boris.html	C:\MyProjects\Bo.	Source File	3/11/98 6:26 PM	NA	Up to date
Boris.pdf	C:\MyProjects\Bo.	Unknown File	10/11/98 9:42 PM	NA	Up to date
Boris_mouse.java	C:\MyProjects\Bo.	Source File		Enabled	Needs building
Eat.java	C:\MyProjects\Bo.	Source File	10/11/98 11:44 PM	Enabled	Needs building
Hunt.java	C:\MyProjects\Bo.	Source File	10/11/98 11:44 PM	Enabled	Needs building
JApplet1.java	C:\MyProjects\Bo.	Source File	10/11/98 10:11 PM	Enabled	Needs building
Litterbox.java	C:\MyProjects\Bo.	Source File	10/11/98 11:44 PM	Enabled	Needs building
Mouse.java	C:\MyProjects\Bo.	Source File	10/11/98 11:44 PM	Enabled	Needs building
Toy.java	C:\MyProjects\Bo.	Source File	10/11/98 11:44 PM	Enabled	Needs building
Imports	NA	NA	NA	NA	NA

Click Name, Folder, Type, Modified, RAD Codegen Status, or Make Status to sort by that attribute. Click again to reverse the sort order. If you do not see all the headings, resize the Project window. See “Using files in a project” on page 3-42 for more information.

You can also turn RAD on or off for a file. For more information, see “Enabling and disabling RAD and automatic code generation” on page 4-45.

Subprojects are visible in the Files view, but are not expandable. To open a subproject, double-click the project icon to open it in a separate Project window. You can double-click a file to open it in the Source window.

The Files view's `Imports` folder contains all the Java source files for the packages your project uses. Whereas in the Packages view the files are listed by package, in the Imports folder the files are listed alphabetically. Imports can be shown or hidden.

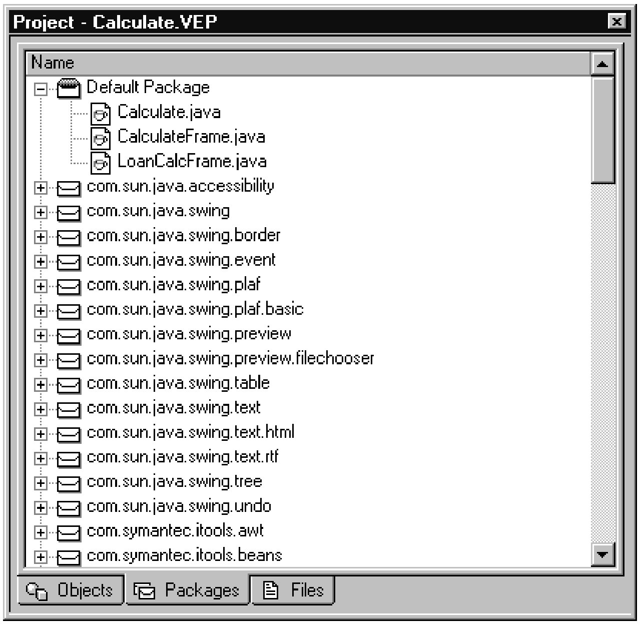
The Packages view

A Java **package** is a group of related classes that can be used by programs that import the package or any file in the package. A package is similar to a C library. The Project window's Packages view lists the Java source files in your project, grouped as Java packages. The Packages view always shows the standard Java packages that Java programs require. It can also display Symantec Visual Cafe packages and packages you add, including your own packages or third-party packages.

Java source code is organized into packages. Each part of a package name generally refers to a hierarchical directory structure. For example, `COM.sun.java.swing` is in the directory structure `/com/sun/java/swing`. If you're going to distribute your packages, you should create a globally unique name based on an Internet domain name. For example, `sun.COM` is specified as `COM.Sun`. This first part of the name is in uppercase letters; if the first part isn't in uppercase letters, it's for local use, except for packages that are part of the Java language and system (which start with `java`). Packages help prevent naming conflicts. For example, two Java files could have the same names, but as long as they're in different packages, there's no naming conflict.

When you add items to a project, a default package that contains the Java files for those objects appears. If you add a Visual Cafe component, the Symantec package also appears in the Packages view; it contains the Java source file for each Visual Cafe component you include (specifically, it contains the source code for the component class that your project object is based on).

The Packages view looks like this:



```
Default Package
    Calculate.java
    CalculateFrame.java
    LoanCalcFrame.java
com.sun.java.accessibility
com.sun.java.swing
com.sun.java.swing.border
com.sun.java.swing.event
com.sun.java.swing.plaf
com.sun.java.swing.plaf.basic
com.sun.java.swing.preview
com.sun.java.swing.preview.filechooser
com.sun.java.swing.table
com.sun.java.swing.text
com.sun.java.swing.text.html
com.sun.java.swing.text.rtf
com.sun.java.swing.tree
com.sun.java.swing.undo
com.symantec.itools.awt
com.symantec.itools.beans
```

You can expand and collapse your view of a package as you would for a hierarchy of folders and files in Windows.

In the Packages view, you can drag and drop files between packages, drag files from a file source — Windows Explorer, for example — and add them to a project, and drag files to the Recycle Bin to remove them from a project. Any of these actions can be undone.

You can also make other packages available to Visual Cafe projects by adding them to the Visual Cafe class path. Then if you add a Java `import` statement or use part of the new package in your Java source code, the package appears in the Packages view. For example, you could add third-party or your own packages containing components or utilities. You can set the class path for a project or for the Visual Cafe environment. For more information, see “Specifying source-file search paths for a project” on page 3-62, “Working with components in a project” on page 3-48, and “About the system path” on page 3-14.

Note: Classes that have not been added to a package display under the default package. Only source files that you've added to the project are listed.

The Packages view shows resource files, not just Java classes. You can change the package of a resource file by dragging and dropping in the package view. When Visual Cafe builds your project, the resource file is copied into its corresponding location on disk, or into the JAR archive.

Note: Remember to keep the directory structure of your package intact, and make sure that your file names use the same upper- and lower-case letters, exactly as they were before copying the package.

If you declare your application or applet as a part of another package, you must specify the output folder in the Package Destination line of the Directories tab in the Project options window. See "In this section you'll learn how to set some common project options in the Project Options dialog box. The following topics are discussed:" on page 3-58.

Changing the Project window's tab display

You can enable or disable a Project window tab (Files, Objects, or Packages), and you can also set which tab appears by default when you open the Project window.

To enable or disable a Project window tab:

1 Right-click a tab in the Project window.

Available tabs are Files, Objects, and Packages.

A pop-up menu appears.

2 Choose the name of the tab you want to enable or disable.

A checkmark indicates that the tab is enabled.

To make a Project window tab the default tab:

1 Right-click a tab in the Project window.

Available tabs are Files, Objects, and Packages.

A pop-up menu appears.

2 For the tab you want to make the default tab, choose either Make Objects Default, Make Packages Default, or Make Files Default.

The Project window tab you selected will now be the default view for all projects.

Dragging and dropping into the Project window

You can use standard graphical user interface (GUI) techniques for dragging and dropping components and files into the Objects view of the Project window.

When you're copying or moving components, if a box appears around a container in the Project window the component is inserted in the container. If a line appears under a component, the new component appears after the component that's underlined. A plus sign (+) appears over the cursor when a copy operation is being performed.

Containers are placed at the top level or in another container, depending on where you drag the component or on the characteristics of the component. Containers at the top level mean a new `.java` file is added to your project; components added to a container mean that code is generated in the `.java` file for the top-level container.

Visual Cafe does not allow inappropriate copies and moves, such as dropping a component into a container when that component must be at the top level, or trying to move a component when files are in use.

Drag from…	**Into…**	**Result…**
Component Library or Palette	Project window	Copies the component to the new project (in other words, instantiates the component).
Project window	Same Project window	Moves the location of the item. Alternatively, you can press CTRL and drag an item to copy it. For example, you can move a component to another container or copy a component within the same container. You can also reorder the components in the Project window list, which affects how components overlap on a form (the z-order), the tab order, and the order they are declared in the Java code.

Drag from…	Into…	Result…
Project window	Different Project window	Copies the file or component to the new project.
Form Designer	Project window	Within the same project, moves the location of the component; or press CTRL and drag to copy a component. When dragging to a different project, copies the component.
Menu Designer	Project window	Within the same project, moves the location of the menu item; or press CTRL and drag to copy a menu item.When dragging to a different project, copies the menu item.
Windows Explorer or other file system window	Project window	Adds the file to the project. (However, it does not copy the file to the project folder.)

Opening editors from the Project window

The Project window helps you to quickly edit the items in a project. To access other editor windows from the Project window:

Double-click…	To get this editor…
A component	Form Designer
A menu	Menu Designer
A Java or HTML file	Source window
An item in the Form Designer	Source window

Components and menus are displayed in the Objects view.

Java files are displayed in the Packages and Files views.

HTML files are displayed in the Objects and Files views. If you've installed Visual Page from the Visual Cafe CD, double-clicking an HTML file in the Object view launches Visual Page.

About the contents of a project

Visual Cafe creates several files that contain information about your project and stores them in the project folder:

File extension	Description
`.vpj`	The Visual Cafe project file
`.vep`	A Visual Cafe file that contains project options and a list of files in the project
`.ve2`	A Visual Cafe file that contains secondary project information
`.cdb`	A compiled database that Visual Cafe uses to track compilation dependencies (created after compilation)

These files aren't needed for deployment. The name of the file with the `.vep` extension displays in the title bar of the Project window; the rest of the project-related files don't appear in the Project window.

Visual Cafe can also create files with the following extensions and stores them in the project folder:

File extension	Description
`.java`	A Java source file, such as for an applet
`.class`	A compiled version of a Java source file
`.obj`	An intermediate file produced when you compile a Java source file to create a native Win32 application or DLL (not available in Visual Cafe Standard Edition)
`.lib`	A library file used with native Win32 applications and DLLs (not available in Visual Cafe Standard Edition)
`.html`	An HTML file
`.properties`	A text file containing properties for localized code (resource bundle)

The only way an HTML file will appear in the Project window is if you manually add it. You can add HTML files to the project as a helpful organizational tool, but it's not required that you do so. See "Adding an existing file to a project" on page 3-44 and "About HTML files in Visual Cafe" on page 3-50 for more information.

You can double-click a Java or HTML file in the Project window to open that file in a Visual Cafe editor. If you've installed Symantec Visual Page, HTML files will display in the Visual Page HTML editor. For more information about opening files from the Project window, see "Opening editors from the Project window" on page 3-10.

When you're ready to deploy your program, you can use Visual Cafe's automatic deployment features to compile and deploy your files. This way you can include everything that's a part of your project, including HTML files, PostScript files, graphics files, Adobe Acrobat PDF files, or any other file. You can also choose to generate a list of properties to use with localization. For more information on localization, see Chapter 13, "Localizing Your Java Programs." If you deploy your files by yourself, then you'll need to compile them first. For more information on deployment, see Chapter 5, "Compiling and Deploying Your Project."

Source files

Visual Cafe can process Java (`.java`), class (`.class`), properties (`.properties`), and documentation files. Java, properties, and documentation files are text files.

The Java source files and compiled Java files have the same file names but different extensions. You see only the `.java` files in the Project window, because they are used for development, while `.class` and `.properties` files are used for deployment.

Additional projects

Visual Cafe allows a project to contain another project. Including other projects lets you group sets of related project items and access all included projects' items within Visual Cafe. Included projects are built when the project containing them is built. Thus, you can develop a suite of applications by having one project for each application and one additional project that includes all the individual projects; the entire suite of applications can then be built with one command.

Documentation files

You can add documentation files to your project to make them readily accessible during development. ReadMe files and Javadoc-generated HTML files are ways to document your program. Because the documentation files are included in the project, you can read or modify them at any time. These files are for your reference only; they will neither be included in the final target nor be involved in any way with building.

If you use the AutoJAR command (from the Project menu), the HTML files will end up in the JAR file as well.

Documentation files need to be saved as text files.

For more information about documenting your program, see “About Javadoc” on page 4-59.

Organizing files and folders

The central element of a project is the **project file**. The project file contains all the information you need to manage the project, such as the locations of items in the project, as well as additional information such as compiler options and browser data.

In general, most of the contents specific to a project, together with the project file, are kept in a folder called the **project folder**. However, project contents don’t all have to reside in the project folder. In addition, a project may include items that are located in the project folder of another project; this arrangement allows projects to share code.

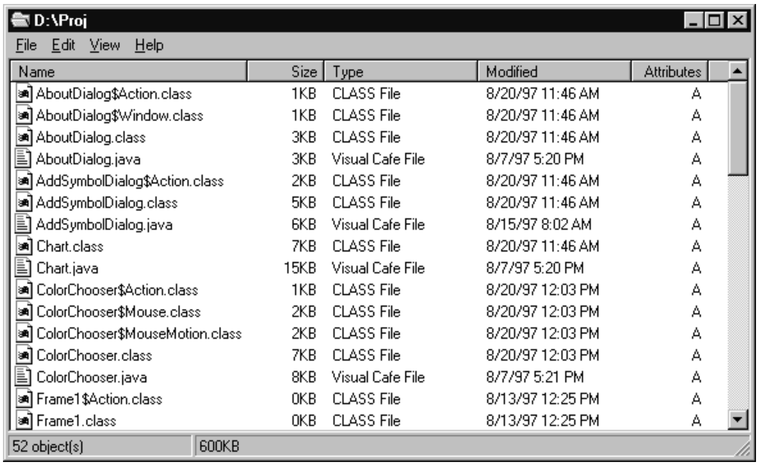
When you organize a target’s files as a project, Visual Cafe can assume full management responsibility. In contrast to traditional “make” systems, this strategy frees you from the bookkeeping involved in accessing the contents of a project and building the target. Because Visual Cafe keeps track of all project contents, the features of Visual Cafe are smoothly integrated. For example, if an error occurs during compilation, you can click to open a Source window that contains the source code, with the questionable line of code highlighted.

Also, Visual Cafe automatically determines those items in a project that need to be rebuilt following changes to any project contents.

Installing Visual Cafe establishes a specific folder-organization plan. This plan is set up to allow Visual Cafe to quickly and unambiguously locate your project's contents.

The first time you save a project, the files it contains are stored in the folder you specify. For efficient project management, you should store all the files of a given project in a single folder dedicated to that project, usually referred to as the **project folder.** Doing so will make deploying your applets, applications, and Web pages much easier.

Here's an example of a project folder's contents:



D:\Proj

Name	Size	Type	Modified	Attributes
AboutDialog$Action.class	1KB	CLASS File	8/20/97 11:46 AM	A
AboutDialog$Window.class	1KB	CLASS File	8/20/97 11:46 AM	A
AboutDialog.class	3KB	CLASS File	8/20/97 11:46 AM	A
AboutDialog.java	3KB	Visual Cafe File	8/7/97 5:20 PM	A
AddSymbolDialog$Action.class	2KB	CLASS File	8/20/97 11:46 AM	A
AddSymbolDialog.class	5KB	CLASS File	8/20/97 11:46 AM	A
AddSymbolDialog.java	6KB	Visual Cafe File	8/15/97 8:02 AM	A
Chart.class	7KB	CLASS File	8/20/97 11:46 AM	A
Chart.java	15KB	Visual Cafe File	8/7/97 5:20 PM	A
ColorChooser$Action.class	1KB	CLASS File	8/20/97 12:03 PM	A
ColorChooser$Mouse.class	2KB	CLASS File	8/20/97 12:03 PM	A
ColorChooser$MouseMotion.class	2KB	CLASS File	8/20/97 12:03 PM	A
ColorChooser.class	7KB	CLASS File	8/20/97 12:03 PM	A
ColorChooser.java	8KB	Visual Cafe File	8/7/97 5:21 PM	A
Frame1$Action.class	0KB	CLASS File	8/13/97 12:25 PM	A
Frame1.class	0KB	CLASS File	8/13/97 12:25 PM	A

52 object(s) 600KB

The Java and Symantec packages are automatically available during the development process. These files are not added to your project when you save it the first time; there are a number of ways you can make imports from these packages available to your applets and applications during deployment.

For more information on deployment, see "Deploying your applet" on page 5-32.

Visual Cafe looks for your project's items in the folders of two paths, the **system path** and the **project path**.

About the system path

Components that will be used in many projects should be placed in folders in the *system path*. In the folder where Visual Cafe is installed, there is a

`Java Libraries` folder. Within that folder, there is a `classes` folder. The default system path is the `classes` folder along with all its subfolders.

About the project path

Components that are specific to a particular project belong in its **project path.** The project folder (the folder that contains the project file), along with all subfolders it contains, is the default project path. You can add paths to the project path, as well as modify the project path, by using the Search Directories option from the Project Options dialog box. For more information, see “Specifying source-file search paths for a project” on page 3-62 and “Working with components in a project” on page 3-48.

Typically, a project file resides in a project folder along with all files specific to that project. The folder may also contain subfolders. Setting up your project contents in this way helps reduce the time it takes Visual Cafe to search for files, and reduces the likelihood of confusion due to duplicate file names. You can expect to have many project folders.

When you first add a file to a project, Visual Cafe notes the tree (folder hierarchy) to which the file belongs. Thus, you can move files in and out of folders and create and rename folders without having to tell Visual Cafe exactly where the files are located. If you move files later on, Visual Cafe first looks in this tree.

About multiple projects and subprojects

In some cases you’ll want to work on more than one project at once. Visual Cafe allows you to work on multiple projects, and helps you keep track of the files they contain.

About multiple projects

Visual Cafe allows you to have multiple projects open at the same time. You can switch between projects by selecting an item from a list in the Windows menu that shows currently open windows and the projects they’re associated with. The File menu contains a list of recently opened projects (which aren't currently open); you can choose from this list to quickly open a recent project.

When working with several projects at the same time, it's important to know which project will be affected by project-related commands you might choose. Visual Cafe applies the following rules to determine the project that will be affected:

- ◆ If the frontmost window is a Project window, the command affects the project the window belongs to.
- ◆ If the frontmost window is not a Project window, the command affects the project that was active when this window was opened.

Viewing active projects

The active project's name is displayed in the Visual Cafe title bar. The Class Browser is also project-dependent. If you have a Class Browser open for each open project, you can identify which project is active by looking at the Visual Cafe title bar for each Class Browser window you select.

This feature is helpful when viewing `.java` files. For example, if you open two or more `.java` files that are in two or more different projects, look at the Visual Cafe title bar to see which project is associated with them. Being able to tell which `.java` file belongs to which project is helpful when debugging, for example, because when you set breakpoints they're associated with that project.

About subprojects

Visual Cafe lets you add subprojects to a project. A *subproject* is a project that is a part of another project. When you compile the parent project, the subproject gets compiled and saved as well. When you run a project that has subprojects, only the parent project is run. To run and debug a subproject, you need to open the project in its own Project window.

You add subprojects to a project by using the Project Files dialog box (see the next section for details). You choose the project file type and then select a project file and add it. Subprojects appear only in the Files view of the Project window.

Note: There is currently no particular order in which subprojects are compiled, although they are compiled every time the parent project is.

Using subprojects

To add a subproject to a project, follow the steps below to add the `.vep` file of another project to the current project. Also see “Opening an existing project” on page 3-37 and “Adding an existing file to a project” on page 3-44.

A quick and easy way to add a subproject to a project is to drag a project’s `.vep` file into the Project window (Files view); and the project you dragged becomes a subproject of the target.

You can also add subprojects by means of the Project Files dialog box.

To add a subproject to a project:

1. Open the project you want to add a subproject to.
2. From the Insert menu, choose Files Into Project.

   The Project Files dialog box appears.
3. Navigate to the desired project folder.
4. Make sure Project files (*.vep) is shown in the Files of type field.

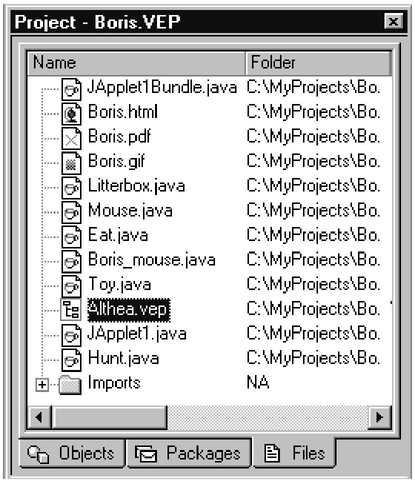
   The available projects display.
5. Select a project.

   This project will become a subproject of the one selected in step 1.
6. Click Add.

   The project is now a subproject of the current project.
7. Click OK.

The project you added becomes a subproject of the parent project. It appears in the Files view of the Project window.

Subprojects can't be expanded in any of the Project window views to show the visual elements within the subproject. Double-clicking the subproject object opens the project in its own Project window.



Name	Folder
JApplet1Bundle.java	C:\MyProjects\Bo.
Boris.html	C:\MyProjects\Bo.
Boris.pdf	C:\MyProjects\Bo.
Boris.gif	C:\MyProjects\Bo.
Litterbox.java	C:\MyProjects\Bo.
Mouse.java	C:\MyProjects\Bo.
Eat.java	C:\MyProjects\Bo.
Boris_mouse.java	C:\MyProjects\Bo.
Toy.java	C:\MyProjects\Bo.
Althea.vep	C:\MyProjects\Bo.
JApplet1.java	C:\MyProjects\Bo.
Hunt.java	C:\MyProjects\Bo.
Imports	NA

About workspaces

The various tools in Visual Cafe are displayed in many individual windows; **workspaces** let you group the windows that have related functions. For example, when you're editing source code you might use a workspace that displays the Source window, the Messages window, and the Project window. When you're debugging your program, you might use another workspace that opens windows for the different debugging tools.

Workspaces are task-oriented as opposed to project-oriented. You create workspaces for different tasks, such as designing, editing, or debugging. You can then access the workspaces from the Workspace toolbar or from the Window menu under Workspaces:

Visual Cafe supplies you with two built-in workspaces: Edit and Debug. The Edit workspace automatically loads when you're creating or editing your program, and opens windows such as the Form Designer, Property List, and Project window.You can also manually open the Source window and other windows from within the Edit workspace. The Debug workspace automatically loads when you run your application, and opens the necessary debugging windows. When the debugging process ends, the development environment returns to the Edit workspace.

You can create, delete, or rename workspaces. Only global-view windows are saved in a workspace.

Workspaces can be loaded automatically when you run a project and when you stop the project. Workspaces are saved dynamically. For example, if the Property List window is open in the Edit workspace and you close that window while in the Edit workspace, the change is saved permanently. This feature ensures that as you run and stop your program, the window state remains as you last left it.

You can create, delete, and rename a workspace by using the Workspaces menu option in the Window menu. Visual Cafe automatically saves changes to a workspace configuration when you exit the workspace.

About dockable windows in a workspace

You can now choose to use the Multiple Document Interface (MDI) for your development environment. This allows you to dock and undock some windows. For more information, see the following section, “Working with the MDI window system.”

Using workspaces

You can choose options on how to use the MDI system, or how to customize other aspects of your workspace. In this section you’ll learn how to dock and otherwise organize windows in a workspace.

Working with the MDI window system

You can customize your workspace even further by retaining your window preferences between Edit and Debug development modes. You can control your workspace by docking or undocking windows and by using the Workspace and Workbook tabs (a Workbook is a Windows user-interface item that displays files on a row of tabs for easy access). Docking a window makes the window stay against the edge of the Workspace area. If you like to have different windows open and vary their locations depending on what development mode you’re working in, you can keep these preferences, even after closing a project or Visual Cafe. For each project, you can retain your settings for the Project, Source, Class Browser, and Hierarchy Editor windows.

These options are available to you when you are in **Multiple Document Interface (MDI)** mode. If you do not turn MDI on, your workspace will look the same as it did in versions of Visual Cafe prior to version 2.5.

While using MDI, you work with regular windows, which appear in the workspace area, and dockable windows, which can be docked along the edges of your Visual Cafe workspace or can be floating in the workspace area.

When using MDI mode, some window positions are saved with the workspace and the project. For more information on setting your workspace options, see “Modifying workspaces” on page 3-23.

When you’re using MDI, you can do the following:

- ◆ Turn Docking View on or off for a dockable window
- ◆ Dock a window floating in the workspace area
- ◆ Change the size of a docked window
- ◆ Prevent a dockable window from docking while dragging it
- ◆ Expand or collapse a docked window
- ◆ Organize the placement of windows in the workspace area
- ◆ Toggle the display of Workbook tabs
- ◆ Activate a window in the workspace area
- ◆ Toggle the display of the status bar

These tasks are discussed in this section.

To enable MDI:

1. From the Tools menu, choose Environment Options, then click the General tab in the Environment Options dialog box.
2. Select MDI Development Environment to enable MDI, or deselect it to turn MDI off.

The workspace area contains non-dockable windows, such as the Form Designer and the Source window, and dockable windows. These dockable windows can have Docking View enabled or disabled. Non-dockable windows appear in the workspace area; dockable windows can be docked along the edges of your Visual Cafe workspace or can float in the workspace area. The windows that are dockable are:

- Breakpoints
- Call Stack
- Component Library
- Find in Files results
- Messages
- Project
- Property List
- Threads
- Variables
- Watch

Note: Docked windows have the same highlight color in the title bar as a non-dockable window that is selected. To ensure that a docked window is the active window, click it.

Tip: If you have maximized a non-dockable window, such as the Source window, you can restore it to its previous size by clicking the innermost Restore button.

To turn Docking View on or off for a dockable window:

- Do either of the following:
  - ❖ Right-click and select Docking View from the pop-up menu.

  or

  - ❖ Activate the window, then choose Docking View from the Window menu.

When you dock a window that you've docked before, the window will dock to its previous size and location, unless you've turned toggling on and off in between docking it.

To dock a window that's floating in the workspace area:

- Do either of the following:
  - ❖ While Docking View is enabled, drag the window to the edge of the workspace.

  or

  - ❖ Double-click the title bar of the window.

A docked window takes up an entire side of the workspace, unless another window is docked along the same side.

To undock a window:

- Do any of the following:
  - ❖ Double-click the title bar.
  - ❖ Drag the window into the workspace area.
  - ❖ Activate the window, then choose Docking View from the Window menu.

To change the size of a docked window:

- Move the cursor over the edge of the window until the cursor changes to a sizing cursor, then click and drag to the desired size. You can drag the edge that's facing the center of the main window.

To prevent a dockable window from docking while dragging it:

- Press CTRL while dragging the window.

To expand or collapse a docked window:

If you have more than one window docked along an edge, you can expand or collapse one of the windows by clicking the triangle button in the corner of the window.

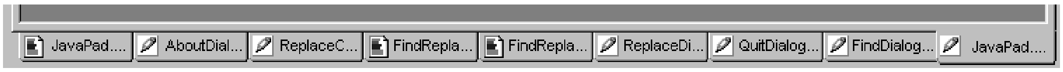
Expanding and collapsing windows along an edge will resize the windows so that they all use equal amounts of space.

To organize the placement of windows in the workspace area:

- From the Window menu, choose either Cascade, Tile Horizontally, or Tile Vertically. You can also drag the windows within the workspace.

This works only for undocked or undockable windows, such as the Form Designer.

At the bottom of the workspace area, you can display Workbook tabs — one for each open window (either non-dockable windows or dockable windows with Docking View disabled). An example is shown here:



Clicking a tab activates that window in the workspace area.

To toggle the display of Workbook tabs:

1. Choose Workbook from the View menu.

   The workbook tabs display.
2. Choose Workbook again to hide the workbook tabs.

To activate a window in the workspace area:

◆ Choose the window from the Window menu. You can also click the window, click a Workbook tab, or choose Next or Previous from the Window menu.

To toggle the display of the status bar:

◆ Choose Status Bar from the View menu.

Modifying workspaces

You can create, delete, and rename a workspace by using the Workspaces menu option. Visual Cafe automatically saves changes to a workspace configuration when you exit the workspace.

To change to a different workspace:

◆ Use either of these methods:

  ❖ From the Window menu, choose Workspaces, then choose one of the workspace names displayed in the submenu.

  ❖ From the Workspace toolbar, select the appropriate workspace name.

To save your current window arrangement as a new workspace:

1. Configure the screen as you like by opening the windows you need and positioning and sizing them to suit your requirements.
2. From the Window menu, choose Workspaces, then choose New.
3. In the New Workspace dialog box, type a new name.
4. Click OK.

Visual Cafe creates a new workspace and displays it in the toolbar's Workspace field. The workspace is also added to the Workspace list in the Window menu.

To rename a workspace:

1. From the Window menu, choose Workspaces, then choose Rename from the submenu.
2. In the Rename Workspace dialog box, type a new name.
3. Click OK.

Visual Cafe changes the workspace name and displays it in the toolbar's Workspace drop-down list box.

To delete a workspace:

◆ From the Window menu, choose Workspaces, then choose Delete from the submenu.

The workspace is deleted and the next workspace in the listing is activated.

Caution: Deleting a workspace immediately deletes the workspace named in the toolbar's Workspace drop-down list box.

You cannot delete the last remaining workspace.

Controlling toolbar position and visibility

Visual Cafe's toolbars are at the top of the Visual Cafe main window. You can control toolbar position and visibility.

To float a toolbar:

1. Drag the toolbar from the top of the Visual Cafe window onto your desktop.

2 Double-click somewhere in the toolbar's background.

To dock a toolbar:

1 Drag the toolbar to the top of the Visual Cafe window.

2 Double-click somewhere in the toolbar's background.

To hide or show a toolbar:

◆ Do either of the following:

- ❖ Right-click at the top of the Visual Cafe window and select the toolbar's name. Select the toolbar's name to show the toolbar, or deselect it to hide it.
- ❖ If the toolbar is undocked, click the close box on the toolbar to hide it.

About project templates

This section describes how to use project types, and the templates that are contained in each, when creating a project. It discusses the different project types and how to create projects based on them.

This section assumes you've already installed Visual Cafe. Before working with Visual Cafe, you should create a common folder to contain your project folders. You can name this common folder anything you like, such as `My Projects`, and you can place it anywhere you like as long as it's outside the system path. See "About the system path" on page 3-14 and "About the project path" on page 3-15 for more information.

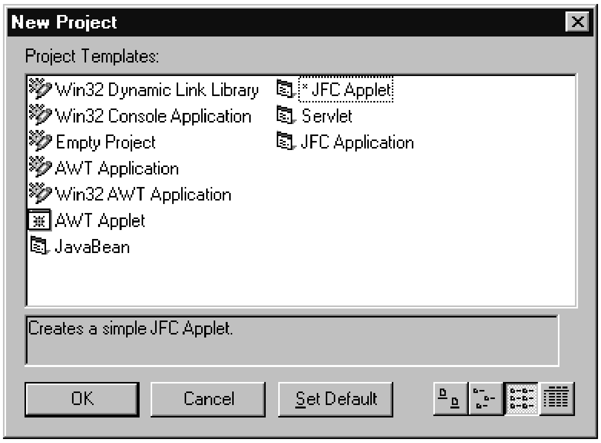
You might also want to create custom templates for the different applets and applications you develop. A **template** is a project that contains files and components you frequently use. For example, if you commonly create a certain kind of applet that resides in an HTML file that's formatted a certain way, you could create this type of generic project and save it as a project template. Then, when you create a new project you can speed your development process by starting with your custom project template. See "Creating a project template" on page 3-28 for more information.

If you don't need to create an entire custom project, but would like to create a custom component, see "Adding a component to a project" on page 3-49.

Using project templates

When you create a new project, you specify a project template as the starting point. Project templates determine those project items that are to be initially added to a project and those configuration options that are to be initially provided. Using project templates reduces the amount of overhead involved in creating projects. You can also create your own project types, as described in “Creating a project template” on page 3-28.

You can choose from a variety of built-in project templates, depending on which Visual Cafe edition you’re using.



Creates a simple JFC Applet.

Note: Some of the project templates shown are not available in the Visual Cafe Standard Edition.

The following project templates are available:

Project type	Description
AWT Applet	A project that contains a single applet and uses the AWT component model. The `Applet` container is a type of panel component that’s designed to appear in an HTML file (such as a Web page) viewed in a Web browser.

Project type	Description
AWT Application	A project that contains a frame (with an Open File dialog and menu bar), an About dialog (with a label and a Done button), and an Exit dialog (with a label and a Yes and a No button). This type of application uses the AWT component model. The Frame container is a special kind of window; this one has been set up as a main application window. In this template, it contains a menu bar and an Open File dialog box, which is a standard Java component. The About and Quit dialog boxes are Visual Cafe components that you can customize like templates; they're tied to the About and Exit menu items.
JFC Applet	A project that contains a JFC/Swing `JApplet` component.
JFC Application	A project that contains a JFC/Swing `JFrame` component, which contains `JPanel`, `JToolbar`, and `JMenubar` components. Several actions are provided to help you get started. The Open menu item opens an Open File dialog box, the Exit menu item opens an Exit dialog box, and the About menu item opens an About dialog box.
JavaBean Wizard	A wizard that helps you create a basic Bean. This template contains the files you need to get started in building and deploying Beans. The JavaBean Wizard creates two files; *beanclass*`BeanInfo.java` and *beanclass*`.java`, where *beanclass* is the name of your Bean. For more information, see "Using the JavaBean Wizard" on page 10-12.
Servlet	A wizard that helps you create a servlet. A servlet works like an applet, but from a server. To run the servlet from Visual Cafe, you can specify various parameters. For more information, see "Specifying execution settings for a servlet" on page 3-33.
Win32 AWT Application	The same as an AWT application, except that the project options are set up to produce a native Win32 executable.
Win32 Console Application	A simple console application that you can use as a starting point. The project options are set up for you. This template has no GUI features, but instead uses a console.
Win32 Dynamic Link Library	A simple DLL you can use as a starting point. The project options are set up for a DLL. This template specifies that the program is a native DLL.
Empty project	A project with no components.

If you're using the Visual Cafe Database Edition, you can choose from some additional project templates. See the *Visual Cafe Database Developer's Guide* for more information.

You can also create your own project templates. See "About files in a project" on page 3-42 for more information.

When you're working with project templates, see the following topics in this section for step-by-step instructions:

- ◆ Setting a new default template
- ◆ Creating a project template
- ◆ Deleting a project template

Setting a new default template

You can select a template to use as the default for all new projects.

To set a default template:

1. From the File menu, choose New Project.

   The current default template is highlighted and an asterisk is placed next to the template name.
2. Select a template.
3. Click Set Default.
4. Click OK.

A new project is created based on the selected template. This template will be used for subsequent new projects until you change the default.

Creating a project template

You use project templates as the basis for new projects. When you create a project from a template, the project initially contains the files and components in the template. Visual Cafe provides a number of built-in templates, as described earlier in this chapter. You can also create your own templates as the starting point for new projects.

Important: When you save a project as a template, make sure all your source files (both `.java` and `.html`) are in the same project folder or a subordinate folder.

To create a project template:

1. Create a project to be used as the template. Add any source code files that you need.
2. With the current project in a Project window, then choose Create Project Template from the Project menu.

   The Create Project Template dialog box appears.
3. Select the group that you want the template to belong to in the Component Library.

   If you do not select a group, the template is added to the Project Templates group.
4. Type a name and description for the template in the appropriate fields.

   The name and description is displayed for the template in the New Project dialog box. The description appears when the template is listed in the Component Library window.
5. Click OK.

Visual Cafe makes a copy of the project and stores it as a project template. The template is now available from the New Project dialog box.

Deleting a project template

The project templates that ship with Visual Cafe cannot be deleted. You can delete a project template that you have created, as long as it is not the current default template. After you delete your custom project template, it no longer appears in the New Project dialog box.

To delete a project template:

1. Open the Component Library.
2. Display the contents of the `Project Templates` folder.
3. Select the template from the template list.
4. Press DELETE.

The project template is deleted from Visual Cafe.

Creating a servlet

Visual Cafe provides a wizard that helps you create a servlet using the Java Servlet API.

It is recommended that you download the Java Servlet Development Kit from `www.javasoft.com` for the most complete information about developing servlets.

To create a servlet with the Servlet project template wizard:

1. From the File menu, choose New Project.
2. Select Servlet and click OK.
3. If you receive the Introduction page, select Don't Show This Page In the Future if you do not want to access this page again. Then click Next.
4. Type a servlet name, and optionally a package and description. Then click Next.

   The servlet name is the name the Java class file will have. The description is the string returned from the `getServletInfo` method. Typically, this is a string containing information about the servlet, such as its author, version, and copyright.

   Tip: After specifying a name, you can click Finish in any page to use the default settings for the servlet.

5. Select the type of servlet you want to create. Then click Next.

   The HTTP servlet extends `javax.servlet.http.HttpServlet`, the Generic servlet extends `javax.servlet.GenericServlet`, and the custom servlet implements the `javax.servlet.Servlet` interface and extends from `java.lang.Object`. The HTTP servlet implements this protocol; if you want to use another protocol, you should create a servlet of one of the other types. The custom servlet type can extend any class you want it to.

   Normally, when extending the functionality of a Web server, you would write an HTTP servlet. If you are providing functionality for another Internet protocol, such as FTP, you would use `GenericServlet`.

You need to directly implement the Servlet interface only if your servlets cannot (or you choose not to) inherit from `GenericServlet` or `HttpServlet`. For example, RMI or CORBA objects that act as servlets will directly implement this interface.

6 Select whether you want the servlet to handle multiple client requests. Then click Next.

   If you select Yes, the service methods of the servlet can be multithreaded. If you select No, the servlet implements the `SingleThreadModel` interface, which will cause a new instance of your servlet to be created to service each new service request; this uses more memory.

   Servlets typically run inside multithreaded servers; servlets must be written to handle multiple service requests simultaneously. It's your responsibility to synchronize access to any shared resources. These resources include in-memory data, such as instance or class variables of the servlet, as well as external components, such as files, databases, and network connections.

   If you do not want your service methods to handle multiple service requests concurrently, you must implement the `SingleThreadModel` interface.

7 If you chose the HTTP or Generic types, select the methods you want to implement. Then click Next.

   You can implement these methods:

   ❖ `init` — Initializes the servlet and logs the initialization in the server log file. You might provide an implementation for this method if you are doing any expensive one-time setup, such as loading configuration data from files or starting helper threads.

   ❖ `destroy` — Destroys the servlet, cleans up resources, and logs the destruction in the servlet log file. You might provide an implementation for this method if you are undoing any initialization work or perhaps synchronizing persistent state with the current in-memory state.

   ❖ `service` — Carries out a request from a client. This method is rarely overridden in HTTP servlets. In an HTTP servlet, standard HTTP requests are supported by dispatching to Java methods specialized to implement them, such as `doGet` and `doPost`.

   You might want to implement the lifecycle methods `init` and `destroy` if you need to manage resources that are held for the lifetime of the servlet. Servlets that do not manage resources do not need to specialize these methods.

The following methods apply only to the HTTP type:

- ❖ `doGet` — Handles the `HTTP GET` operation.
- ❖ `doPost` — Handles the `HTTP POST` operation.
- ❖ `doPut` — Handles the `HTTP PUT` operation, which is like sending a file through FTP.
- ❖ `doDelete` — Handles the `HTTP DELETE` operation, which lets a client request that a URL be removed from the server.

8 If you chose the HTTP types, select whether you want your HTTP request methods to generate HTML pages. Then click Next.

If you select Yes, Visual Cafe generates the code for responding with an HTML page, along with a skeleton HTML page you can modify.

9 If you want to add parameters, click Add, then double-click in a field to make it editable. If you want to generate get/set methods for your initialization parameters, select this option. Then click Next.

Some servers allow an administrator to display and change the value of initialization parameters through get/set parameters. There are two types of parameters you can have your servlet handle:

- ❖ Initialization parameters — These are handled in the `init` method of the servlet. They are specified in server-specific ways (typically in a properties file or perhaps through a tool provided by the server). Typically, they provide some sort of configuration information to the servlet, for example, the name of a file to read or the name of a directory to write to. The `init` method of the servlet retrieves the parameter through the `ServletConfig` interface, which is passed to the `init` method by the server as an argument.
- ❖ Service Request parameters — These are handled in any of the service methods of the servlet. They are name/value parameters passed to the servlet with an individual service request; for example, a `GET` request might pass a series of parameters, such as keywords to search for if the `GET` is handling requests of a search-engine type servlet.

10 In the summary page, check your selections. Click Back to change any information. Click Finish when the page displays what you want.

A new project is created with your servlet class in it.

You can run and debug the servlet from the Visual Cafe environment; see the next section for more information on setting your project options. Visual Cafe uses the `ServletRunner` (a very lightweight Web server) to run the servlet locally, and starts your default Web browser, which will generate the service request that is sent to the servlet through the `ServletRunner`.

Specifying execution settings for a servlet

Visual Cafe lets you run and debug a servlet from its environment. For more information on creating a servlet project, see the previous section.

To specify execution settings for a servlet:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. In the Project Options dialog box, click the Project tab, and choose a project type of Servlet.

4 Specify the options you want:

Field	Description
Start with Web page	Specify an HTML file in one of these ways: Choose Automatic to run from an automatically generated HTML file. This option opens the browser and generates a `GET` request of the servlet by specifying the servlet in the browser's location field. For example, `http://127.0.0.1:8080/servlet/TestServlet` might be added to the browser's location field, which essentially means `GET TestServlet` located at `127.0.0.1:8080/servlet/`, so an `HTTP GET` service request is sent to the servlet. Choose one of your own HTML files from the pop-up menu. HTML files that you added to your project automatically appear in the pop-up menu. Click the browse button (...) for an HTML file. Type in the field to specify a file name or a URL, for example, `http://myserver.someplac.com/somedirectory/some.htm`. This would be a good way of testing posting to a servlet. For example, you might create an HTML page that has a form that the page sends to the servlet (this would be similar to sending the form to a CGI script for processing). Then the servlet's `doPost` method would get called.
Servlet class	The name of the class file.

Field	Description
ServletRunner arguments	The multithreaded `ServletRunner`, intended for testing, can run multiple servlets or one servlet that calls other servlets to handle client requests. It does not automatically reload servlets when they are updated, but there is little overload when stopping and starting the ServletRunner to use a new servlet version. You can specify the following arguments: `-p` *port* — the port number to listen on `-b` *backlog* — the listen backlog `-m` *max* — maximum number of connection handlers `-t` *timeout* — connection timeout in milliseconds `-d` *dir* — servlet directory `-r` *root* — document root directory `-s` *filename* — servlet property file name `-v` — verbose output
Servlet arguments	If your servlet takes initialization arguments, supply them here or in a properties file. The syntax of a single parameter is *parameterName*=*parameterValue*. Parameters are strung together with an ampersand (&) separating each name/value pair. For example, the arguments to a database servlet could look like this: `username=`*fill_in_the_user*&`password=`*fill_in_ the_password*&`owner=`*fill_in_ the_name*

5 Click OK.

The change takes effect the next time that you run your project.

Working with projects

When you're working with projects in Visual Cafe, there are many tasks that you can perform, such as:

- Opening an existing project
- Saving a project
- Renaming a project
- Copying a project
- Deleting a project
- Opening items in a project
- Closing a project
- Opening editors from the Project window

These topics are discussed in the following sections.

If you want to work on an existing project that was created in Visual Cafe version 2.0, you must migrate it to release 3.0 first. See “Migrating a project from earlier versions of Visual Cafe” on page 3-38 for more information.

If you’re working with multiple projects, see “About files in a project” on page 3-42. If you’re working with subprojects, see “About subprojects” on page 3-16.

Creating a new project

In this section you’ll learn how to set up and work with a project. As soon as you launch Visual Cafe, you can begin working on a new project.

To start Visual Cafe:

1. Open the Windows Start menu.
2. Click Programs, then choose Symantec Visual Cafe from the submenu.
3. Select Visual Cafe. The Visual Cafe environment is displayed on your desktop.

To create a new project:

1. From the File menu, choose New Project.

   The New Project dialog box appears.
2. Select the project template you want to use as the basis for your new project.

The default template is indicated by an asterisk. You can easily change the default template by selecting another template and clicking Set Default.

3. Click OK.

   A new Project window opens with the selected template loaded. All objects in the template are added to the project.

Note: Only one project appears in a Project window.

Opening an existing project

You can open an existing project by way of the File menu in Visual Cafe, or a Windows file system window (such as Windows Explorer).

To open an existing project:

1. From the File menu, choose Open.

   The Open dialog box appears.
2. Navigate to the project folder.
3. Make sure Visual Cafe Project is shown in the Files of type field.

   The projects are displayed.
4. Select a project from the list.
5. Click Open.

   The project opens with the last saved window configuration.

To open an existing project from Windows Explorer or another file system window:

◆ Double-click the project file, which has the extension `.vep`.

Using older projects and files

If you've created programs in an older version of Visual Cafe and would like to use them in the current version, you can open the old files in version 3.0 and have Visual Cafe automatically update them.

Migrating a project from earlier versions of Visual Cafe

Visual Cafe version 3.0 project files (the `.vep`, `.vpj`, `.ve2`, and `.cdb` files) are not backward-compatible with Visual Cafe version 2.0 project files. If you save a 2.0 project in the Visual Cafe 3.0 software, you cannot convert the project back to 2.0. Therefore, you might want to copy your 2.0 project directories before opening them in the 3.0 software. That way you will still have your 2.0 project files.

When you open a 2.0 project in Visual Cafe version 3.0, Visual Cafe updates the files to use the new version of the components (specifically, the `INIT` portion of the code). If you add a Visual Cafe version 2.0 Java file without first opening its 2.0 project, the Java file will not be set up to use the new Visual Cafe components. You should first open the 2.0 project in the newer version of Visual Cafe so that the file is updated for you.

Note: If you create an applet using JDK 1.1, the Web browser must support this version of the JDK or the applet will not run in the browser.

Migrating Java source files from JDK 1.0 to JDK 1.1

Visual Cafe offers a utility that automatically converts a Java source file from the JDK 1.0 event model to the JDK 1.1 event model.

To migrate a project file to the JDK 1.1 event model:

1. Open the 1.0-based file in the Source window.
2. From the Tools menu, choose Migrate Event Bindings from 1.0 to 1.1.

   Visual Cafe parses the `handleEvent` and `action` methods. From these methods, new code is generated to handle the events in the JDK 1.1 event model. This includes generating the required listeners and adapters and registering the listeners. For menus with menu items created as quoted literals, such as `menu1.add("Open")`, a `MenuItem` object is created to support the new event model.

   The comment tag `//{{INIT_CONTROLS` is generated by Visual Cafe to mark the location where components are created and initialized. The comment tag `//{{REGISTER_LISTENERS` marks the location where listeners are registered.
3. Look over your code and make modifications as needed. For example, you might have code that you need to move from the `handleEvent` and `action` methods.

Saving a project

The first time you save a project, all the files it contains are saved to the folder you specify. After you save a project once, Visual Cafe automatically saves those files that change during the development of your project.

Note: Saving just the project saves the project files only, not other files such as applets. Choose Save All from the File menu to save all files within a project.

In this section you'll learn how to save a project for the first time, save only the project files, save all the files in a project, and save individual project files. Visual Cafe also lets you customize your backup and save procedures to save files for recovery in case of a system crash and make source-file backups automatic. These environment options are discussed in "Setting backup and save options" on page 3-83.

To save a project for the first time:

1 Activate the Project window, then choose Save from the File menu. The Save dialog box appears.

  The project and all the files it contains should be in the same folder. For easier project management, you should save each project in its own folder.

2 Select a folder and type a name in the Save As field.

  The Visual Cafe project file must have the extension `.vep`. If you type just the first part of the file name, Visual Cafe will add the `.vep` extension for you.

3 Click Save. The new file name appears in the title bar.

To save project files only:

◆ Activate the Project window, then choose Save from the File menu.

  Only the project file is saved, not related files such as applets.

To save all the files in a project:

◆ Activate the Project window, then choose Save All from the File menu.

  All files in the project are saved.

After you save a project once, you can save a single file in the project separately from the project's other files.

To save one file in a project:

1 Open the file in an editor and activate that window.

Remember that a top-level container in the Objects view is associated with a Java source file.

2 From the File menu, choose Save.

The Save menu item is enabled only if there are changes to save.

Renaming a project

To rename a project, simply save it with a different name.

To save and rename a project:

1 Activate the Project window, then choose Save As from the File menu.

The Save As dialog box appears.

2 Type the project name in the File name field, then click Save.

The Visual Cafe project file must have the extension `.vep`. If you type just the first part of the file name, Visual Cafe will add the `.vep` extension for you.

3 Using Windows tools, delete from the project folder the project files that have the old name and the extensions `.vep`, `.vpj`, `.ve2`, and `.cdb` (if present).

Copying a project

If you want to reuse a particular project, you can copy the project file and other related files to a new location.

To copy a project:

1 If Visual Cafe is running, make sure that the Project window for the project is closed.

Note: You can't copy files while they are in use.

2 Do any of the following:

- ❖ In Windows, copy the files in the project and paste them in a new folder, preserving the folder structure.
- ❖ Drag the files and drop them where you want to copy them.
- ❖ If all of your files are in one project folder, you can simply copy the project folder, then rename it.

Deleting a project

If you no longer have a need for a particular project, you can delete it.

To delete a project:

1 If Visual Cafe is running, make sure that the Project window for the project is closed.

Note: You can't delete files while they are in use.

2 Do any of the following:

- ❖ From Windows, delete the project folder and all files in the project (except imports).
- ❖ Click on the files you want to delete. The selected files will be highlighted.
- ❖ Drag the files to the Recycle Bin, or press the DELETE key.

Opening items in a project

To open an item in a project and access its contents, double-click the item in the Project window.

If the file for a project item is a text file, it's displayed in an editor window. If it's a project file, the project is opened by Visual Cafe and its Project window is displayed. If the item is a `.class` file, the source file displays.

For all project items, double-clicking is a shortcut for selecting the item and choosing Open filename from the File menu.

Closing a project

When you close a project, all windows associated with the project close (except the Property List, which clears).

To close the active project:

◆ With the Project window active, choose Close Project from the File menu.

  You're prompted to save any unsaved changes. Visual Cafe remains open so you can work on other projects.

To close all projects and exit Visual Cafe:

◆ Choose Exit from the File menu.

  You're prompted to save any unsaved changes. The Project and Visual Cafe windows close.

About files in a project

The Java source files (`.java`) and compiled Java files (`.class`) have the same file names, but different extensions. You see only the `.java` files in the Project window, because they're used for development, while `.class` files are used for deployment.

Note: Visual Cafe lets you package `.class` files into `.zip` or `.jar` archives to speed up downloading and conserve disk space.

For more information about files in projects, see "About the contents of a project" on page 3-11. Also see the following section "Using files in a project" and "Working with components in a project" on page 3-48.

Using files in a project

The Files view lists all the files your project contains, except for the project management files (`.vep`, `.vpj`, `.ve2`, and `.cdb` files). At the top level are the files for the components in your project, as well as any HTML files you've added. (For more information about the Files view, see "The Files view" on page 3-5.)

In the Imports folder are all the Java source files for the packages your project uses. Whereas in the Packages view the files are listed by package (see "The Packages view" on page 3-6), in the Imports folder the files are listed alphabetically.

You can sort in the Files view by clicking a heading button, and you can also turn RAD (and the resulting automatic code generation) on and off for a file. For more information, see "Enabling and disabling RAD and automatic code generation" on page 4-45.

HTML files are also listed in the Objects view (see "Viewing the components and HTML files in a project" on page 3-48).

Note: If a file with the extension `.java` appears, the corresponding `.class` file is not listed, because you're interested only in `.java` files during development.

When working with individual files, see these related tasks in this section for more information:

- Adding a new file to a project
- Adding an existing file to a project
- Deleting a file from a project
- Copying a file in a project
- Sharing files among projects

Adding a new file to a project

Visual Cafe lets you create a new text file from within its environment. You can add HTML code to create HTML files and Java code to create Java source code files; you can then add these files to your project.

To add a new file to a project:

1. From the File menu, choose New, then choose File from the submenu.

   An empty Source window opens.

2 From the File menu, choose Save As.

3 In the Save As dialog box, type the file name (including the appropriate extension) and select the Add to Project option.

4 (Optional) Select or deselect Enable RAD.

5 Click Save.

The file name appears in the Source window's title bar.

6 In the Source window, enter the appropriate code.

The new file is added to the project.

Adding an existing file to a project

You can add `.html`, `.java`, `.class`, and `.vep` project files to a project, as well as any other type of file. If you have the Visual Cafe Professional Edition or Visual Cafe Database Edition, you can add DLLs as well. You can add ZIP files and DLLs to a project and, from your Java code, import any packages or class files it contains.

Note: Packages in a ZIP file or DLL do not appear in the Packages view of the Project window.

The only way an HTML file will appear in the Project window is if you manually add it. You can add HTML files to a project as a helpful organizational tool, but its not required. Adding HTML files to your project allows you to see how your Java applet works.

Although Visual Cafe automatically creates many Java source files for you in its visual environment, in some cases you may want to add Java files directly. For example, if you wanted to import a Java source file for an applet that you created in a product other than Visual Cafe, you could place the file in a project folder, then add the `.java` file to the Project window. Visual Cafe will attempt to translate the file into its visual environment.

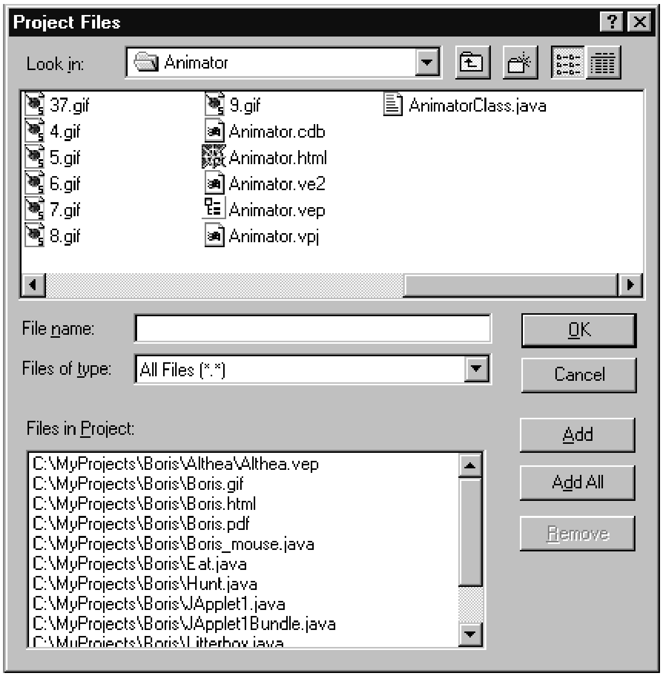
If you add the project (`.vep`) file of a project to the current, open project, the added project becomes a subproject of the open project it resides in.

Remember that you should store all the files of a given project in a single folder dedicated to that project. Doing so will make deploying your applets, applications, and Web pages much easier. See "Working with components in a project" on page 3-48 for more information.

To add files while in the Visual Cafe environment:

1 From the Insert menu, choose Files into Project.

The Project Files dialog box appears:



```
C:\MyProjects\Boris\Althea\Althea.vep
C:\MyProjects\Boris\Boris.gif
C:\MyProjects\Boris\Boris.html
C:\MyProjects\Boris\Boris.pdf
C:\MyProjects\Boris\Boris_mouse.java
C:\MyProjects\Boris\Eat.java
C:\MyProjects\Boris\Hunt.java
C:\MyProjects\Boris\JApplet1.java
C:\MyProjects\Boris\JApplet1Bundle.java
C:\MyProjects\Boris\Litterbox.java
```

2 Choose either Net Files (`*.java`, `*.html`, `*.htm`, `*.gif`, `*.jpg`, `*.jpeg`, `*.au`, `*.prop`) or All Files (`*.*`) from the Files of Type drop-down list.

If Net Files is chosen, only files in the project that have the specified extensions are displayed in the upper scrolling list. If All Files is chosen, all files are listed.

3 Navigate to the appropriate folder and either double-click the name of the file in the upper scrolling list or select the name and click Add.

The name of the file is added to the lower scrolling list.

4 Repeat step 3 for each additional file you want to add to the project.

5 Click Add or Add All, as appropriate.

6 Click OK.

The file or files appear in the Project window.

To add files from Windows Explorer or another file system:

- ◆ Drag Java or HTML files from Windows Explorer into the Files view of the Project window.

To add files from the Find in Files dialog box:

1. From the Search menu, choose Find in Files.
2. Select a folder.
3. Click Find.
4. Right-click on the window with the list of files. Then choose Add All to Project.

   The selected files are added to your active project. You can easily add a large subdirectory to your project from the Find in Files dialog box.

Deleting a file from a project

As you're developing your project, you can delete HTML and Java source files as needed.

To remove a file with the DELETE key:

- ◆ Select a file in the Project window's Files view, and press DELETE.

Alternatively, you can use the Project Files dialog box.

To remove files by using the Project Files dialog box:

1. Make the Project window active.
2. While any view (Files, Objects, or Packages) is displayed, choose Files into Project from the Insert menu.

   or

   In the Files view, right-click the object window to display the pop-up menu, then choose Insert Files.
3. In the Project Files dialog box, select one or more files from the list in the bottom pane.
4. Click Remove.
5. Click OK.

Note: Deleting a file from a project does not delete it from the project folder nor your hard drive. You must manually delete it.

When you remove a Java source file from a project, all associated visual elements that are created in the Java code are also removed.

Note: You can't remove imported files; they are an implicit reference to source files in use by your project.

Copying a file in a project

You can copy and paste a file that's associated with an object from one project to another, or within the same project. The file is duplicated and placed in the target project folder, and the associated project files are updated to reflect the change.

If you want to copy another type of file, you need to do so from a file system window, such as Windows Explorer.

To copy a file:

1. Open the project(s) you want to use and click the Objects tab in the Project window.
2. In the Project window, select the file you want to copy.
3. From the Edit menu, choose Copy (or click the toolbar's Copy button).
4. Activate the Project window that you're moving the file to.
5. From the Edit menu, choose Paste (or click the toolbar's Paste button).

The file appears in the Project window. If you're pasting into the same project you copied the file from, the file is renamed to prevent file name conflicts.

Sharing files among projects

You can reuse a file if it doesn't include code generated by Visual Cafe. Just add the file to another project. For more information, see "Adding an existing file to a project" on page 4-35.

Any file that contains code that was automatically generated by Visual Cafe should not be shared among multiple projects. The file can change in one project, causing version problems in any other projects it belongs to. Instead, you have a number of options. For example, you can copy the file into the new project folder and add it to the project, create a new project template tailored to your requirements, add your own custom components to the Component Library, cut and paste Java code, and so on. For more information, see "Copying a file in a project" on page 4-39 and "Creating a project template" on page 3-86.

Note: File names are relative to the current project. For example, if you add a file called `foo.java` and it resides in a subfolder (called `foo1`) of the project folder, the file name is stored as `foo1\foo.java` and is not fully qualified. If `foo1` is a folder at the same level as the project folder, the file will be stored as `..\foo1\foo.java.`

Working with components in a project

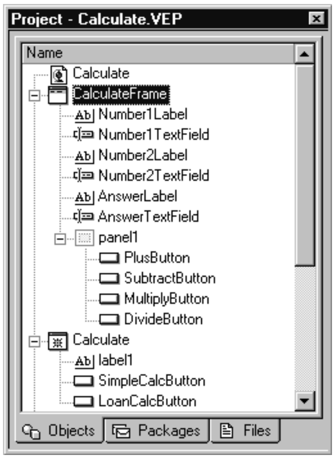
Chapter 7 provides detailed information on working with components. In this section you'll learn the basics of adding a component to a project. This section explains how to perform the following tasks:

- Viewing the components and HTML files in a project
- Adding a component to a project

For more information about adding, copying, renaming, and deleting components, as well as additional component operations, see "Working with forms and components" on page 7-26.

Viewing the components and HTML files in a project

The Objects view of the Project window shows the components in your project, as well as any HTML files you've added. The components in a container appear subordinate to the container, like a file system display. The containers at the top level are separate Java files in your project, while the components in the containers are Java code within the container Java file.



Adding a component to a project

Visual Cafe provides several ways to add components to your project. When you add components directly to a form in the Form Designer, they're also added to the associated project.

You can add a component by copying it from another location. For more information, see "Copying components" on page 7-29.

To add a component by using an Insert menu item:

◆ Do either of the following:

  ❖ While the Project window or Form Designer is active, choose an item (such as a Bean or another type of component) from the Insert menu to add it to your project.

  or

  ❖ Right-click the Project window and choose an item from the pop-up menu.

  The component is added to the container that's currently selected in the Project window.

Note: If the component doesn't get added correctly, you'll be able to see a file in the Packages and Files views of the Project window, but you won't see an object in the Objects view. You'll probably need to correct the Java code to get the file to parse. For more information, see Chapter 4, "Working with Source Code."

About HTML files in Visual Cafe

Applets run within another program, usually a Web browser. You add an applet tag to HTML code in a Web page to add that applet to the page. An **applet tag** is HTML code that causes an applet to appear in a Web page. For more information, see "How HTML and Java work together: the applet tag" on page 3-51.

Visual Cafe's AppletViewer lets you run and debug applets within the Visual Cafe environment, without supplying an HTML file. You can also run and debug your applet in a Web browser, which lets you view the HTML page and the applet at the same time. If you want to test your applet within an HTML page, you can select an HTML file in which to run the applet.

You can add HTML files to a project as a useful organizational tool, but it's not required. Once you add an HTML file to your project, you can view it in the Objects and Files views, open it quickly from the Project window, and see it in the project options. For more information, see "Specifying an applet's HTML file" on page 5-3.

If you've installed Visual Page and the software has been set up properly, you can double-click an HTML file in the Objects view to launch Visual Page and display the HTML file. You can also view and edit an HTML file directly in Visual Cafe's Source window; for example, you can double-click the HTML files in the Project window's Files view.

To add an applet to an HTML page, you can either manually add an applet tag to the file or drag the applet from the Visual Cafe Project window to an HTML page displayed in Visual Page.

How HTML and Java work together: the applet tag

Applets are designed to be embedded in Web pages. Basic operations such as starting, stopping, and displaying the applet are all handled by the Web browser. To tell the Web browser to display the applet, you have to put certain information about the applet in the HTML file.

When a Java-capable browser encounters an applet tag, it reserves onscreen space for the applet, loads the Applet subclass onto the computer on which the browser is running, and creates an instance of the Applet subclass. Next, the browser initializes the applet, and the applet is off and running.

For the Web browser to be able to display an applet, it requires some basic information, which is provided by applet tags. Within the applet tag you specify where to find the `.class` file and how large to make the display space in the Web page. Like most HTML tags, the applet tag has an opening tag, `<APPLET>`, and a closing tag, `</APPLET>`. The applet tag is a link to a class file that contains bytecode. The Web browser interprets the bytecode and displays the applet in the Web page.

In addition, there are three required attributes for the applet tag: `CODE`, `WIDTH`, and `HEIGHT`.

The following listing shows an applet tag that includes the applet by the name of `HelloWorldApplet` in an HTML page:

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>
Here is the output of my program:
<APPLET CODE="HelloWorldApplet.class" WIDTH=150
 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

`<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>` instructs the browser to load the applet whose compiled code is in the file named `HelloWorldApplet.class.` The browser looks for this file in the same directory as the HTML document that contains the tag.

When the browser finds the class file of the applet, it loads, creates, and displays an instance of the class. If you include an applet tag twice in one HTML page, the browser loads the class file once and creates and displays two instances of the class.

The `WIDTH` and `HEIGHT` attributes work the same way as they do with an `<IMG>` tag: they specify the size of the applet's display area in pixels. Most browsers do not let the applet resize itself to be larger or smaller than this display area.

To specify a JAR file in an HTML file, add the variable `ARCHIVE="`*name*`.jar"` to the applet tag. You can specify multiple JAR files by delimiting them with a comma. You can use the `CODEBASE` variable to specify the location of class files and the `ALT` variable to specify text that will appear if a Web browser understands applet tags but can't display an applet.

To ensure that the required Symantec custom classes are available to your applets, you need to supply them on your Web site. For instructions on deployment, see "Deploying your applet" on page 5-32.

Adding an applet to an HTML page

You can add an applet to an HTML page by adding an applet tag to the HTML code, or by dragging the applet from the Project window into an HTML page displayed in Visual Page.

An applet tag is HTML code that causes an applet to appear in a Web page. It has the following basic format:

```
<APPLET code="applet.class" width=pixw height=pixh></APPLET>
```

applet is the name of the applet.

pixw is the number of pixels for the onscreen width.

pixh is the number of pixels for the onscreen height.

If you drag an applet to an HTML page displayed in Visual Page, Visual Page adds the applet tag for you.

Here are some additional parameters you might find useful:

Attribute	Description
`codebase=`*codebaseURL*	This optional attribute specifies the directory that contains the applet class file(s). The default is the location of the HTML file.
`archive=`*archiveList*	This optional attribute describes one or more archives, delimited by a comma, that contain classes and other resources that will be preloaded. The classes are loaded with the codebase, if specified. You can specify archives that are JAR files.
`alt=`*alternateText*	This optional attribute specifies any text that should be displayed if the browser understands the `APPLET` tag but cannot run Java applets.

See also "Specifying an applet's HTML file" on page 5-3. Consult an HTML book for more information on the applet tag.

Using HTML files

When working with HTML files, you can do the following:

- View an edit the files
- Add an applet to an HTML page
- Pass parameters to applets

These tasks are discussed in the following sections.

Viewing and editing HTML files

If HTML files are included in your project, you can view and edit them from within Visual Cafe or Visual Page.

Visual Cafe can generate all the HTML files you need to deploy your applets. This file is called `autogen_`*projectname*`.html`, and should not be edited because it could be regenerated by Visual Cafe. If you do add changes to this file, they will be lost when it's regenerated.

To view or edit an HTML file:

1 Open the file by using one of these methods:

- ❖ In the Project window (Objects or Files view), double-click the HTML file name. In order for an HTML file to appear in the Objects or Files view, it must be inserted into the project.
- ❖ In the Objects view of the Project window, right-click and choose Edit Source.
- ❖ Choose Open from the File menu, then select the file and click Open.

The HTML file appears in the Source window. If you've installed Visual Page, opening a file from the Objects view opens the file in Visual Page; opening a file from the Files view opens it in the Source window.

2 View the file and edit it in Visual Cafe's Source window or in Visual Page.

3 Save the file by choosing Save from the File menu.

Passing parameters to applets from an HTML file

It's often helpful to have an applet receive information from an HTML document. This lets you customize how applets appear in Web pages. To pass information from an HTML document to a Java applet, use the `<PARAM>` tag.

The `<PARAM>` tag can appear between the HTML `<APPLET>` and `</APPLET>` tags. The `<PARAM>` tag has two attributes, `NAME` and `VALUE` tags, which are used to pass data to a Java applet:

```
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=100>
<PARAM> NAME="Color" VALUE="red">
<PARAM> NAME="Number" VALUE="81">
</APPLET>
```

The HTML file can pass multiple parameters. After the parameters are set in the HTML file, they can be retrieved by an applet with the `getParameter` method:

```
String x=getParameter("Color");
```

In the example above, variable *`x`* is declared as type `String` to hold the `Color` parameter retrieved from the HTML document. You must specify the name of the parameter to retrieve from the HTML file.

The parameter is named `Color`:

```
<PARAM> NAME="Color" VALUE="red">
```

The parameter name is specified as the argument passed to `getParameter` in the applet code:

```
getParameter("Color");
```

This results in the `getParameter` method returning the value `red` from the HTML file.

Customizing a project

You can change a variety of settings that apply to an individual project by going to the Project menu and choosing Options. This opens the Project Options dialog box, where you can set characteristics that apply to a particular project, such as project release type, run-time arguments, compiler settings, and search folders.

If you'd like to set options that apply to the Visual Cafe environment itself, and thus all projects, see the following section, "Customizing the Visual Cafe environment."

About project options

Project options are project-specific, unlike the preferences you set for the Visual Cafe environment as a whole using the Environment Options dialog box. Project Options settings remain bound to a project even when you close and later reopen the project. They control general project behavior, including build and run settings.

Note: Most changes made in the Project Options dialog box do not take effect until the next time they are needed. For example, if you change run-time arguments, they do not take effect until the next time you run your Java application.

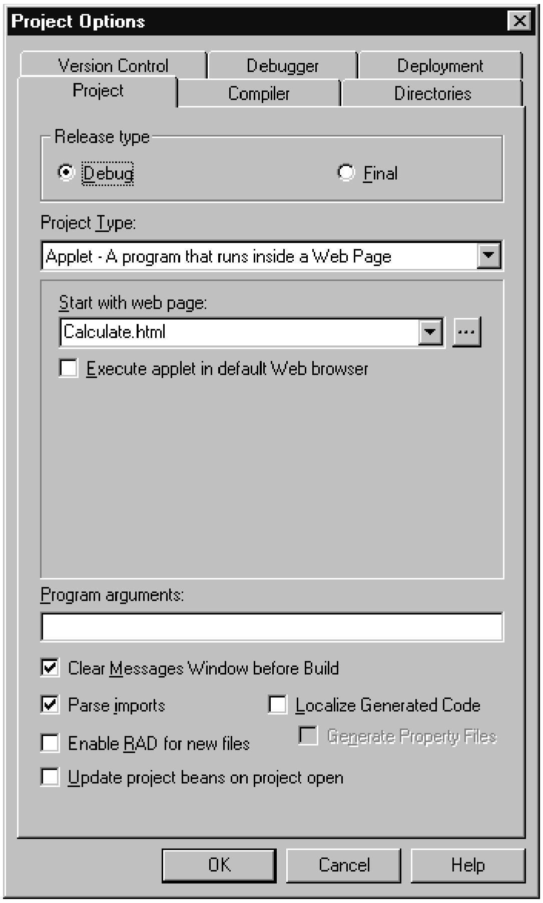
To set options that apply to a single project:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

The Project Options dialog box is organized into categories; each category's options are on a page. Click the Project, Compiler, Directories, Version Control, Debugger, or Deployment tab to go to that category of project options.



Release type

[✓ Debug] Final

Project Type: Applet - A program that runs inside a Web Page

Start with web page: Calculate.html

☐ Execute applet in default Web browser

Program arguments:

☑ Clear Messages Window before Build

☑ Parse imports

☐ Localize Generated Code

☐ Enable RAD for new files

☐ Generate Property Files

☐ Update project beans on project open

By default, when the Project Options dialog box appears it displays the Project page. By clicking different tags you can move freely between options pages as you configure options. You can move around the options on each page by using the TAB key. You can select another options tab by using the Right and Left Arrow keys.

Each options page contains Cancel and OK buttons. If you click Cancel, the options you just applied will not be set. Clicking OK saves the options settings for the current editing session; these options settings are then available for selection — for example, in another editing session or from the Project window.

Other elements that are found on all options pages include the Help button and the Options pop-up menu.

Project options can be set for final and debug projects and subprojects. The page shows the settings that are common among the selected projects.

If the settings differ among the selected projects, the settings indicate that state.

Many of the basic project options are described in this section. Others are described elsewhere in this manual, where project-related procedures are discussed. This table shows where you can read about additional project options:

Topic name	**Page**
"Specifying execution settings for a servlet"	page 3-33
"About files in a project"	page 3-42
"Working with components in a project"	page 3-48
"Specifying source-file search paths for a project"	page 3-62
"Specifying an applet's HTML file"	page 5-3
"Specifying an applet's HTML file"	page 5-3
"Specifying the main class to run for an application"	page 5-5
"Specifying arguments for application execution"	page 5-6
"Specifying whether builds are debug or final"	page 5-16
"Specifying whether to parse imports"	page 5-18
"Specifying the output folder for a project"	page 5-19
"Specifying whether to clear messages before a build"	page 5-19
"Setting deployment options for a project"	page 5-39
"Setting compiler options"	page 5-57
"Setting exceptions"	page 6-38

Topic name	Page
"Debugging applets in a Web browser"	page 6-42
"Setting project options for native programs"	page 11-7
"Setting version control options"	page 12-7

Setting project options

In this section you'll learn how to set some common project options in the Project Options dialog box. The following topics are discussed:

- Setting the program type
- Specifying class-file search paths for a project
- Specifying source-file search paths for a project
- Setting the class path

Setting the program type

If you started developing a project based on the empty project template, you can still tell Visual Cafe what kind program type you want it to be.

To set the program type:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Project tab (if it's not already active).

4 Select one of the following options:

Option	**Description**
Applet	The project is an applet. When you run a project, the Start with Web page setting determines which HTML file is used. The HTML file determines which applets are run. See “Specifying an applet’s HTML file” on page 5-3. To specify that your applets should run in your default Web browser, select the Execute applet in default Web browser option. Deselect it if you want to run applets in the Applet Viewer associated with the Visual Cafe environment. For more information, see “Running a project” on page 5-1.
Application	The project is a stand-alone application. To run the application from Visual Cafe, the main class must also be specified, as well as arguments that must be passed to it. See “Specifying the main class to run for an application” on page 5-5.
Servlet	The project contains a servlet, most likely created using the Servlet Wizard (a project template). To run the servlet from Visual Cafe, you can specify various parameters. For more information, see “Specifying execution settings for a servlet” on page 3-33.
Win32 Application (Professional and Database editions)	The project is a native, stand-alone executable. To run the application from Visual Cafe, the main class must also be specified. See “Specifying the main class to run for an application” on page 5-5. You can also set the application name, which is by default the project name appended with the `.exe` extension. See “Specifying the name of a native application or DLL” on page 11-7. If you want to run your executable from a different directory than where it is located, see “Specifying the working directories for a native program” on page 11-9.

Option	Description
Win32 DLL (Professional and Database editions)	The project is a native Dynamic Link Library (DLL). You can choose the program to use to run and debug the DLL. See “Specifying a program for running and debugging a DLL” on page 11-9. You can also set the library name, which is by default the project name appended with the appropriate extension. See “Specifying the name of a native application or DLL” on page 11-7. If you want to run your executable from a different directory than where it is located, see “Specifying the working directories for a native program” on page 11-9.

5 Click OK.

The changes takes effect next time you run your project.

Specifying class-file search paths for a project

You can tell Visual Cafe where to look for the class files used by a project — for example, where to find the packages you’re using. You can do so by setting the class path for an individual project, rather than setting it for the whole Visual Cafe environment. Here is how you can do this, in the order in which they are searched:

- ◆ Create your own custom list of directories
- ◆ Set Visual Cafe to automatically generate the class path based on the `.java` files you have added to your project (including those created when you add top-level components)
- ◆ Use the class path that is set for the Visual Cafe environment

For more information, see “Setting environment variables in the sc.ini file” on page 3-72 and “Setting internal VM environment options” on page 5-28.

To specify class search paths for a project:

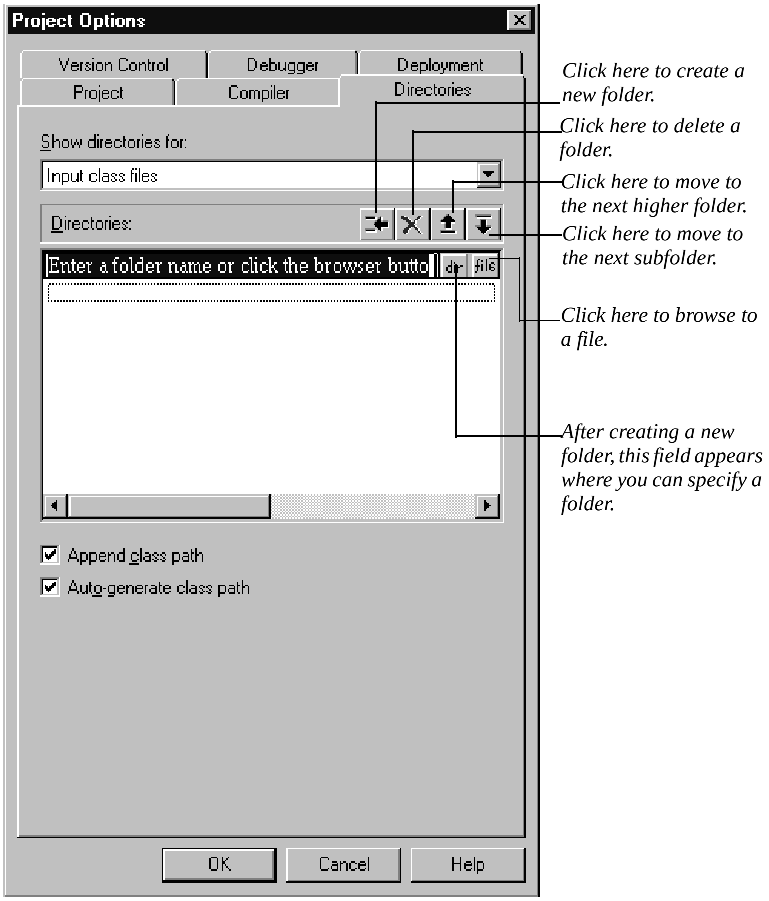
1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Directories tab.

4 In the Show directories for list, choose Input class files.


Project Options
Version Control
Debugger
Deployment
Project
Compiler
Directories
Show directories for:
Input class files
Directories:
Enter a folder name or click the browser butto
dir
file
Append class path
Auto-generate class path
OK
Cancel
Help
Click here to create a new folder.
Click here to delete a folder.
Click here to move to the next higher folder.
Click here to move to the next subfolder.
Click here to browse to a file.
After creating a new folder, this field appears where you can specify a folder.

A list of folders (directories) displays. The directory order affects the search order; the topmost directory is the first to be searched.

5 Modify the list as needed:

- To change the order in which directory are searched, select a folder and move it with the Up Arrow and Down Arrow buttons.
- To delete a directory from the list, select the folder and click the Delete button.
- To add a directory to the list, select the blank entry (marked by an empty box) at the bottom of the list and type the directory name, including the full path. Or click the New button (located

above the text box), then select a directory by clicking the Browse (...) button that displays in the field. You can also use the New button to insert a new entry above the selected entry.

6. To generate the class path based on the files in the project, select Auto-generate class path. Otherwise, deselect it.

   By default, this option is selected. When this option is selected, if a file is not in the project folder, the file path is added to the class path.

7. To append the Visual Cafe environment class path to the project class path, select Append class path. If you deselect it, the class path defined for the Visual Cafe environment is not used.

   By default, this option is selected.

8. Click OK.

   The changes take effect immediately.

Specifying source-file search paths for a project

The source search path applies to Java source files and any text file that can be opened in the Source window.

Note: You can set the source search path for the entire Visual Cafe environment from the Environment Options dialog box and with the `javainc` statement in the Visual Cafe `\Bin\sc.ini` file. For more information, see “Setting environment variables in the sc.ini file” on page 3-72 and “Setting internal VM environment options” on page 5-28.

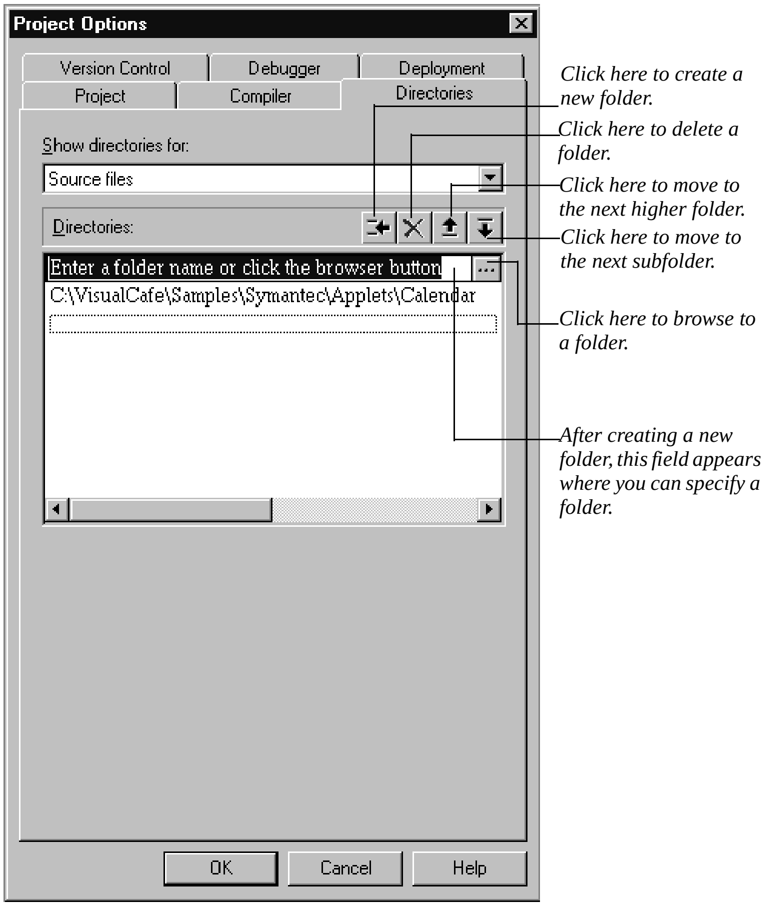
To specify source file search paths:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.

3. Click the Directories tab.

4 In the field under Show directories for, choose Source files.


Project Options
Version Control
Debugger
Deployment
Project
Compiler
Directories
Show directories for:
Source files
Directories:
Enter a folder name or click the browser button
C:\VisualCafe\Samples\Symantec\Applets\Calendar
OK
Cancel
Help
Click here to create a new folder.
Click here to delete a folder.
Click here to move to the next higher folder.
Click here to move to the next subfolder.
Click here to browse to a folder.
After creating a new folder, this field appears where you can specify a folder.

You can now add directories (folders) that can contain source files. The directory order affects the search order: the topmost directory is the first to be searched. The default setting, the project folder, does not appear in the list; it is the first directory in the search order.

5 Modify the list as needed:

- To change the order in which directories are searched, select a directory and move it with the Up Arrow and Down Arrow buttons.
- To delete a directory from the list, select the directory and click the Delete button.

- ❖ To add a directory to the list, select the blank entry (marked by an empty box) at the bottom of the list and type the directory name, including the full path. Or click the New button (located above the text box), then select a directory by clicking the … Browse button (...) that displays in the field. You can also use the New button to insert a new entry above the selected entry.

6 Click OK.

The changes takes effect immediately.

Customizing the Visual Cafe environment

You can customize the Visual Cafe environment in a variety of ways by going to the Tools menu and choosing Environment Options. This opens the Environment Options dialog box. Here you can set characteristics that apply to the Visual Cafe environment as a whole, not just to single projects as in the Project Options dialog box that was described in the previous section. Environment options include setting the Visual Cafe startup mode, defining the Help file set, debugging tasks, text formatting, and code editing.

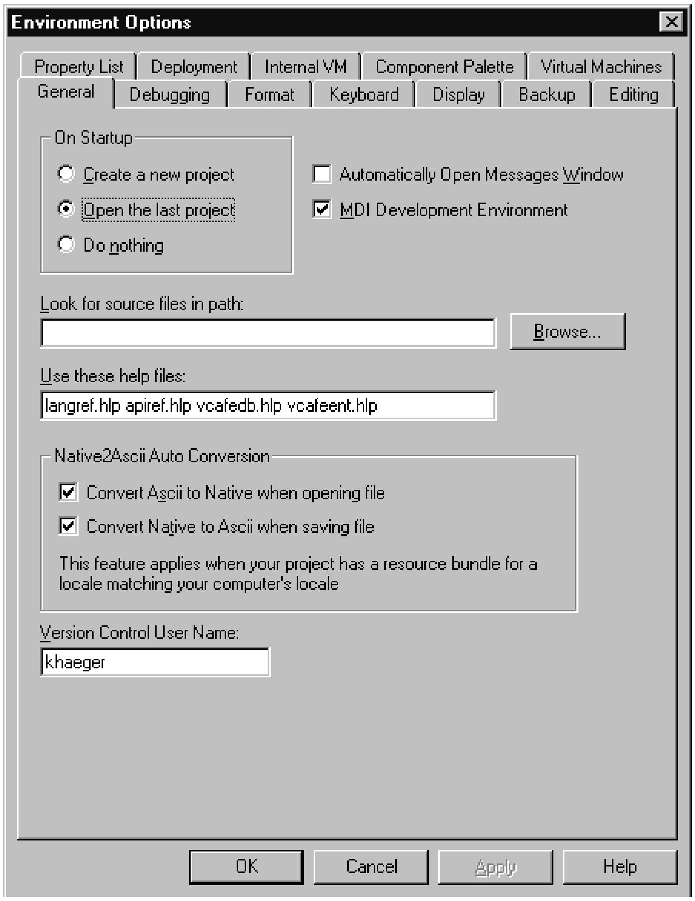
About environment options

Environment options affect the entire Visual Cafe environmen: they apply to every Visual Cafe project. The options you set stay in effect until you return to the Environment Options dialog box and reset them.

To set options that apply to all projects:

◆ From the Tools menu, choose Environment Options.

The Environment Options dialog box appears.


Environment Options

On Startup
- ( ) Create a new project
- (●) Open the last project
- ( ) Do nothing

- [ ] Automatically Open Messages Window
- [✓] MDI Development Environment

Look for source files in path:

Use these help files: langref.hlp apiref.hlp vcafedb.hlp vcafeent.hlp

Native2Ascii Auto Conversion
- [✓] Convert Ascii to Native when opening file
- [✓] Convert Native to Ascii when saving file

This feature applies when your project has a resource bundle for a locale matching your computer's locale

Version Control User Name: khaeger

Most changes made in the Environment Options dialog box take effect immediately. If this isn't the case, a message will appear as a reminder to restart Windows in order for the changes to take effect.

The Environment Options dialog box is organized into categories of options, much like the Project Options dialog box. Click the General, Debugging, Format, Keyboard, Display, Backup, Editing, Property List, Deployment, Internal VM, Component Palette, or Virtual Machines tabs to go to that category of environment options.

By default, when the Environment Options dialog box appears, it displays the General page. Clicking another tab selects the corresponding page. You can move freely between options pages as you configure options. You can move around the options on each page by using the TAB key. You can select another options tab by using the Right and Left Arrow keys.

Each options page contains Cancel and OK buttons. If you click Cancel, the options you just applied will not be set. Clicking OK saves the options settings for the current editing session; these options settings are then available for selection — for example, in another editing session or from the Project window.

Other common elements on all options pages include the Help button and the Apply button. Click Apply to try out settings, and then if you don't like the result, you can click Cancel and no permanent changes have been made.

Many of the basic environment options are described in this section. Others are described elsewhere in this manual, where environment-related procedures are discussed. This table shows where you can read about additional environment options:

Topic name	Page
"Working with the MDI window system"	page 3-19
"Configuring the Class Browser"	page 4-6
"Showing horizontal scroll bars"	page 4-42
"Controlling the cursor style"	page 4-43
"Changing code spacing and text capitalization"	page 4-44
"Setting text formatting for a single file"	page 4-53
"Specifying custom keyword formatting"	page 4-58
"Using different Java virtual machines in Visual Cafe"	page 5-22
"Setting internal VM environment options"	page 5-28
"Setting deployment options"	page 5-38
"Setting deployment options for all projects"	page 5-52
"Switching to the Debug workspace when running in the debugger"	page 6-12
"Enabling or disabling ValueTips at debug time"	page 6-28
"Using incremental debugging"	page 6-40

Topic name	**Page**
"Customizing the Component Palette"	page 7-13

Setting environment options

In this section you'll learn how to set some common environment options in the Environment Options dialog box. The following topics are discussed:

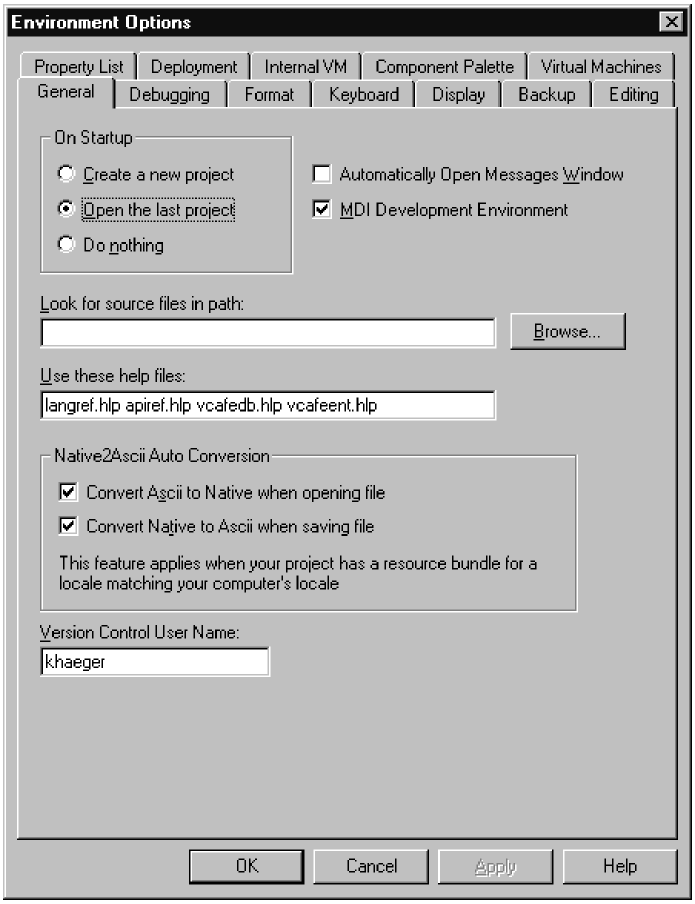
- Defining the Visual Cafe startup mode
- Setting the scope of the Undo command
- Defining the Help file set
- Specifying source-file search paths for Visual Cafe
- Inheriting the class path from the Windows environment
- Setting the class path for a Web browser
- Setting environment variables in the sc.ini file
- Mapping Visual Cafe commands to key sequences
- Customizing the display font and color

Defining the Visual Cafe startup mode

The startup mode defines what happens when you start a new Visual Cafe session.

To establish the startup mode:

1 From the Tools menu, choose Environment Options, then click the General tab.



On Startup
- ( ) Create a new project
- (●) Open the last project
- ( ) Do nothing

- [ ] Automatically Open Messages Window
- [x] MDI Development Environment

Look for source files in path:

Use these help files: langref.hlp apiref.hlp vcafedb.hlp vcafeent.hlp

Native2Ascii Auto Conversion
- [x] Convert Ascii to Native when opening file
- [x] Convert Native to Ascii when saving file

This feature applies when your project has a resource bundle for a locale matching your computer's locale

Version Control User Name: khaeger

2 In the On Startup area, select the appropriate option.

Option	Description
Create a new project	Open a new project each time Visual Cafe starts.
Open the last project	Open the project that was active the last time you exited Visual Cafe; if there is none, a new project is created. (This is the default)

Option	Description
Do nothing	Specify that no project opens and that you will select an appropriate action after Visual Cafe starts.

Setting the scope of the Undo command

The Undo command is available from the Edit menu and has standard Windows functionality. The default number of undoable actions is 300.

To set the scope of the Undo command:

1 From the Tools menu, choose Environment Options, then click the Backup tab.

2 In the Save actions for undo field, enter the number of previous actions that you want Visual Cafe to store for each window.

The number of undoable actions is set.

Defining the Help file set

The Help file set is a group of WinHelp `.hlp` and `.cnt` files that Visual Cafe uses at run time. They are available when you use F1 help from the Source window or Source pane.

To define the Help file set:

1 From the Tools menu, choose Environment Options, then click the General tab.

2 In the Help Files field, enter the `.hlp` file names.

Note: Certain folders are automatically searched by Visual Cafe: the installation folder and the release `\bin` and `\help` folders.

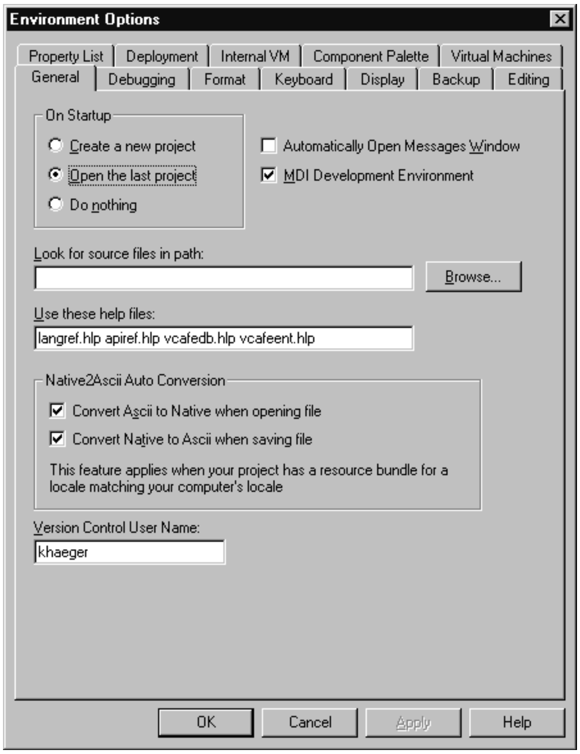
Specifying source-file search paths for Visual Cafe

You can specify the path to locate source files if they are not in the project folder. This setting applies to the entire Visual Cafe environment.

Note: You can set the source-file search path for a project from the Project Options dialog box, and for the entire environment with the `javainc` statement in the `\VisualCafe\Bin\sc.ini` file. You can also set environment variables in the Environment Options dialog box. For more information, see "Setting internal VM environment options" on page 5-28.

To specify the source-file path:

1 From the Tools menu, choose Environment Options, then click the General tab.



On Startup
- ( ) Create a new project
- (●) Open the last project
- ( ) Do nothing

- [ ] Automatically Open Messages Window
- [✓] MDI Development Environment

Look for source files in path:

Use these help files: langref.hlp apiref.hlp vcafedb.hlp vcafeent.hlp

Native2Ascii Auto Conversion
- [✓] Convert Ascii to Native when opening file
- [✓] Convert Native to Ascii when saving file

This feature applies when your project has a resource bundle for a locale matching your computer's locale

Version Control User Name: khaeger

2 In the Look for source files in path field, type the path to locate source files if they are not in the project.

Tip: If you add a plus sign (+) to the end of a path, Visual Cafe searches in that path and all subdirectories. For example, `c:\myapps+` tells Visual Cafe to search in `c:\myapps` and all of its subdirectories.

Inheriting the class path from the Windows environment

You can set up the `sc.ini` file so that Windows class path information is inherited by the entire Visual Cafe environment. However, you should realize that the class path information might not be useful for Visual Cafe projects and can cause delays whenever Visual Cafe searches for the source file corresponding to a class file when your Java programs are built. Some products that use class files do not have their own `.ini` file, so they put their class path information in the Windows environment.

For Windows 95 and 98, the class path is defined in the `autoexec.bat` file.

Windows NT can also use the class path in an `autoexec.bat` file, if it is present. For Windows NT 4.0 and higher, you can set the `CLASSPATH` variable by opening the Control Panel, then choosing System, then clicking the Environment tab and entering a value so that it appears in the System or User Variables list box.

Note: If you put a JAR in your class path, and the JAR is only on your development machine, do not use compression in that JAR file. Compression in a JAR that's in your class path will slow your compilation time.

To inherit the Windows class path setting:

1. In the `[Environment]` section of `\VisualCafe\Bin\sc.ini`, add the following specification at the end of the class statement:

   ```
   ;%classpath%
   ```

2. Restart Visual Cafe for the change to take effect.

Setting the class path for a Web browser

Before deploying your applet you might want your Web browser to be able to locate the Visual Cafe classes when the browser is launched outside of the Visual Cafe environment. The Visual Cafe installer can set this class path information for you. To set it manually, make sure the following is part of the class path information for your Windows environment:

```
C:\visualcafe\bin\components\symbeans.jar
```

Setting environment variables in the sc.ini file

You can set Visual Cafe environment variables (which don't affect the setting for your entire computer) by making changes to the `sc.ini` file. Visual Cafe saves all its environment settings in this file.

Most Visual Cafe environment setting changes that you'll want to make to the `sc.ini` file can be done easily and quickly by using the Internal VM tab of the Environment Options dialog box. For more information, see "Setting internal VM environment options" on page 5-28.

However, if there are still options that you want to customize that aren't available in the Internal VM tab, you can edit the `sc.ini` file directly. You can edit the `[Environment]` section of the `sc.ini` file, located in your Visual Cafe `Bin` folder. The statement syntax is similar to that for `autoexec.bat`, except that you omit the `set` keyword. Any changes take effect the next time you start Visual Cafe.

Note: By default, Visual Cafe does not read the environment settings for Windows, as specified in the `autoexec.bat` file. This is a change from earlier versions of Visual Cafe. You can, however, choose to inherit the class path from the Windows environment. For more information, see "Inheriting the class path from the Windows environment" on page 3-71.

Setting the class path

Class path directories are placed in a semicolon-delimited (;) list in the `sc.ini` file, and this list is passed to various tools, such as the compiler, AppletViewer, and so on. The default `classpath` statement is similar to this:

```
CLASSPATH=.;%@P%\..\JAVA\LIB;%@P%\..\JAVA\LIB\SYMCLASS.ZIP;%
 @P%\..\JAVA\LIB\SYMANTEC.ZIP;%@P%\..\JAVA\LIB\CLASSES.ZIP
```

To add directories, you should append them to this list, not delete the default `classpath` statement, because it can affect the operation of Visual Cafe. Remember to separate each directory with a semicolon (;). For example, to add `c:\development\classes`, append `;c:\development\classes` so that you have the following:

```
CLASSPATH=.;%@P%\..\JAVA\LIB;%@P%\..\JAVA\LIB\SYMCLASS.ZIP;%
 @P%\..\JAVA\LIB\SYMANTEC.ZIP;%@P%\..\JAVA\LIB\CLASSES.ZIP;
 c:\development\classes
```

Other examples

To set an environment variable `FOO` to the value `BAR`, you would add the following line to the `[Environment]` section of `sc.ini`:

```
FOO=BAR
```

The following example appends the system-wide value of `TEMP` to `C:\MYTEMP` and sets `TEMP` to this new value in the Visual Cafe environment:

```
TEMP=C:\MYTEMP;%TEMP%
```

So if the system-wide value `TEMP` is `C:\TEMP`, in Visual Cafe `TEMP` will be `C:\MYTEMP;C:\TEMP`.

Here `%@P%\` is a macro that translates to the `Bin` directory of Visual Cafe:

```
RESDIRE=%@P%\..\resdir
```

In a typical install this would be `c:\visualcafe\resdir`. Note that you must have the final backslash (\) for the `%@P%` macro to work.

Mapping Visual Cafe commands to key sequences

Visual Cafe lets you define a custom keystroke sequence for many Visual Cafe editing operations and macros. Use a command-key sequence to press keyboard keys to activate menu commands. Use macros to automate sequences of commands.

Your settings can be saved in a `.key` file, which is stored in the `\Bin\Keys` folder. Saving to a file allows you to reload or distribute the key assignments. A VJ key file is also available, so that you can use Visual J++ keyboard commands.

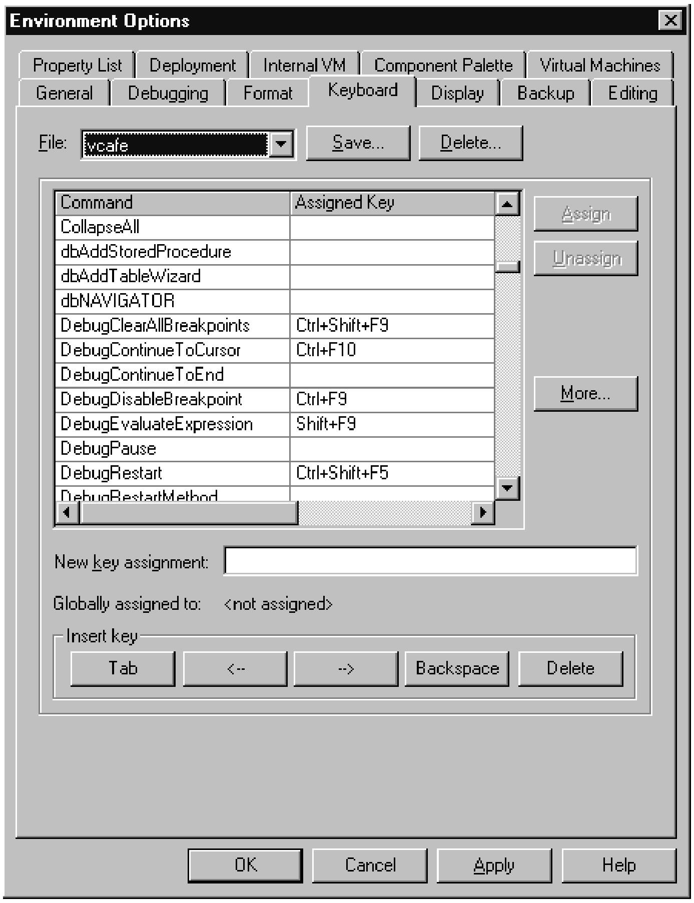
Macros are stored in the command list as `macro`*filename*. ScriptMaker macro files, stored in `\Bin\Macs\Src`, appear in the list.

Note: Help on keyboard commands is available in the Macro help. This help is available from the Help menu, and also by pressing F1 when you are editing macros. For example, from the Tools menu, choose Macro, then ScriptMaker, then click Edit; in the window that displays, press F1 to get the Macro help contents.

In the Keyboard view of the Environment Options dialog box, you can also set editing options for the Source window.

To access the Environment Options Keyboard view:

- From the Tools menu, choose Environment Options, then click the Keyboard tab.



Command	Assigned Key
CollapseAll	
dbAddStoredProcedure	
dbAddTableWizard	
dbNAVIGATOR	
DebugClearAllBreakpoints	Ctrl+Shift+F9
DebugContinueToCursor	Ctrl+F10
DebugContinueToEnd	
DebugDisableBreakpoint	Ctrl+F9
DebugEvaluateExpression	Shift+F9
DebugPause	
DebugRestart	Ctrl+Shift+F5
DebugRestartMethod	

New key assignment:

Globally assigned to: <not assigned>

From the Keyboard tab of the Environment Options dialog box, you can do te following:

- Choose a key file for use in the Visual Cafe environment
- Add a key file
- Delete a key file
- Map a command to a key sequence
- Delete a key assignment for a command

- Copy the command assignment list as text
- Modify what commands display
- Specify key-editing options for the Source window

Choosing a set of key assignments

You can specify what set of command key assignments to use in Visual Cafe by choosing a key file. You can choose key assignment files based on the Brief, Visual Cafe, Visual J++, Norton, and Emacs environments.

To choose a key file:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Choose a key file from the File drop-down menu. The options are as follows:

Option	Description
brief	Specifies the keyboard assignments used in the Brief programming environment.
emacs	Specifies the keyboard assignments used in the Emacs programming environment.
norton	Specifies the keyboard assignments used in the Norton programming environment.
untitled	Specifies the current set of keyboard assignments.
vcafe	Reverts to the default keyboard assignments in Visual Cafe.
vj	Specifies the keyboard assignments used in the Visual J++ programming environment.

   The keyboard assignments for that environment are loaded for use in the Visual Cafe environment.

3. Click OK.

   The changes take effect immediately.

Adding a set of keyboard assignments

You can add your own set of keyboard assignments for use in the Visual Cafe environment. When you start modifying the key assignments, Untitled appears in the list of available key files. You can save the current set of keyboard assignments under another name to create a new set of keyboard assignments. You can save it under an existing key file name to create a new version of that key file.

To add a key file:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Make your modifications. For more information, see “Mapping a command to a keyboard sequence” on page 3-77 and “Deleting the keyboard assignment for a command” on page 3-77.

   Untitled appears in the list of available key files.
3. Click Save.

   The Save Key Bindings dialog box displays.
4. Enter a file name, or select one from the drop-down list.
5. Click OK.

   The changes take effect immediately.

Deleting a set of keyboard assignments

You can choose to delete a set of keyboard assignments by deleting a key file.

To delete a key file:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Choose a key file from the File drop-down menu.
3. Click Delete.

   A dialog box displays, asking you to confirm the file deletion.
4. Click OK.

   The changes take effect immediately.

Mapping a command to a keyboard sequence

You can define a custom keystroke sequence for many editing operations and macros.

To map a command to a key sequence:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Choose a key file from the File drop-down menu.
3. From the command list, select a command.

   If an item has more than one key assignment, the item appears once for each key assignment.

   Tip: Click the column header to sort the list by command or key assignment, and by forward or reverse alphabetical order.

4. To specify a key assignment, click in the New Key Assignment field and specify the key sequence as follows:
   - ❖ Press the key sequence on the keyboard.
   - ❖ Click a button in the Insert Key area.

     If the value you enter is already mapped to a command or macro, that command name appears in the Assigned To area.

   Tip: A command can have multiple key assignments. If the command already has a key assignment and you add another assignment, the command now has two separate assignments.

5. Assign the key sequence to the command by clicking Assign.

   The assignments are automatically saved in an untitled file.

6. To save the settings to a file, click Save.

   In the dialog box, specify the name of a new or existing `.key` file. Saving to a file allows you to reload or distribute the key assignments. Key files are stored in the `\Bin\Keys` folder.

Deleting the keyboard assignment for a command

You can easily delete a keyboard assignment for a command.

To delete the key assignment for a command:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Choose a key file from the File drop-down menu.
3. From the Command list, select a command.

   If an item has more than one key assignment, the item appears once for each key assignment.

   Tip: Click the column header to sort the list by command or key assignment, and by forward or reverse alphabetical order.

4. Select the command in the Command list and click Unassign.

   The key assignment is deleted.

Copying the command assignment list as text

You can save the set of keyboard assignments as text and use it in another window, such as the Source window. This way you can easily create a list of all your command-key combinations.

To copy the command assignment list as text:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.
2. Right-click in the Command list and select Copy All.

   You can paste the list into another window, such as the Source window.

Modifying which commands are shown

You can modify what types of commands are shown in the list of commands for keyboard assignments. You can choose to display only the commands that have key assignments, or all commands, thus including unbound commands. You can also choose to display only member, text, or class commands, or all three command types.

To modify what commands are shown:

1. From the Tools menu, choose Environment Options, then click the Keyboard tab.

2 To view all commands, including commands that aren't bound to key assignments, right-click in the Command list and select Unbound Commands.

You can deselect this option to view only the commands that have key assignments.

3 To view commands that begin with either Member, Text, or Class, right-click and select the desired option. Select All to view all commands.

The changes take effect immediately.

Specifying key-editing options for editing source code

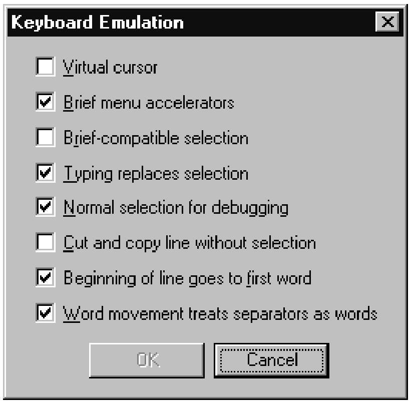
These option settings apply to all Source windows.

To specify editing options:

1 From the Tools menu, choose Environment Options, then click the Keyboard tab.

2 Click More.

The Keyboard Emulation dialog box displays.


Keyboard Emulation

- [ ] Virtual cursor
- [✓] Brief menu accelerators
- [ ] Brief-compatible selection
- [✓] Typing replaces selection
- [✓] Normal selection for debugging
- [ ] Cut and copy line without selection
- [✓] Beginning of line goes to first word
- [✓] Word movement treats separators as words

3 Select the options you want:

Option	Description
Virtual cursor	When selected, you can position the cursor anywhere in the file, regardless of the placement of the line endings. When deselected, you can't position the cursor past the end of a line. This option is deselected by default.
Brief menu accelerators	Disables menu keys. After this is selected, any new windows will not have any underscores beneath the top-level menu items. This option is selected by default.
Brief-compatible selection	If you choose this option, the editor stays in the selection mode when you use the arrow keys. This option is deselected by default.
Typing replaces selection	When selected, the source editor follows the Windows standard convention of replacing any selected text with the first character typed or pasted. When deselected, typing or pasting inserts the text to the left of the current selection. This option is selected by default.
Normal selection for debugging	Enables normal selection of text when in debugging mode. If cleared, you can drag from the Source window to the Variables, Watch, and Thread windows while debugging. This option is selected by default.
Cut and copy line without selection	When selected, you can quickly cut or copy the current line using the standard cut or copy keystrokes (without having to first select the line). When deselected, you can cut or copy a line by using the keys assigned to the `EditorCutLine` or `EditorCopyLine` functions. This option is deselected by default.

4 Click OK.

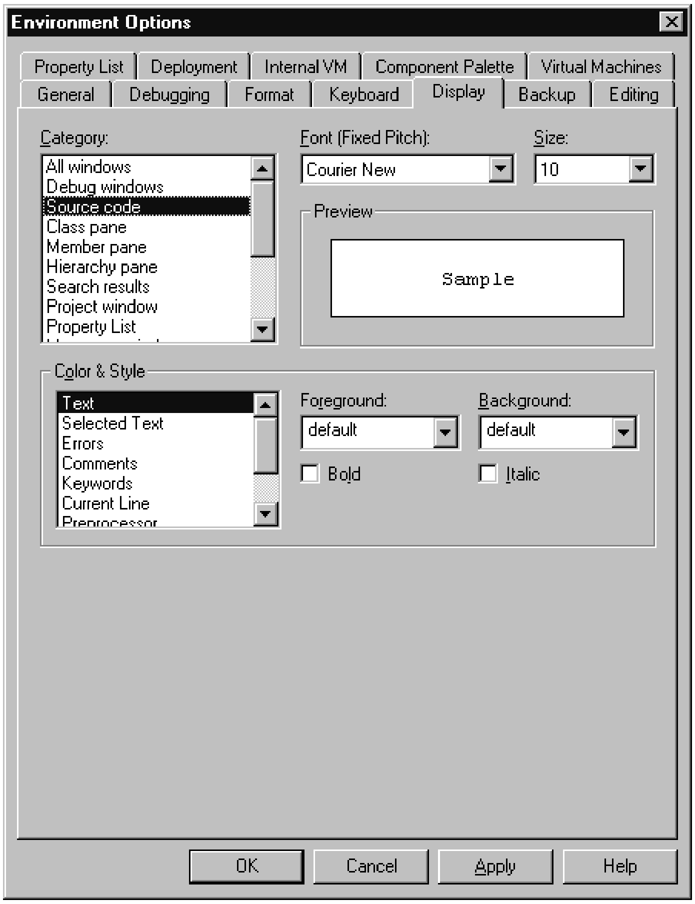
The changes take effect immediately.

Customizing the display font and color

Visual Cafe allows you to set the font, font style, and font color for development environment windows, Visual Cafe editors, and window items. You can also change the appearance of elements in your source code, such as comments, keywords, current line, and breakpoints.

To access the Environment Options Display view:

From the Tools menu, choose Environment Options, then click the Display tab.



Category: All windows, Debug windows, Source code, Class pane, Member pane, Hierarchy pane, Search results, Project window, Property List

Font (Fixed Pitch): Courier New

Size: 10

Color & Style: Text, Selected Text, Errors, Comments, Keywords, Current Line, Preprocessor

Foreground: default

Background: default

Bold

Italic

From the Display tab of the Environment Options dialog box, you can specify the following environment options:

- ◆ Setting the font name and size

◆ Setting the font color and style

You can change the font display settings for any of the following elements in the Visual Cafe environment:

All windows	Property List
Debug windows	Messages window
Source code	Breakpoints window
Class pane	Variables window
Member pane	Watch window
Hierarchy pane	Threads window
Search results	Call Stack window
Project window	Component Library window

Most of the above elements affect regular text and selected text. You can also change how Java interface text displays in the Hierarchy pane or Class pane.

When setting font display options for Source code, you can make separate font settings for each of the following source elements:

Environment element	Description
Errors	Lines where compiler errors are found
Comments	Java comments
Keywords	Java keywords
Current Line	The line that contains the insertion point
Preprocessor	Java preprocessor directives
Custom Keywords	Special keywords that you define
Execution Line	During debugging, the current execution line

Setting the font name and size

You can set the font and size of text that appears in development environment windows, editors, and window items in the Visual Cafe environment. Use the Preview area to see what the font settings look like before setting them.

To set the font name and size:

1 From the Tools menu, choose Environment Options, then click the Keyboard tab.

2 In the Category list, choose the item you want to modify. From the Font drop-down list, choose the font that you want.

 In the Preview area, you can see how the text will look.

3 From the Size drop-down menu, choose the font size that you want.

 In the Preview area you can see how the text will look.

4 Click OK.

 The changes take effect immediately.

Setting the font color and style

You can set the font color and style of text that appears in development environment windows, editors, and window items in the Visual Cafe environment. You can choose to set a foreground color as well as a background color. Use the Preview area to see what the font settings look like before setting them.

To set the font color and style:

1 From the Tools menu, choose Environment Options, then click the Keyboard tab.

2 In the Category list, choose the item you want to modify.

3 Select an item in the Color & Style list, then choose a value for Foreground or Background, or select Bold or Italic.

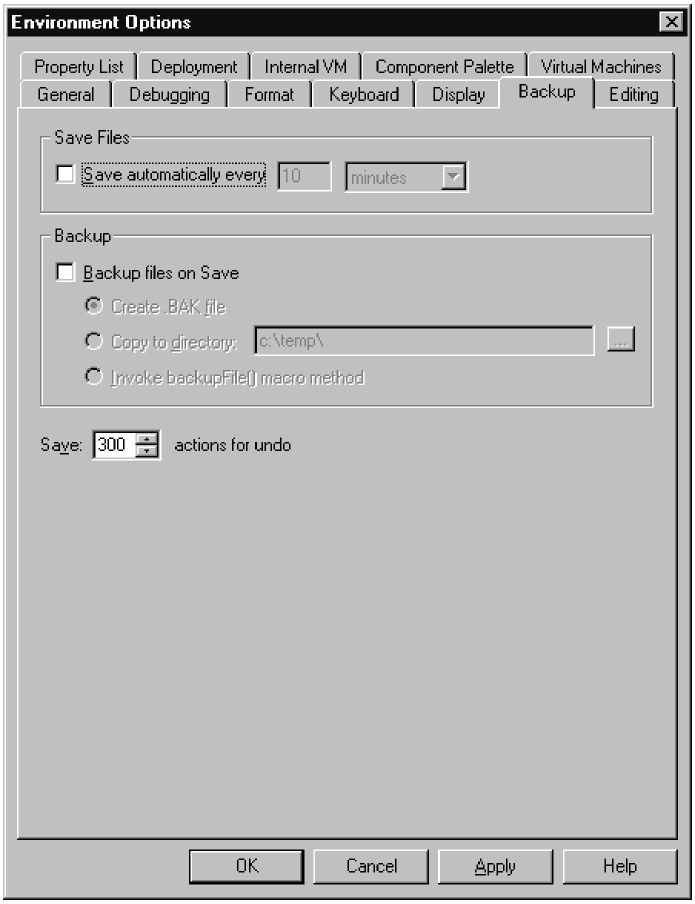
 The default setting is the default text color as set in the Windows Control Panel.

Setting backup and save options

You can choose to back up, or save a copy of, source files in a temporary location. This is helpful to prevent data loss in case files get corrupted or damaged somehow, of if you want to revert to an earlier saved version of a file. If you wish, you can also automate source-file backups.

To access the Environment Options Backup view:

◆ From the Tools menu, choose Environment Options, then click the Backup tab.



Environment Options

Save Files

☐ Save automatically every 10 minutes

Backup

☐ Backup files on Save

● Create .BAK file

○ Copy to directory: c:\temp\

○ Invoke backupFile() macro method

Save: 300 actions for undo

Saving files for recovery purposes

You can automate the saving of files in temporary locations so you can recover changes if there is a system failure.

To establish file saving:

1 From the Tools menu, choose Environment Options, then click the Backup tab.

2 Select Save Automatically and choose a time interval.

When selected, each modified edit buffer is saved at regular intervals in a temporary file in the `temp` home directory. Under normal circumstances, these temporary files are automatically deleted when the editor exits. If a system crash or power failure occurs, the editor will not exit normally and these temporary files will not be deleted. This permits you to recover work that would otherwise be lost. The temporary files created by the autosave function have names that begin with the character ~, followed by a unique number and the extension `.sav`; for example, `~4289352.sav`.

For identification purposes, the autosave function adds a line in the following format at the beginning of each temporary file:

```
; Visual Cafe AUTOSAVE C:\DIR\FILENAME.EXT 10-12-98
 7:35 pm
```

This line contains the complete path name of the corresponding file and the date and time of the autosave, formatted as a Java language comment. The rest of the temporary file, starting with line 2, stores the contents of the buffer at the time the autosave was performed.

To recover from a system crash or power failure, examine each file with the extension .`sav` in the `temp` directory.

Automating source file backups

You can control whether files in a project are automatically backed up each time that a save is performed. When a file is backed up, a copy of it is saved with the file extension `.bak`. You can also specify the location and name of the backup files.

Note: Only Java and HTML files that have changed are backed up. For example, if you save all files in a project, only the files that have changed are backed up.

To automate project backups:

1 From the Tools menu, choose Environment Options, then click the Backup tab.

2 Select Backup files on Save.

3 Select the location and name of the backup files:

Option	Description
Create BAK file	Create one or more backup files that are named *file*`.bak`.
Copy to directory	Copy the source files to the specified directory. You can type the directory with the full path into the text box, or click **...** to select a directory from a dialog box. There is no default.
Invoke OnBackup script	Run your own macro, created with the Visual Cafe macro utility and containing an `BackupFile` method. All macros are placed in the `\VisualCafe\Bin\Macs\Src` folder.

CHAPTER 4

Working with Source Code

This chapter describes how to edit your projects' Java source code in Visual Cafe. You'll also learn how to add your own packages or third-party packages to Visual Cafe.

Classes are the foundation of object-oriented programming. To make working with classes easier, Visual Cafe provides several powerful features:

- Class Browser
- Insert Class Wizard
- Hierarchy Editor
- Source window

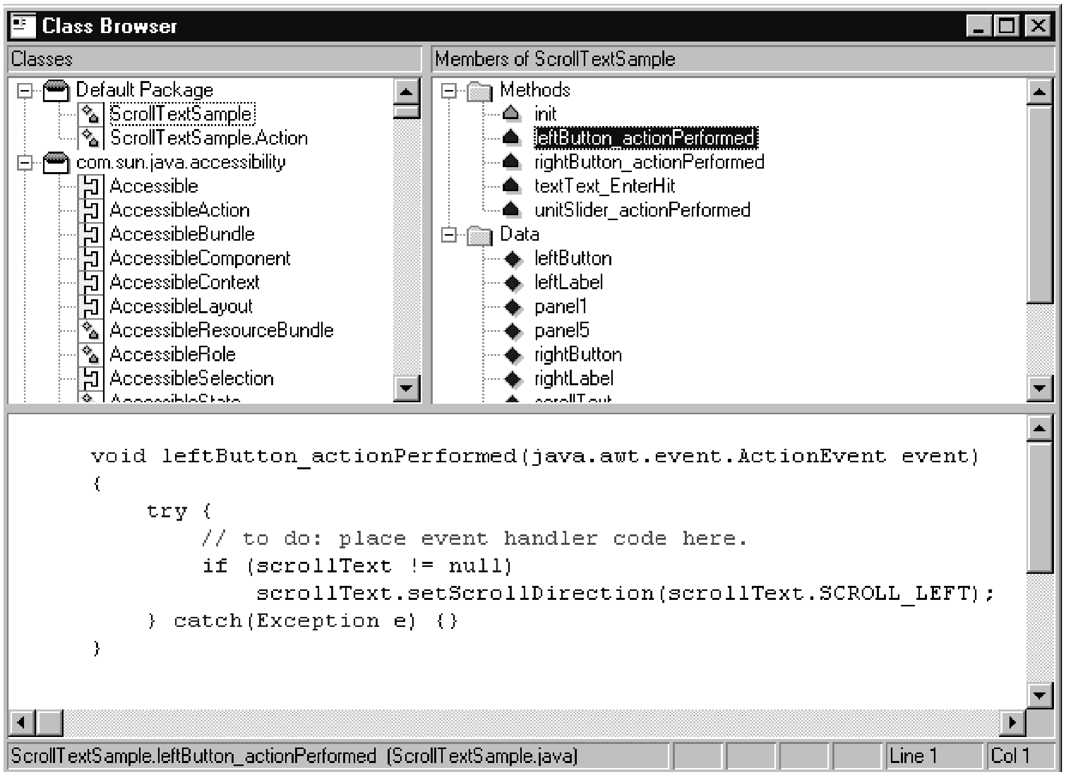
In this chapter you'll learn how to use these features to enhance your programs.

The Class Browser and Source window are designed with object-oriented program development in mind. The Class Browser and Source window now include support for JDK keywords.

About classes, members, and the Class Browser

It's not very easy to read a 27-page source file filled with references to multiple classes; you need to be able to keep track of what each of the classes does, what data is in each class, and how the methods within the classes work. The Class Browser helps you work with your source code in an organized way.

The Class Browser is a three-pane window that lists all the classes, methods, and data items contained in your program. This tool provides abstraction from the underlying source files by letting you navigate and edit your classes and members quickly. In the Class Browser you're free from the clutter of other member implementations in the same source file.



```
void leftButton_actionPerformed(java.awt.event.ActionEvent event)
{
    try {
        // to do: place event handler code here.
        if (scrollText != null)
            scrollText.setScrollDirection(scrollText.SCROLL_LEFT);
    } catch(Exception e) {}
}
```

ScrollTextSample.leftButton_actionPerformed (ScrollTextSample.java) Line 1 Col 1

The Class Browser window shows the class hierarchy in your project and allows you to add classes, modify extension relationships, and view and edit class member **declarations** and definitions. The Class Browser window shows **data members** and **methods** for each class and an edit area for working directly with the body of a method.

You see only the methods and data members for the selected class, and only the Java code for the selected member. The isolation of member source code provides an extra degree of security by ensuring that you don't unintentionally change code outside of the object's scope. Each member has a color-coded icon to indicate its access privileges (green is public, yellow is protected, blue is package, and red is private).

There are three panes in the Class Browser: the Classes, Members, and Source panes. Both the Classes and Members panes support keyboard incremental searches. As you type the name of a class or member, the list of matching objects is refined until the desired class is automatically selected.

Pop-up menus are available for each pane when you right-click. You can select multiple items in a pane by using SHIFT-click.

You can display classes and members in a variety of views and filter the items that are displayed. When you're showing classes by a hierarchical view and a class implements one or more interfaces, the class displays below each interface.

Classes defined in subprojects are not displayed in a project's Class Browser window — except for base classes from which one or more classes in a project are derived. These base classes are listed in italics in the Classes pane; you can't examine these classes in the Members or Source panes. To browse classes defined in a subproject, make the subproject the frontmost project, then open a new Class Browser window.

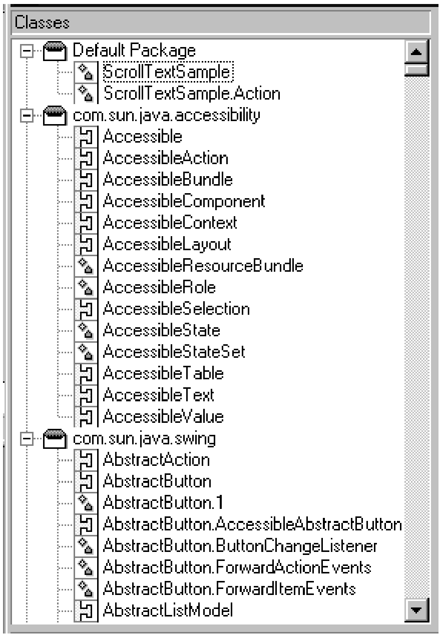
Whereas when you work in the Source window you're asked if you want to save changes when you close the window, in the Class Browser you're not prompted. Any changes you make to classes or inheritance relationships modify the source code immediately as you move between members or classes.

Note: Visual Cafe does not allow you to change the structure of the standard Java hierarchy or the source code of standard Java classes.

For more information, see "Working with classes" on page 4-6 and "Working with members" on page 4-27.

About the Classes pane

The Class Browser's Classes pane displays all the classes that are part of the current project. By default, the classes are displayed by package. Your project files will be displayed as part of the default package when you start Visual Cafe because you haven't created a package structure yet. In this case, the default package represents the entire project. This view also provides an outline in which subclasses display indented and below their parent. A class that implements interfaces appears below each interface class.

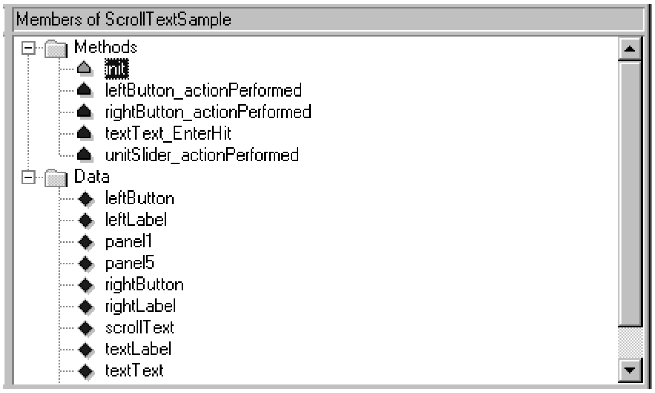


The classes may be listed alphabetically, by package, or hierarchically. You can choose to display your classes either alphabetically or by package.

See “Using the Classes pane” on page 4-12 for more information.

About the Members pane

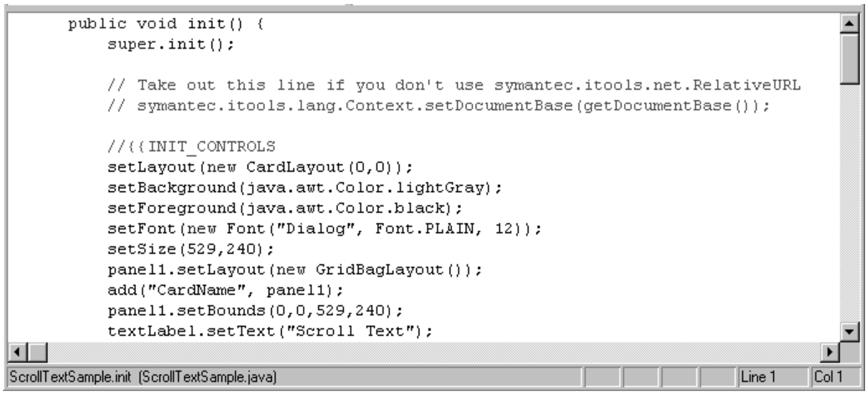
The data and methods in a class are called **members** of that class. The Class Browser’s Members pane gives you an organized look at the data and methods the selected class is made of. The color of an icon shows level of access for a class: yellow means it's protected, green means it's a public class, and blue means it's a package.



See "Using the Members pane" on page 4-13 for more information.

About the Source pane

The Source pane displays the source code for the member data or method that's selected in the Members pane. You can edit the code for your programs in the Source pane. All changes made to the class using the Class Browser will automatically be added to the associated source file.



```
    public void init() {
        super.init();

        // Take out this line if you don't use symantec.itools.net.RelativeURL
        // symantec.itools.lang.Context.setDocumentBase(getDocumentBase());

        //{{INIT_CONTROLS
        setLayout(new CardLayout(0,0));
        setBackground(java.awt.Color.lightGray);
        setForeground(java.awt.Color.black);
        setFont(new Font("Dialog", Font.PLAIN, 12));
        setSize(529,240);
        panel1.setLayout(new GridBagLayout());
        add("CardName", panel1);
        panel1.setBounds(0,0,529,240);
        textLabel.setText("Scroll Text");
```

The Source pane displays the source code for a class definition, a member definition, or a data definition. All editing operations available in the Source window are available in the Source pane. Any editing operation done within the Source pane is synchronized with all open Source windows.

Note: The Class Browser's Source pane has the same editing functions as the Source window. See "Using the Source window" on page 4-41 for more information.

Working with classes

When working with the classes, you can perform the following tasks:

- ◆ Configuring the Class Browser
- ◆ Opening a Class Browser window
- ◆ Using the Classes pane
- ◆ Using the Members pane
- ◆ Using the Source pane
- ◆ Using the Insert Class Wizard
- ◆ Adding a class
- ◆ Copying or moving a class
- ◆ Renaming a class
- ◆ Viewing and editing the source code for a class
- ◆ Deleting a class
- ◆ Working with members

These topics are discussed in the following sections.

Configuring the Class Browser

You can choose how the Class Browser displays information. You can also enable multiple-component selection and select confirmation options for the Class Browser. When the Class Browser opens, all packages are expanded.

Tip: To collapse all packages, select a package and press SHIFT and - (the hyphen key) at the same time.

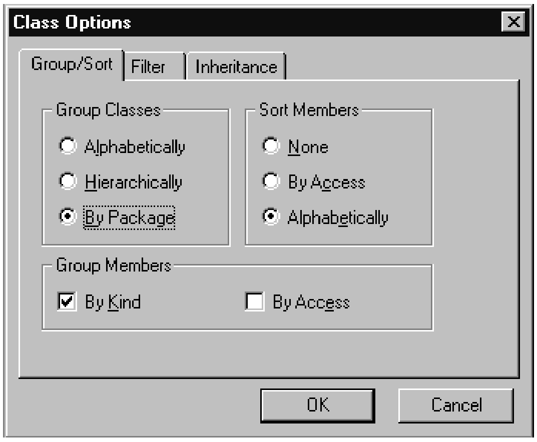
To configure the Class Browser window:

- Drag the sides of each pane to adjust the display of its contents.
  or
- Use the Classes menu to:
  - toggle the display of pane titles on and off
  - toggle the class list between alphabetical and hierarchical order

To change the way in which classes and members are displayed:

1. In the Classes pane, choose Options from the Classes menu, or right-click and choose Options from the pop-up menu.

   The Class Options dialog box appears.



Class Options

Group/Sort | Filter | Inheritance

Group Classes: ( ) Alphabetically ( ) Hierarchically (●) By Package

Sort Members: ( ) None ( ) By Access (●) Alphabetically

Group Members: [✓] By Kind [ ] By Access

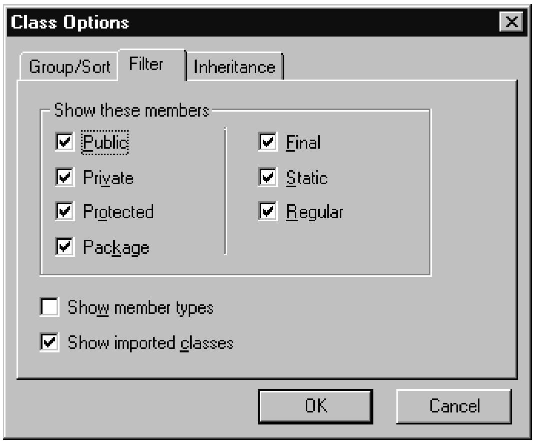
OK | Cancel

2. To change the ordering of classes and member elements in the Class Browser, click the Group/Sort tab and select from the following options:

Option	Description
Group Classes	Specifies how the classes in the Classes pane are to be grouped. You can group classes alphabetically, hierarchically, or by package.

Option	Description
Sort Members	Specifies how the members in the Members pane are to be sorted in their group. Grouping of members is controlled with the Group Members option. The None option sorts the elements based on the order in which they were created.
Group Members	Defines the grouping of members in the Members pane. The By Kind option groups the elements as methods or data. The By Access option groups the elements by their access type. For more information about viewing a member's access type, see "Viewing a member's attributes" on page 4-33.

3 To filter classes and members, click the Filter tab:


Class Options

Group/Sort | Filter | Inheritance

Show these members
- [✓] Public
- [✓] Private
- [✓] Protected
- [✓] Package
- [✓] Final
- [✓] Static
- [✓] Regular

- [ ] Show member types
- [✓] Show imported classes

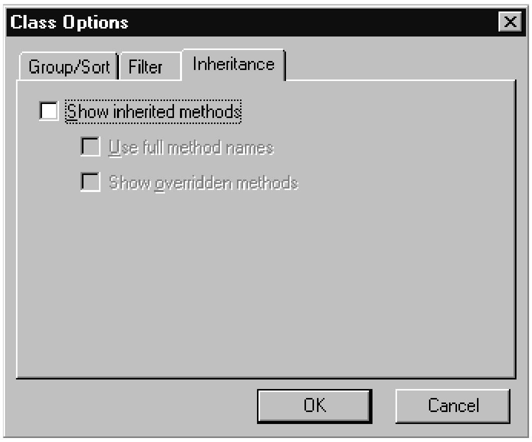
OK | Cancel

Choose from the following options:

Option	Description
Show these members	Defines the type of members to display in the Members pane. Options include access type and method types. There must be at least one option selected in each group.

Option	Description
Show member types	Includes the method's argument in the listing.
Show imported classes	Clearing this option prevents imported classes from displaying in the Class and Members panes.
Final, Static, Regular	These are Java method types. Refer to your Java documentation for more information.

4 To specify inheritance options, click the Inheritance tab:



☐ Show inherited methods
☐ Use full method names
☐ Show overridden methods

To set the Class Browser to show inherited methods, select the Show inherited methods option. This activates two suboptions.

- ❖ To add the method's package and class name to the method names listing, select Use full method names.
- ❖ To add methods to the member listing that are overridden by methods in the class, select Show overridden methods.

5 Click OK.

The changes take effect immediately.

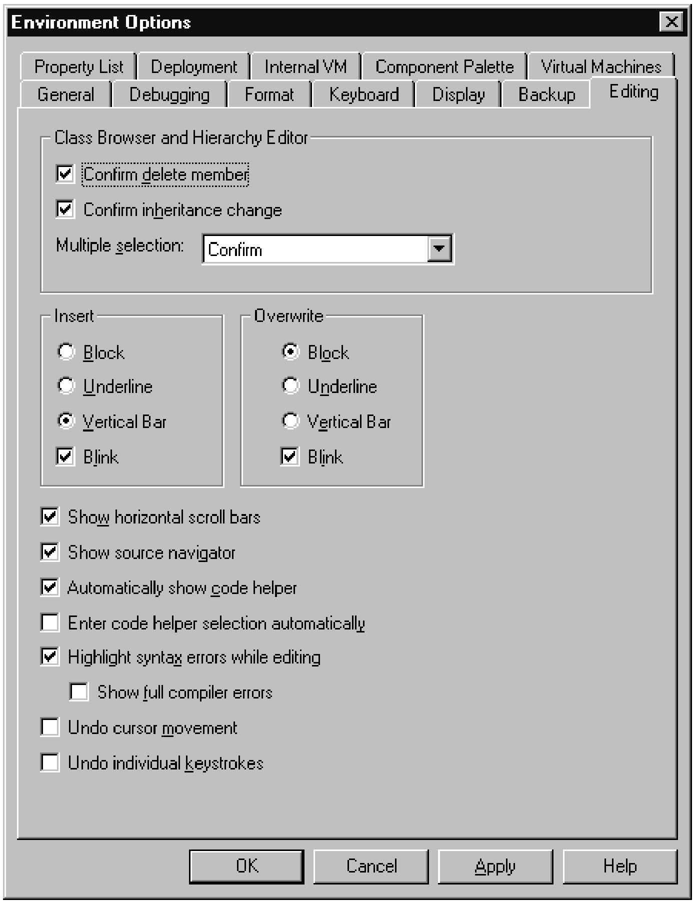
Configuring the Class Browser and Hierarchy Editor

You can specify how both the Class Browser and Hierarchy Editor respond to member deletions, inheritance changes, and the selection of multiple items.

For more information about the Hierarchy Editor, see “About the Hierarchy Editor” on page 4-35 and “Using the Hierarchy Editor” on page 4-36.

To set Class Browser and Hierarchy Editor options:

1 From the Tools menu, choose Environment Options. Click the Editing tab.



Class Browser and Hierarchy Editor

[✓] Confirm delete member

[✓] Confirm inheritance change

Multiple selection: Confirm

Insert

( ) Block

( ) Underline

(●) Vertical Bar

[✓] Blink

Overwrite

(●) Block

( ) Underline

( ) Vertical Bar

[✓] Blink

[✓] Show horizontal scroll bars

[✓] Show source navigator

[✓] Automatically show code helper

[ ] Enter code helper selection automatically

[✓] Highlight syntax errors while editing

[ ] Show full compiler errors

[ ] Undo cursor movement

[ ] Undo individual keystrokes

2 In the Class Browser and Hierarchy Editor area, select or clear appropriate options:.

Option	Description
Confirm Delete Member	Requires confirmation of member deletions.

Option	Description
Confirm Inheritance Change	Requires confirmation of inheritance deletions.
Multiple Selection	Specifies whether you can select multiple classes in the Class Browser and Hierarchy Editor. If you select Confirm, you can select multiple classes; changes to multiple selections require confirmation.

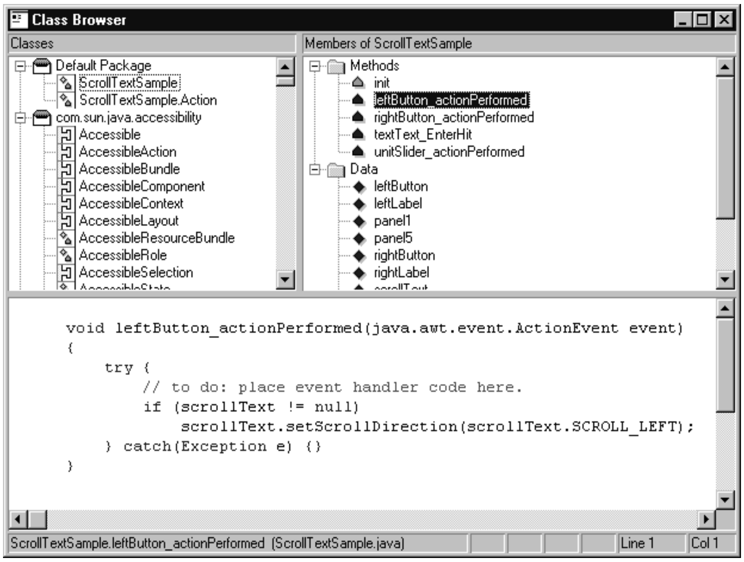
Opening a Class Browser window

You can open the Class Browser to see classes, methods, and data variables, and you can have more than one Class Browser window open at a time.

To open a Class Browser window:

◆ From Visual Cafe's View menu, choose Class Browser.

If you choose this command with a Class Browser window already open, the Class Browser window becomes the frontmost window.



```
void leftButton_actionPerformed(java.awt.event.ActionEvent event)
{
    try {
        // to do: place event handler code here.
        if (scrollText != null)
            scrollText.setScrollDirection(scrollText.SCROLL_LEFT);
    } catch(Exception e) {}
}
```

The following information is displayed in the Class Browser window:

- Classes pane — shows all classes defined for the project after compiling the project
- Members pane — shows the member functions and data members for a selected class
- Source pane — shows the source code for a class, member, or data item

The Class Browser's Classes, Source, and Members panes are all lists. You scroll a list until an item is visible. You can also type the first few letters of the item's name to have the list automatically scroll to the first item that begins with those letters.

Note: You can change the relative size of the panes by dragging the size bars. Once you've established a new relative size for a pane, it's maintained when the window is resized.

To open a new Class Browser window when one or more Class Browser windows are already open:

- Choose New Window from the Window menu.
  The Class Browser opens.

Note: You can't switch a Class Browser window to a different project after it's been opened. To examine the classes of a different project, make the other project the active project, then open a new Class Browser window.

Using the Classes pane

If you select a class in the Classes pane, the member functions it implements are displayed in the Members pane. The data members it defines are also displayed in the Members pane. Inherited member functions and data members are not displayed. You may display a class declaration in the Source pane by double-clicking its name in the Classes pane.

By default, classes appear alphabetically by full package name. The classes in the default package are displayed first. If you wish, you can change the way in which classes are displayed in the Classes pane.

To specify how classes appear, see "Configuring the Class Browser" on page 4-6.

Note: In order to have a class to appear in the Class Browser, the file that the class is in must be a part of the current project.

To locate a class in the Classes pane:

1 Click in the Classes pane, then type the class name.

For example, to locate `java.awt.FlowLayout`, type `flo`, press TAB to go to the next entry with that letter sequence.

As you type, selections are made to match the text you enter. As you continue typing, the search is refined.

2 The class you're searching for is highlighted.

Notes: In the Class Browser the search is conducted by package, and only expanded packages are searched. If you want to locate a class in a particular package, you need to expand the package by clicking the +, then start typing. If you want to exclude a package from a search, click the - to collapse the package. You can also press SHIFT-+ and SHIFT- – to expand and collapse packages.

If the class you want is not displayed in the pane, you might need to change which classes are displayed, as described next.

Using the Members pane

By default, the methods and data members display in two groups, with each element sorted alphabetically within the group. Each element has a color-coded icon to indicate its access privileges (green means public, yellow means protected, blue means package, and red means private.) If you wish, you can change the way in which members are displayed in the Members pane.

See "Working with members" on page 4-27 for more information.

To change the way in which inherited methods are displayed:

1. While the Class Browser is the active window, choose Options from the Classes menu. Alternatively, you can right-click in the Class Browser's Members pane and choose Options.

   The Class Options dialog box appears.
2. Click the Inheritance tab.
3. Select Show inherited methods, or deselect it, as appropriate.

   If selected, these suboptions are now available to select or deselect:

   ❖ Use full method names

     Selecting this option adds the method's package and class name to the method names in the listing.

   ❖ Show overridden methods

     This option adds methods that are overridden by methods in the class to the Member listing.
4. Click OK.

   The changes take effect immediately.

Using the Source pane

You can view source code in the Source pane of the Class Browser or in a Source window.

Note: The Class Browser's Source pane has the same editing functions as the Source window. See "Using the Source window" on page 4-41 for more information.

To display source code in the Class Browser's Source pane:

◆ Select a class in the Classes pane, then a member in the Members pane.

To display source code in the Source window:

◆ Select a class from the Class Browser's Classes pane, then choose Go to Source from the Classes menu; alternatively, you can right-click a class and choose Go to Source. See "Using the Source window" on page 4-41 for more information.

Using the Insert Class Wizard

The Insert Class Wizard makes creating new Java classes and interfaces easier and more foolproof by setting up a complete prototype for you. You can also edit existing classes.

The following enhancements have been added to the Insert Class Wizard:

- ◆ Bean option
- ◆ Properties page
- ◆ Events page

Launch the Insert Class Wizard from the Insert menu, Class Browser, or Hierarchy Editor.

To define a new class or interface with the Insert Class Wizard:

1 To define a new class or interface, do one of the following:

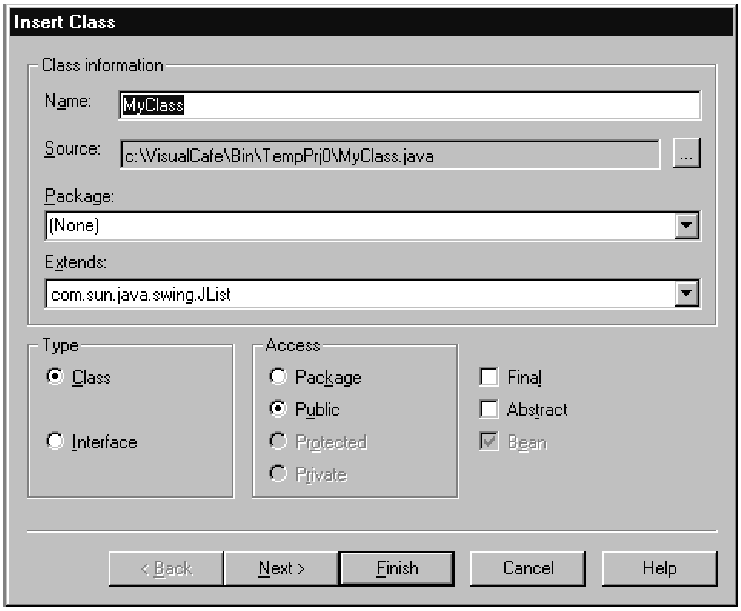
  - ❖ From the Insert menu, choose Class.
  - ❖ Select a class or interface in the Classes pane of the Class Browser, then right-click and choose Insert Class.
  - ❖ Select a class or interface in the Hierarchy Editor, then right-click and choose Insert Class.
  - ❖ Select a class or interface in the Hierarchy Editor, then drag a line from it.
  - ❖ By default (as set in the Environment Options dialog box) you can press INSERT from the Class Browser or Hierarchy Editor.

  When you select a class, that class becomes the default class to inherit from.

  To edit a class or interface, do one of the following:

  - ❖ Select a class or interface in the Classes pane of the Class Browser, then choose Edit Class from the Classes menu, or right-click and choose Edit Class.

2 On the first page of the wizard, specify the options that you need:



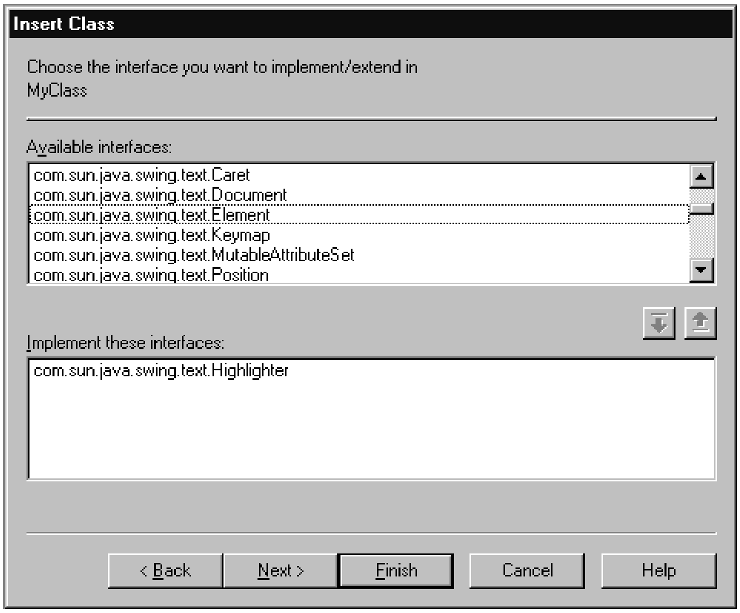
Name: MyClass
Source: c:\VisualCafe\Bin\TempPrj0\MyClass.java
Package: (None)
Extends: com.sun.java.swing.JList
Type: [✓ Class] Interface
Access: Package [✓ Public] Protected Private
☐ Final
☐ Abstract
☑ Bean

Option	Description
Type	Select Class if you're creating a class, or Interface if you're creating an interface.
Name	Type the name of the class or interface.
Source	Type the complete path to the Java file. Click the Browse button (...) to browse. In the Edit Class File dialog box, you can specify the package and Java file name; the file path is displayed for you.
Package	Choose a package to add the class or interface to, or None if you don't want to modify the existing package.
Extends	If you're defining a class, choose a class to extend from.

Option	Description
Access	Select whether you want to make access to your class through Package, Public, Private or Protected. Public is available only if the class name and file name are the same; Protected and Private are for inner classes only.
Final or Abstract	Select Final class or Abstract class.
Bean	Select Bean if you want a class constructor method that takes zero parameters (it has a null constructor). Remember that the minimum requirements of a Bean are that it can be instantiated (it is not an abstract class or interface), it is public, and it has a public class constructor method that takes zero parameters.

3 Click Finish if you're finished with the definition, or Next to continue.

The next page of the wizard appears, where you can choose the interfaces to implement:


Insert Class

Choose the interface you want to implement/extend in MyClass

Available interfaces:

com.sun.java.swing.text.Caret
com.sun.java.swing.text.Document
com.sun.java.swing.text.Element
com.sun.java.swing.text.Keymap
com.sun.java.swing.text.MutableAttributeSet
com.sun.java.swing.text.Position

Implement these interfaces:

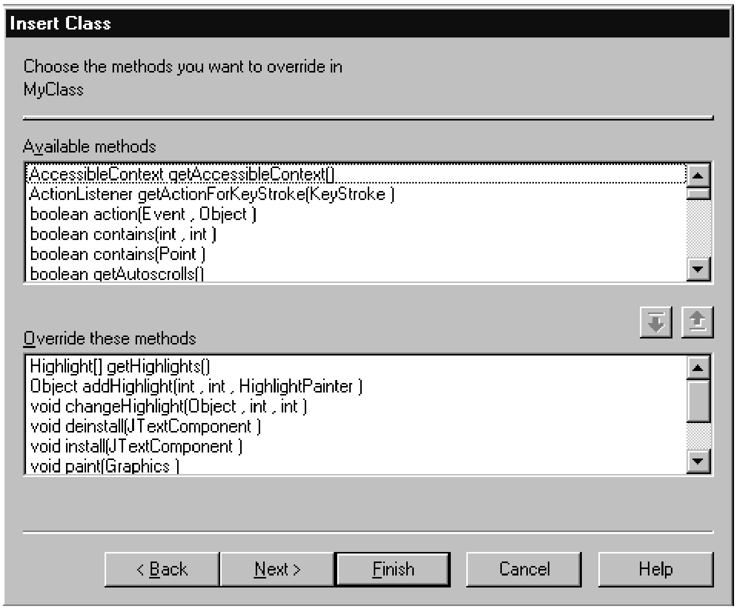
com.sun.java.swing.text.Highlighter

4. From the Available interfaces list, select the interfaces you want to implement and click the button with a downward-pointing arrow.

   To move an interface from the lower list box to the upper one, select the interface and click the button with a upward-pointing arrow.

5. Click Finish if you're finished with the definition, or Next to continue.

   The next page of the wizard appears, where you can choose the methods to override. Required methods appear in the Override these methods list box.


Insert Class

Choose the methods you want to override in MyClass

Available methods

AccessibleContext getAccessibleContext()
ActionListener getActionForKeyStroke(KeyStroke)
boolean action(Event , Object)
boolean contains(int , int)
boolean contains(Point)
boolean getAutoscrolls()

Override these methods

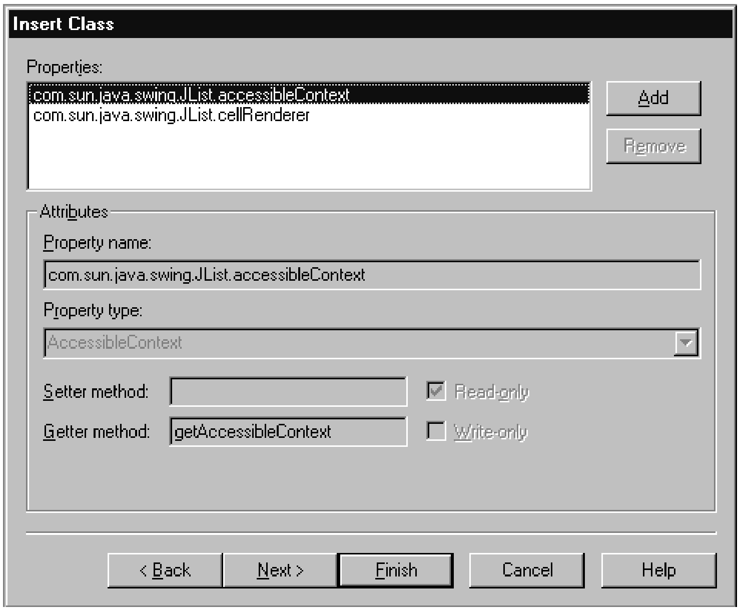
Highlight[] getHighlights()
Object addHighlight(int , int , HighlightPainter)
void changeHighlight(Object , int , int)
void deinstall(JTextComponent)
void install(JTextComponent)
void paint(Graphics)

6. From the Available methods list, select the methods you want to override and click the downward-pointing arrow.

   To move a method from the lower list box to the upper one, select the method and click the upward-pointing arrow. You can't move methods that are required.

7. Click Finish if you're finished with the definition, or Next to continue.

The next page of the wizard appears, where you can work with properties.



Properties:
com.sun.java.swing.JList.accessibleContext
com.sun.java.swing.JList.cellRenderer

Property name: com.sun.java.swing.JList.accessibleContext
Property type: AccessibleContext
Setter method:
Getter method: getAccessibleContext
[✓ Read-only]
Write-only

8 Specify properties.

- ❖ To add a property, click Add and specify the characteristics of the property.
- ❖ To remove a property, select a property and click Remove. You cannot remove properties that were introspected from the base class.
- ❖ To edit a property, select the property and specify the characteristics of the property.

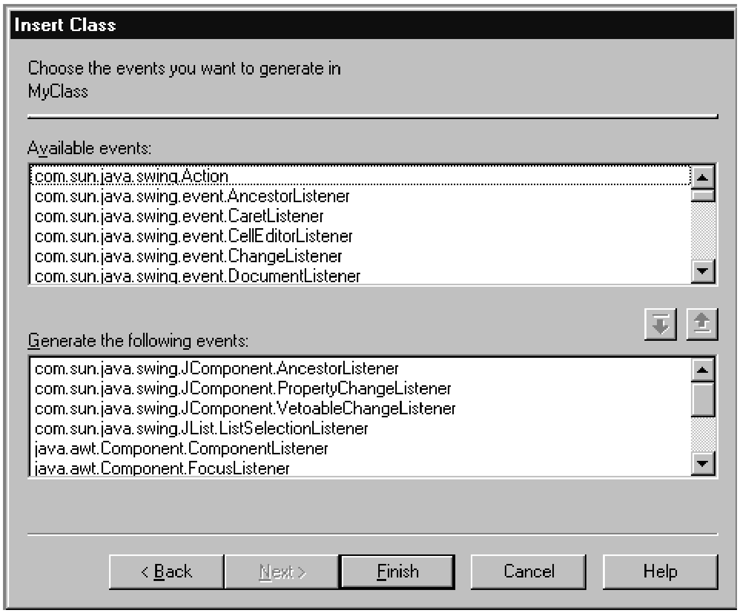
Here are descriptions of the property characteristics you can enter:

Field	Description
Name	Name of the property. This field is required.
Type	Choose a type from the drop-down list.
Setter method	Visual Cafe introspects the setter method from the base class.
Getter method	Visual Cafe introspects the getter method from the base class.

Field	Description
Default value	The default value of the property. It should be valid for the type you chose.
Read-only	When you select Read-only, there is no setter method.
Write-only	When you select Write-only, there is no getter method.
Bound	Selecting this option generates code that takes care of creating notifications so that other objects get notified when the property changes value.
Constrained	Selecting this option generates code that takes care of creating notifications so that other objects get notified before the property changes value and can reject the change.
Transient	When you select Transient, the property is not serializable.

9 Click Finish if you are finished with the definition, or Next to continue.

The next page of the wizard appears, where you choose events to generate:


Insert Class

Choose the events you want to generate in MyClass

Available events:

```
com.sun.java.swing.Action
com.sun.java.swing.event.AncestorListener
com.sun.java.swing.event.CaretListener
com.sun.java.swing.event.CellEditorListener
com.sun.java.swing.event.ChangeListener
com.sun.java.swing.event.DocumentListener
```

Generate the following events:

```
com.sun.java.swing.JComponent.AncestorListener
com.sun.java.swing.JComponent.PropertyChangeListener
com.sun.java.swing.JComponent.VetoableChangeListener
com.sun.java.swing.JList.ListSelectionListener
java.awt.Component.ComponentListener
java.awt.Component.FocusListener
```

10 From the Available events list, select the events that you want to generate and click the downward-pointing arrow.

To move an event from the lower list box to the upper one, select the event and click the upward-pointing arrow.

11 If you want to review or change part of the definition, click Back.

You can go back to any of the previous wizard pages.

12 Click Finish when you're finished with the definition.

The new class is inserted into the active project.

13 Complete the definition of the class or interface in the Source window or the Class Browser's Source pane.

For more information, see "Using the Source window" on page 4-41 or "Working with classes" on page 4-6.

Adding a class

You can add a class at any time by choosing Class from the Insert menu.

You can add a class with the Insert Class Wizard (see "Using the Insert Class Wizard" on page 4-15 for more information). You can also add a class from the Source window (see "Using the Source window" on page 4-41 for more information) or the Project window.

To add a class:

◆ Choose Class from the Insert menu, or right-click and choose Insert Class from the pop-up menu.

You can locate classes in the Source window, Hierarchy Editor, Project window, and Class Browser.

To add a class name by dragging it into source code:

◆ Select the class in the Classes pane, then drag it into the Class Browser's Source pane or a Source window.

  The full class name, full method signature, or data variable is added at the location where you drop it.

To add a class from the Source window:

◆ Type the Java code that creates a class.

  For more information, see "Using the Code Helper" on page 4-49.

To add a class from the Project window:

◆ Add a top-level component to the Project window's Objects view.

  A new class and Java source file are created.

Editing a class

You can edit classes with the Insert Class Wizard, or manually edit them from the Source window, Class Browser, or Hierarchy Editor.

For information on using the Insert Class Wizard, see "Using the Insert Class Wizard" on page 4-15.

For information about editing a class from the Hierarchy Editor, see "Changing a class inheritance" on page 4-37.

To manually edit a class in the Source window:

◆ Find the class and edit the Java code.

To manually edit a class from the Class Browser:

1. Select a class in the Classes pane.
2. Select a member in the Members pane.
3. Edit the class in the Source pane.

Copying or moving a class

You can use menu commands or drag-and-drop to copy or move classes. You can copy and move text within the same window or between the Source window and the Class Browser's Source pane.

To copy or move a class:

◆ Do one of the following:
  - ❖ Select some code, then choose Cut or Copy from the Edit menu, or right-click and choose Cut or Copy. Position the cursor where you want to paste, then choose Paste from the Edit menu, or right-click and choose Paste.
  - ❖ Drag a block of selected text to move it.
  - ❖ Press CTRL while dragging a block of selected text to copy the text. (A + appears over the cursor when a copy operation is being performed.)

Renaming a class

You can rename a class with the Insert Class Wizard (see "Using the Insert Class Wizard" on page 4-15 for more information), or manually rename a class from the Project window, Source window, or Class Browser.

To rename a class in the Project window:

◆ Rename a top-level component in the Project window's Objects view. Doing so changes the class name, constructor, and all references in the source file.

To rename a class in the Source window:

- ◆ Find the class and rename it, then search for its name and rename all occurrences.

To rename a class in the Class Browser

1. Select the class in the Classes pane. Then click the class again.
2. When an edit box appears, type the new name or edit the existing name.

   The class is renamed, including the constructor.

Viewing and editing the source code for a class

You can view and edit a project's source code in the Class Browser's Source pane or in the Source window. Both provide the same editing functions.

To display source code in the Class Browser's Source pane:

1. Select a class in the Classes pane.
2. Select a member in the Members pane.
3. View and edit the class in the Source pane.

To display source code in the Source window:

- ◆ Select a class from the Class Browser's Classes pane, then choose Go to Source from the Classes menu; alternatively, you can right-click a class and choose Go to Source. See "Using the Source window" on page 4-41 for more information.

Deleting a class

You may want to delete a class at some point. Deleting a class removes the source code from the file. This is different from deleting an object (from the Objects view of the Project window); deleting an object removes the associated `.java` file from the project. For more information, see "Deleting a file from a project" on page 3-46.

Note: You cannot delete the default components, only instances of them.

To delete a class:

1 In the Classes pane, select a class.

2 Right-click and select Remove class.

The source code associated with the class is removed from its file.

Finding a class or class definition

You can locate classes in the Source window, Class Browser, Hierarchy Editor, and Project window, and view class definitions.

For more information about searching files, see "Searching one or more files" on page 4-73.

To search from the Source window or pane:

◆ While the Source window is active or while your cursor is in the Source pane of the Class Browser, choose Find from the Search menu to look in the current file.

  or

◆ Choose Find in Files from the Search menu to look in multiple files.

For more information, see "Searching and replacing" on page 4-74.

To locate a class in the Class Browser:

◆ Click in the Classes pane, then type the class name. For example, to locate `java.awt.FlowLayout`, type `flo`. Press TAB to go to the next entry with that letter sequence.

  As you type, selections are made to match the text you enter. As you continue typing, the search is refined.

  The class you are searching for is highlighted.

  Note: In the Class Browser, the search is conducted by package and only expanded packages are searched. So if you want to locate a class in a particular package, you need to expand the package by clicking the +, then start typing. If you want to exclude a package from a search, then collapse the package (click the -).

  If the class you want is not displayed in the pane, you might need to change what classes are displayed by choosing Options from the

Classes options. See “Configuring the Class Browser” on page 4-6 for more information.

To locate a class in the Hierarchy Editor:

◆ While the Hierarchy Editor is the active window, type the class name. For example, to locate `java.awt.FlowLayout`, type `flo`.

As you type, selections are made to match the text you enter. As you continue typing, the search is refined.

The class you are searching for is highlighted.

Note: If the class you want is not displayed, you might have to enable viewing imports. While the Hierarchy Editor is the active window, choose View Imports from the Hierarchy menu (or right-click and choose View Imports) to toggle the display.

To go to a class definition from the Source window or pane:

◆ Perform these steps from a Source window or the Source pane of the Class Browser.

- ❖ Select or click in a class, then choose Go to Definition from the Search menu (or right-click and choose Go to Definition).
- ❖ If a Members window displays, select the member you want to view.

A Class Browser window appears. For more information, see “About classes, members, and the Class Browser” on page 4-1.

To go to a class definition from the Class Browser:

◆ Do one of the following:

- ❖ To display source code in the Source pane, select a class in the Classes pane then a member in the Members pane.

or

- ❖ To display source code in a Source window, select a class from the Classes pane then choose Go to Source from the Classes menu, or right-click a class and choose Go to Source.

For more information, see “About the Source window” on page 4-38.

To go to a class definition from the Hierarchy Editor:

- Do one of the following:
  - To view a class in the Source window, select a class and choose Go to Source from the Hierarchy menu, or right-click a class and choose Go to Source.

  or

  - To view a class in the Class Browser, double-click a class in the Hierarchy Editor.

To go to a class definition from the Project window:

- Do one of the following:
  - Double-click a class in the Packages or Files view.

  or

  - Select a class in the Packages or Files view, then choose Edit Source from the Object menu, or right-click and choose Edit Source.

Working with members

Members are the data and methods in a class. When working with members, here are the tasks you can perform:

- Finding a member
- Adding a member
- Copying or moving a member
- Deleting a member
- Renaming a member
- Viewing a member's source code
- Viewing a member's attributes

For more information, see "Using the Members pane" on page 4-13. For information about editing event methods, see Chapter 9, "Working with Events and Interactions."

Finding a member

You can quickly locate methods (including event handlers) and data variables in the Source window and view their definitions. You can also locate methods or data variables in the Class Browser's Members pane.

To go to a method in the Source window:

1. In the Source Window's Objects drop-down list, choose an object.
2. In the Events/Methods drop-down list, choose the event or method.

   Existing events and methods are shown in bold. If you choose an event or method that's not bold, it's created for you.

   Note: Only top-level components have methods in the Events/ Methods list.

To go to a member definition from the Source window or pane:

Perform these steps from a Source window or the Class Browser's Source pane:

1. Select a member, then choose Go to Definition from the Search menu, or right-click and choose Go to Definition.
2. If a Members window displays, select the member you want to view.

   A Class Browser window appears. For more information, see "About classes, members, and the Class Browser" on page 4-1.

To search from the Source window or Source pane:

Do either of the following:

◆ While the Source window is active or while your cursor is in the Class Browser's Source pane, choose Find from the Search menu to look in the current file.

◆ Choose Find in Files from the Search menu to look in multiple files. For more information, see "Searching one or more files" on page 4-73.

To locate a method or data variable:

1. Select a class in the Class Browser's Classes pane.

Note: If the class you want is not displayed in the pane, you might need to change which classes are displayed. For more information, see "Configuring the Class Browser" on page 4-6

2 Click in the Members pane, then type the method or data variable name. For example, to locate the `CENTER` data variable for the `FlowLayout` class, type `cen`. Press TAB to go to the next entry with that letter sequence.

The method or data variable you're searching for is highlighted. The source code is displayed in the Source pane.

Each member has a color-coded icon to indicate its access privileges (green is public, yellow is protected, blue is package, and red is private).

Note: If the method or data variable you want is not displayed in the source pane, you might need to change which members are displayed, as described earlier in this section.

Adding a member

You can conveniently add members from the Source window or the Class Browser's Members pane. For information about adding event handlers, see Chapter 9, "Working with Events and Interactions."

Right-clicking in the Members pane allows you to add new members to the class. You're presented with a dialog box where you can type the declaration of the method or data. You can also choose from a list of methods that can be overridden. The new members are automatically added to the source code for the class.

To create a member from the Class Browser:

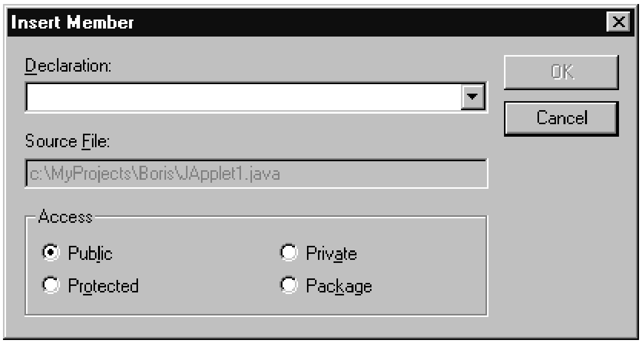
1 In the Classes pane, select the class you want to add a method or data variable to.

For more information, see "Finding a class or class definition" on page 4-25.

2 While the Class Browser is the active window, choose Insert Member from the Insert menu. Alternatively, you can right-click in the Members pane and choose Insert Member.

By default (as set in the Environment Options dialog box) you can also press INSERT. For more information, see “Mapping Visual Cafe commands to key sequences” on page 3-73.

The Insert Member dialog box appears.



Source File: c:\MyProjects\Boris\JApplet1.java

Access: [✓ Public] Private Protected Package

3 Type the declaration and choose the access type, then click OK.

For example, `int myVar()` is a valid declaration you could type. The member appears in the Members pane.

4 Select the member in the Members pane to view and edit source code in the bottom editing window.

Note: If the method or data variable you want is not displayed in the pane, you might need to change which members are displayed, as described in “Using the Members pane” on page 4-13.

To manually add a new member from the Source window:

◆ Type the Java code to add the member.

For more information, see “Using the Code Helper” on page 4-49, “Using the Syntax Checker” on page 4-51, or a Java programming book.

To create or add an event or method from the Source window:

1 Open the Source window for the applet or form.

2 In the Source window’s Objects drop-down list, choose an object.

3 In the Events/Methods drop-down list, choose the event or method.

Existing events and methods are shown in bold. If you choose an event or method that is not bold, it’s created for you.

Note: If you choose an event, Visual Cafe adds the event handler and other code.

For more information about events and interactions, see Chapter 9, "Working with Events and Interactions."

To add a member name by dragging it into source code:

◆ Select the member in the Members pane, then drag it into the Class Browser's Source pane or a Source window.

  The full class name, full method signature, or data variable is added at the location where you drop it.

Copying or moving a member

You can use menu commands or drag-and-drop to copy or move members. You can copy and move text within the same window, or between the Source window and the Class Browser's Source pane.

To copy or move a method or data variable:

◆ Do any of the following:
  - ❖ Select code, then choose Cut or Copy from the Edit menu, or right-click and choose Cut or Copy.
  - ❖ Position the cursor where you want to paste, then choose Paste from the Edit menu, or right-click and choose Paste.
  - ❖ Drag a block of selected text to move it.
  - ❖ Press CTRL while dragging a block of selected text to copy the text. (A + appears over the cursor when a copy operation is being performed.)

Deleting a member

You can delete a method or data variable from the Source window or the Class Browser's Members pane.

Note: Deleting an event handler is the same as deleting an interaction. See "Deleting an interaction" on page 9-17 for more information.

To delete a member from the Source window:

◆ Select the source code for the member, then press DELETE, or choose Delete from the Edit menu.

The member is deleted from the class.

To delete a member from the Class Browser:

1 In the Class Browser's Classes pane, select a class that contains the member.

For more information, see "Finding a class or class definition" on page 4-25.

2 Select a member or SHIFT-click multiple members in the Members pane.

Note: If a method or data variable you want is not displayed in the pane, you might need to change which members are displayed, as described in "Using the Members pane" on page 4-13.

---

3 While the Class Browser is the active window, choose Delete Member from the Classes menu, or right-click in the Members pane and choose Delete Member.

The member is deleted from the class.

Note: You get a deletion confirmation dialog box only if it's enabled in the Environment Options dialog box. For more information, see "Configuring the Class Browser" on page 4-6.

Renaming a member

You can rename members from the Source window or the Class Browser's Members pane (see "Using the Members pane" on page 4-13 for details).

To rename a member in the Source window:

◆ Find the member and rename it, then search for its name and rename all occurrences.

For more information, see "Finding a member" on page 4-28.

To rename a member in the Class Browser:

1. Select a class in the Classes pane, then the member in the Members pane. Click the member again.

   For more information, see “Finding a class or class definition” on page 4-25 or “Finding a member” on page 4-28.

2. When an edit box appears, type the new name or edit the existing name.

Viewing a member’s source code

You can view a member’s source code in the Class Browser’s Source pane or in a Source window (see “Using the Source window” on page 4-41 for more information). The Class Browser’s Source pane has the same editing options as the Source window.

To view a member’s source code from the Class Browser:

◆ Select a class in the Classes pane, then a member in the Members pane.

  The source code appears in the Class Browser’s Source pane.

To view a member’s source code from the Source window:

◆ Select a class from the Class Browser’s Classes pane, then choose Go to Source from the Classes menu. Alternatively, you can right-click a class and choose Go to Source.

  A Source window appears. Here you can edit the member’s Java source code.

Viewing a member’s attributes

You can easily view the attributes of a member from the Class Browser. Also, you can change the access type of selected members, such as whether it’s public, private, or protected.

To view the attributes of a member from the Class Browser:

1. In the Classes pane, select the class that contains the member.

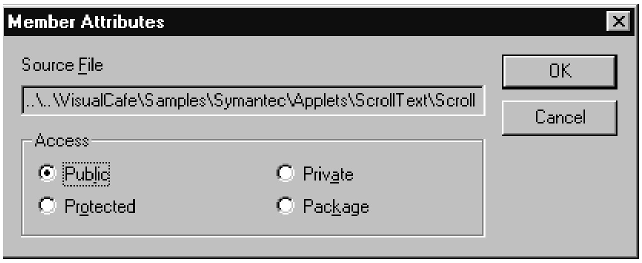
   For more information, see “Finding a class or class definition” on page 4-25.

2 Click on a member in the Members pane.

If a method or data variable you want is not displayed in the pane, you might need to change which members are displayed. For more information, see “Using the Members pane” on page 4-13.

3 Choose Member Attributes from the Classes menu. Or, you can right-click in the Members pane and choose Member Attributes.

The Member Attributes dialog box appears.



Member Attributes

Source File: ..\..\VisualCafe\Samples\Symantec\Applets\ScrollText\Scroll

Access: [✓ Public] Private Protected Package

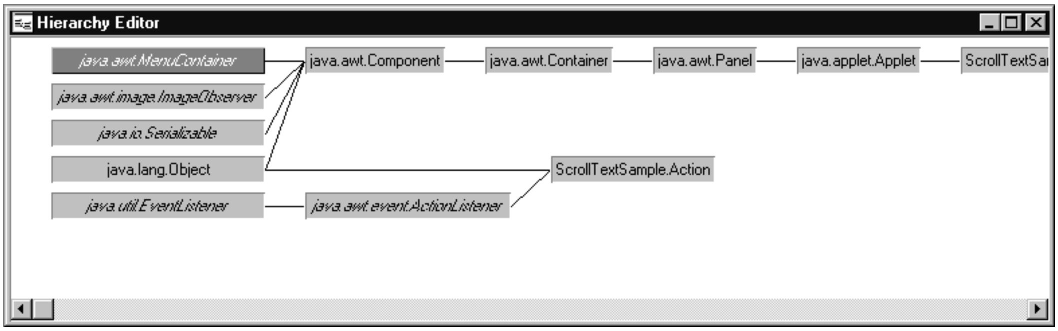
The class declaration is modified and the Members pane’s display is updated to reflect the change.

If you’re editing several members simultaneously and the original access specifiers are not identical, a Don’t Change option displays. This option lets you change member attributes without affecting the original access of each member.

Note: The Class Browser window does not automatically keep member functions’ definitions synchronized with their declarations. If you change a member function’s declaration or definition in the Source pane, you must manually update the corresponding declaration or definition to match.

About the Hierarchy Editor

The **Hierarchy Editor** is a tool that provides you with a visual representation of the classes in your project and their inheritance relationships. You can optionally show imports as well.



All Java programs have a hierarchical structure. You can use the Hierarchy Editor to directly manipulate the class relationships of your projects by dragging and dropping from one class to another.

Note: Visual Cafe does not let you change the structure of the existing preparsed class information, such as information that is associated with the JDK and JFC source code.

You can change inheritance by clicking the line between a parent and base class, then dragging the line by its anchor to another base class.

Double-clicking a class opens the Class Browser on the selection.

You can view all classes used by your project by right-clicking and choosing View Imports. You can also view class data and methods using the Hierarchy Editor's Member and Source windows.

You can extend existing classes by clicking and dragging. You can also create a new parent-child relationship by clicking a class and dragging to the desired parent class. If you double-click a class, that class is displayed in the Class Browser so you can view its data and methods.

Caution: When you make changes to classes or inheritance relationships, the relationships are automatically changed in the underlying source code and in all open windows in the Visual Cafe environment. For that reason, you should be careful when making changes with the Hierarchy Editor.

To display the Hierarchy Editor:

- ◆ While the project you want to view is active, choose Hierarchy Editor from the View menu.

Using the Hierarchy Editor

When using the Hierarchy Editor, you can perform the following tasks:

- ◆ Viewing imports
- ◆ Locating a class in the Hierarchy Editor
- ◆ Changing a class inheritance
- ◆ Changing class attributes

Each of these topics is discussed in this section.

You can configure the Hierarchy Editor from the Editing page in the Environment Options dialog box. For more information, see “Configuring the Class Browser and Hierarchy Editor” on page 4-9.

Viewing imports

You can choose to view the import hierarchy in the Hierarchy Editor, or opt to hide it from view.

To enable and disable viewing imports:

- ◆ While the Hierarchy Editor is the active window, choose View Imports from the Hierarchy menu (or right-click and choose View Imports) to toggle the display.

Locating a class in the Hierarchy Editor

You can quickly locate a particular class in the Hierarchy Editor by typing part of the class name.

To locate a class:

◆ While the Hierarchy Editor is the active window, type the class name. For example, to locate `java.awt.FlowLayout`, type `flo`.

  As you type, selections are made to match the text you enter. As you continue typing, the search is refined.

  The class you're searching for is highlighted.

Changing a class inheritance

You can quickly change or delete the inheritance relationship between a class and its parent in the Hierarchy Editor and the Class Browser.

To change the inheritance hierarchy:

◆ Click the line between two classes, then drag the line by its anchor to another class.

  The class on the right now derives from the class you connected it to. This changes the class's source code.

To remove an inheritance using the Hierarchy Editor:

1. Select the line that links a class and the class it derives from.
2. Right-click the window to display the pop-up menu and choose Remove Inheritance.

   The class now extends directly from `java.lang.Object.`

To delete a class inheritance using the Class Browser:

1. In the Classes pane, choose Options from the Classes menu, or right-click and choose Options.
2. In the Group/Sort tab, specify a Class Grouping of Hierarchically, then click OK.

When you're showing classes in a Hierarchical view and a class implements one or more interfaces, it displays below each interface.

3 In the Classes pane, select a class then press DELETE.

If you delete a class that's specified below an interface, that interface is deleted from the class. If you delete a class below a class, the class now inherits directly from `java.lang.Object`.

To view a class in a Source window:

Select a class, then choose Go to Source from the Hierarchy menu, or right-click a class and choose Go to Source. See "Using the Source window" on page 4-41 for more information.

Changing class attributes

You can change a class's name and its base class.

To change the attributes for the selected class:

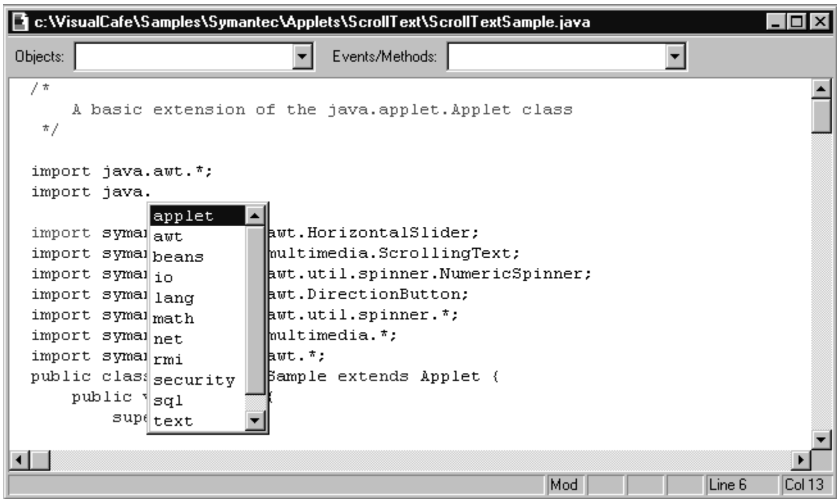
1 Choose Class Attributes from the Classes or Hierarchy menu.

The Class Attributes dialog box appears, which opens the Insert Class Wizard.

2 Use the Insert Class Wizard to edit the class. For more information, see "Using the Insert Class Wizard" on page 4-15.

About the Source window

Visual Cafe's Source window lets you create, examine, and modify your project's source files. Because these files are standard text files, you can, in principle, use any source editor to work with them — but Visual Cafe's Source window is designed to work in concert with other Visual Cafe tools.



```
/*
    A basic extension of the java.applet.Applet class
 */

import java.awt.*;
import java.

import syma      awt.HorizontalSlider;
import syma      multimedia.ScrollingText;
import syma      awt.util.spinner.NumericSpinner;
import syma      awt.DirectionButton;
import syma      awt.util.spinner.*;
import syma      multimedia.*;
import syma      awt.*;
public clas      Sample extends Applet {
    public v          {
        supe
```

The Source window includes a full-featured text editor with many features that are particularly useful when editing Java source code, such as full Java syntax highlighting and flexible navigation tools. The Source window supplies standard Windows functions for cutting, copying, pasting, and deleting text. It also includes helpful editing features such as checking delimiters and automatically indenting or unindenting after braces, as well as providing contextual programming help with the Code Helper and Syntax Checker.

Note: The Source window and the Class Browser's Source pane share the same editing functionality.

The Source window also plays an important role in debugging your project; you can use the Source window for monitoring program execution while debugging, and for setting breakpoints at design time (for more information, see Chapter 6, "Debugging Your Program").

Visual Cafe automatically saves all files open in Source windows when you rebuild your project. During compilation, error messages are displayed in the Messages window; when you double-click on an error message, Visual Cafe opens a Source window on the corresponding source file, if necessary, then jumps to the line in the source code that caused the error.

The Source window can display keywords and comments in special font styles and colors. This technique helps you track errors in source code while you're editing. For example, an unmatched comment (`/*` without a matching `*/`) turns a large part of the code a different color, making it obvious where the problem lies. Also, keywords are easier to spot when they're in a different color or font style. Misspelled keywords can be caught immediately when they remain displayed in the default font. By default, comments are green, Java language keywords are blue, and standard code text is black.

Source windows are tied to individual components and class files. All methods in a single component are displayed in one Source window. The active component and a complete list of its events are displayed at the top of the window. If the event has not yet been bound to an event handler, selecting an event name creates an event handler for that event. Events that have been bound to an event handler appear in bold text. When you select a bound event, the event handler associated with that event name is displayed in the Source window's text box.

In the Events/Methods drop-down box in the Source window, Visual Cafe displays all events that are associated with the active component's class. Events that have code associated with them appear in boldface text. You can select an event handler from this list to edit or create it, and the associated code displays in the window. For more information, see Chapter 9, "Working with Events and Interactions."

To display the Source window:

◆ Do one of the following:

- ❖ Select a file in the Project window, then choose Edit Source from the Object menu, or right-click and choose Edit Source.
- ❖ Double-click a file in the Packages or Files view of the Project window. (If Symantec Visual Page is not installed, double-clicking an HTML file in the Objects view opens a Source window.)
- ❖ While the Form Designer is the active window, choose Edit Source from the Object menu, or right-click and choose Edit Source.
- ❖ Double-click in the Form Designer.
- ❖ In the Classes pane of the Class Browser, select a class and choose Go to Source from the Classes menu, or right-click a class and choose Go to Source.

- ❖ In the Hierarchy Editor, select a class and choose Go to Source from the Hierarchy menu, or right-click a class and choose Edit Source.
- ❖ Open a file by choosing Open from the File menu.

Using the Source window

You can use the Source window in conjunction with the following tasks:

- ◆ Editing a source file
- ◆ Showing horizontal scroll bars
- ◆ Typing in the Source window
- ◆ Enabling and disabling RAD and automatic code generation
- ◆ Printing a source code file
- ◆ Adding custom code to a source file
- ◆ Using the Code Helper
- ◆ Using the Syntax Checker
- ◆ Correcting syntax errors

These topics are discussed in this section.

Editing a source file

You can use the Source window to edit your code manually. To help you edit your source code, Visual Cafe provides two helpful tools, the Code Helper and the Syntax checker. These tools are described later in this chapter.

To edit a currently active file in the Source window:

- ◆ Choose Edit from the Source menu. Visual Cafe moves that project to the foreground and opens an editing window for the file.

To open a source file that is not currently active:

- ◆ Click the source file in the Project window, then choose Edit Source from the Object menu.

The Source window provides standard Windows functions for cutting, copying, pasting, and deleting text. You can aceess these commands from the Edit menu, or by right-clicking and selecting them from the context menu.

To cut, copy, or paste source code:

- ◆ Do one of the following:
  - ❖ Select code, then choose either Cut or Copy from the Edit menu, or right-click and choose either Cut or Copy.
  - ❖ Position the cursor where you want to paste code, then choose Paste from the Edit menu, or right-click and choose Paste.
  - ❖ Drag a block of selected text to move it.
  - ❖ Press CTRL while dragging a block of selected text to copy the text. A plus sign (+) appears over the cursor when a copy operation is being performed.

To select and move a block of code:

- ◆ You can select a block of text in any of the following ways:
  - ❖ Clicking and dragging the mouse
  - ❖ SHIFT-clicking
  - ❖ ALT-clicking. The ALT-click drag performs a column select.

  By clicking and dragging a block of selected text, you can reposition the text anywhere in the current file.

If you wish, you can copy a block of text instead of moving it.

Showing horizontal scroll bars

You can set Visual Cafe to display horizontal scroll bars at the bottom of Source windows and panes.

To show horizontal scroll bars:

1. From the Tools menu, choose Environment Options, then click the Editing tab.
2. Select Show horizontal scroll bars to display a scroll bar at the bottom of Source windows and panes. This option is selected by default.
3. Click OK.

   The changes take effect immediately.

Typing in the Source window

While you're typing in the Source window, you can perform the following tasks:

- Toggling typing modes
- Controlling the cursor style
- Getting help on a Java keyword or method
- Changing code spacing and text capitalization

These topics are discussed in this section.

Toggling typing modes

The Source window supports two typing modes: *overtype*, which replaces characters as you type, and *insert*, which adds new characters to the file.

To toggle the typing mode:

- Press the INSERT key to toggle between overtype and insert modes.

  The typing mode is displayed in the Source window's status bar.

Note: A change in typing mode applies to all open Source windows, not just the active Source window.

Controlling the cursor style

In a Source window or pane, you can toggle between Insert mode and Overwrite mode by pressing the INSERT key. In Overwrite mode, the new characters overwrite the existing text. The Editing tab of the Environment Options dialog box lets you specify the appearance of the Insert and Overwrite modes.

The Insert and Overwrite groups offer you options that govern the cursor style in each mode.

Option	Description
Block	Covers the current character.
Underline	Underlines the current character.
Vertical bar	The standard insertion point cursor; displays between characters.
Blink	When selected, the cursor blinks. You can set the blink rate in the Windows Control Panel.

To set the cursor style:

1. From the Tools menu, choose Environment Options, then click the Editing tab.
2. For the Insert and Overwrite modes, select either Block, Underline, or Vertical Bar to specify the cursor style. You can also choose Blink to specify a blinking cursor.
3. Click OK.

   The changes take effect immediately.

Getting help on a Java keyword or method

If you want to know more about a specific Java keyword or method, you can easily find information from the online *Java Reference Help*.

To get help on a Java keyword or method:

◆ Position the cursor on a keyword or method, then press F1.

A list of associated topics displays in the Java Reference Help.

Changing code spacing and text capitalization

Visual Cafe lets you set formatting options for the source file you're working on. You can change code indentation, convert tabs to spaces or vice versa, and change portions of code to uppercase or lowercase letters.

To indent or unindent code:

- Select the lines of code, then choose either Indent or Unindent from the Source window. You can also press TAB to indent code, or SHIFT-TAB to unindent code.

  The code is moved one tab position.

  You can specify indentation options for a file, or for all projects. See “Setting text formatting for a single file” on page 4-53 and “Setting text formatting for the Visual Cafe environment” on page 4-54 for more information.

To convert tabs to spaces or spaces to tabs:

- Select the code, then choose either Tabs to Spaces or Spaces to Tabs from the Source menu.

  All tab characters in the selected text become spaces, or spaces become tabs. The number of spaces for each tab character depends on the Tab width value for the format options.

  You can also set formatting options for all source files in the Environment Options dialog box. For more information, see “Setting text formatting for the Visual Cafe environment” on page 4-54.

To change text to all uppercase or all lowercase letters:

- Select the code, then choose either Uppercase or Lowercase from the Source menu.

Enabling and disabling RAD and automatic code generation

You can turn off the visual environment for a file, and the automatic code generation that occurs, by translating the visual environment into source code. **Rapid Application Development**, or RAD, refers to the process of creating your programs in a visual development environment. You can specify in your project options whether new files you add to the project have RAD enabled or disabled. You can also turn RAD on and off for an individual file.

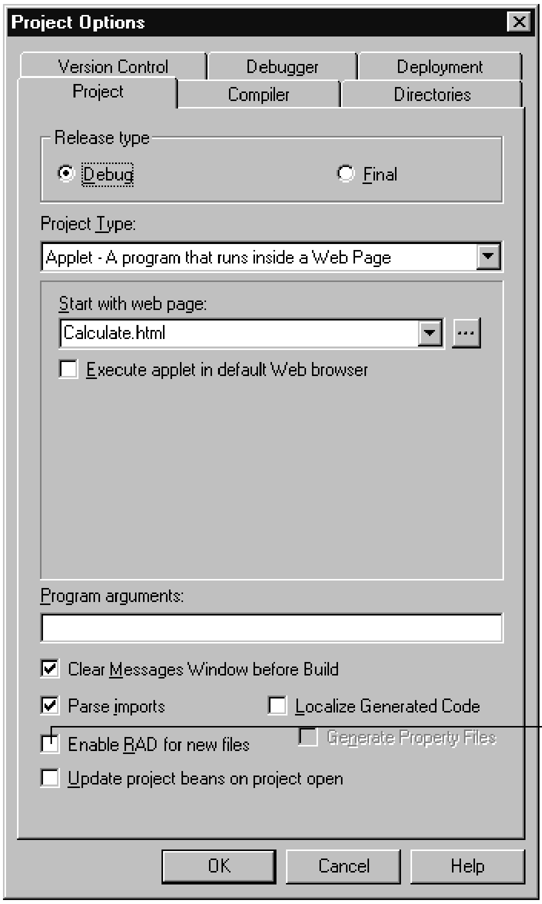
By default, RAD is turned on. This means that components appear in the Objects view of the Project window, that you can design your forms in the Form Designer and Menu Designer, and that you can use the Interaction Wizard to specify interactions. You might want to turn off RAD to use less computer resources and make Visual Cafe run faster. When RAD is turned off, components disappear from the Objects view and you cannot use the Form Designer with the file.

Caution: If RAD is off and you turn it on, Visual Cafe might change your code in order to parse it into its visual environment.

To enable or disable RAD for new files:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box displays.
3. Click the Project tab:



Release type: [✓ Debug] Final

Project Type: Applet - A program that runs inside a Web Page

Start with web page: Calculate.html

☐ Execute applet in default Web browser

Program arguments:

☑ Clear Messages Window before Build

☑ Parse imports ☐ Localize Generated Code

☐ Enable RAD for new files ☐ Generate Property Files

☐ Update project beans on project open

Click here to enable or disable RAD for new files

4 Select Enable RAD for new files to specify that new files have RAD enabled. Or deselect it to disable RAD for new files.

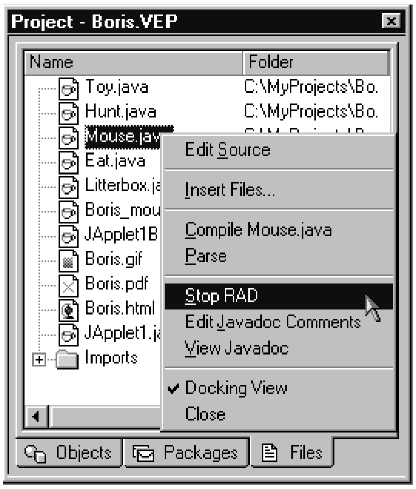
5 Click OK.

The change takes effect immediately for new files.

To enable or disable RAD for an existing file in a project:

1 In the Project window, click the Files tab.

2 Right-click a file, then choose Stop RAD or Start RAD.



To enable or disable RAD when saving a source file:

1 While in the Source window, choose Save As from the File menu.

The Save As dialog box displays.

2 Select or deselect Enable RAD, as you wish.

Printing a source code file

You can easily print a source code file, just as you would in any word-processing application.

To print the file displayed in the Source window:

◆ Choose Print from the File menu.

Adding custom code to a source file

Although you can use Visual Cafe to create applets and applications that don't require any custom Java code, you may want to add your own code to advanced applets and applications. You can add custom code to your source file from the Source window or the Class Browser's Source pane.

You can add custom code for error handling, event control, and complex component relationships and behavior.

In addition, you can bind event handlers to any component and to menu commands.

Caution: Custom code, interactions, and event bindings are not deleted from the source file when you delete a component. You must manually maintain any file that contains custom code.

As described in "Enabling and disabling RAD and automatic code generation" on page 4-45, you can use Visual Cafe's visual environment to quickly create form layouts. If you don't want to use the visual environment, or if you're finished using the visual environment and want to stop Visual Cafe from automatically generating Java code, you can turn off the visual environment for a single file or for all new files that are added to a project.

If you're using the visual environment, you should create your project and use Visual Cafe's visual tools to create your forms before you add any custom code. You can add components to your project, arrange components in the Form Designer, design menus in the Menu Designer, modify the look of individual components with the Property List, and add interactions between components or within a component with the Interaction Wizard.

When you're satisfied with how your project forms are laid out, you can then add code enhancements using the Source window or the Class Browser's Source pane.

Guidelines and warnings

◆ Whenever possible, make object changes using Visual Cafe, rather than adding code directly to the source file. For example, you can add components, classes, or class members, or change component properties by using the Component Palette, a menu selection, or the Property List window.

- Do not add or modify custom code within code blocks that are regenerated. Visual Cafe places special comments in the source code to indicate the beginning and end of blocks of code that it manages. These code blocks start with `//{{` and end with `//}}`. For example, if you add a button to a form, Visual Cafe generates the following code:

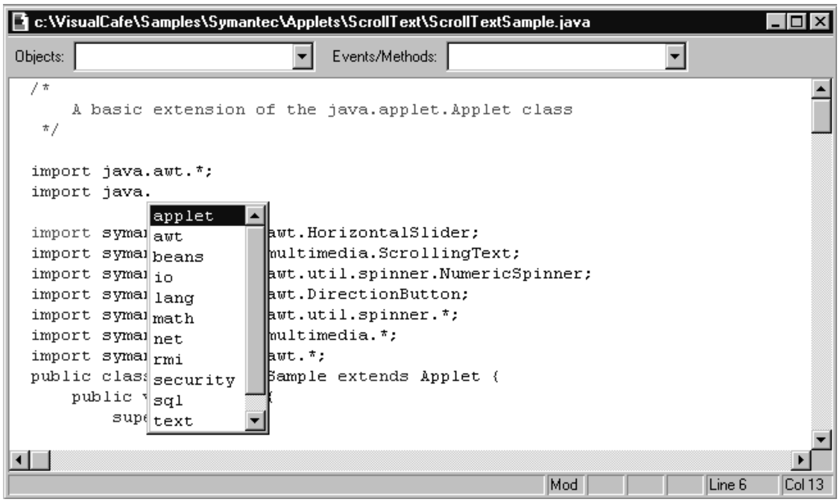
```
//{{DECLARE_CONTROLS
java.awt.Button button1;
//}}

//{{INIT_CONTROLS
button1 = new java.awt.Button("button");
button1.reshape(63,77,88,30);
add(button1);
//}}
```

To modify the code within these tags, you should try to use code syntax that matches what Visual Cafe can generate. You should realize that in some cases Visual Cafe might be unable to back-parse your Java file into its visual environment.

You should try not to move this type of code block. But if you do, be sure to move the entire code block, including its comment tags.

Using the Code Helper

To assist you in writing Java code, Visual Cafe provides a contextual helper called the **Code Helper,** which interprets the current context and provides either a list of the methods in a given class, a list of the different versions of a particular method, or a list of class objects that begin with a particular character sequence. The Code Helper also helps you enter Javadoc tags.



```
/*
    A basic extension of the java.applet.Applet class
 */

import java.awt.*;
import java.

import syma       awt.HorizontalSlider;
import syma       multimedia.ScrollingText;
import syma       awt.util.spinner.NumericSpinner;
import syma       awt.DirectionButton;
import syma       awt.util.spinner.*;
import syma       multimedia.*;
import syma       awt.*;
public clas       Sample extends Applet {
    public        {
        supe
```

To activate the Code Helper:

1. Choose Environment Options from the Tools menu, then click the Editing tab.
2. To activate the Code Helper, select Automatically Show Code Helper. (Deselect this option to deactivate the Code Helper.)

   If you want the Code Helper to automatically enter the suggestions it makes, select Enter code helper selection automatically. This option is deselected by default.
3. Click Apply or OK to save the setting.

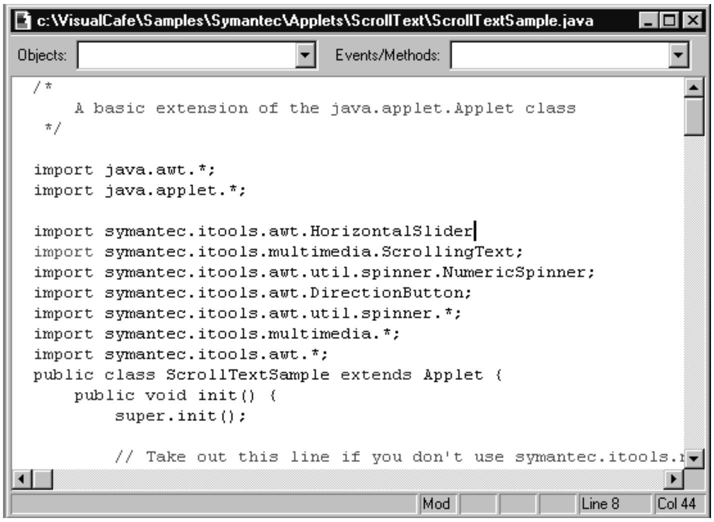
To use the Code Helper:

◆ As you type in the Source window, a drop-down list appears and you can choose items from the list.

You can use the Code Helper on an as-needed basis by pressing CTRL-J.

Using the Syntax Checker

Visual Cafe's **Syntax Checker** lets you identify problems that the compiler would find, but alerts you while you're in the process of writing code. This feature can save you time and assist with troubleshooting.


c:\VisualCafe\Samples\Symantec\Applets\ScrollText\ScrollTextSample.java
Objects:
Events/Methods:
Mod
Line 8
Col 44

```
/*
    A basic extension of the java.applet.Applet class
 */

import java.awt.*;
import java.applet.*;

import symantec.itools.awt.HorizontalSlider
import symantec.itools.multimedia.ScrollingText;
import symantec.itools.awt.util.spinner.NumericSpinner;
import symantec.itools.awt.DirectionButton;
import symantec.itools.awt.util.spinner.*;
import symantec.itools.multimedia.*;
import symantec.itools.awt.*;
public class ScrollTextSample extends Applet {
    public void init() {
        super.init();

        // Take out this line if you don't use symantec.itools.
```

To activate or deactivate the Syntax Checker:

1. Choose Environment Options from the Tools menu, then click the Editing tab.
2. To activate the Syntax Checker, select Highlight Syntax Errors While Editing. Otherwise, deselect it.
3. If you want the Syntax Checker to display more errors (including some complex errors), select Show Full Compiler Errors.

   With this option deselected, only parse errors are given. The Syntax Checker will show errors for issues such as missing semicolons, unmatched parentheses, unterminated strings, and malformed expressions. With the option selected, a full compile occurs. The Syntax Checker will find errors for additional issues such as classes not found in imports, invalid types for expressions, and unknown methods.
4. Click Apply or OK to save your settings.

To use the Syntax Checker:

- ◆ As you type in the Source window, invalid items are highlighted in the error color, and a yellow tool tip displays the error.

Tip: To view or set your error color, choose Environment Options from the Tools menu, click the Display tab, then select Errors in the Color and Style list.

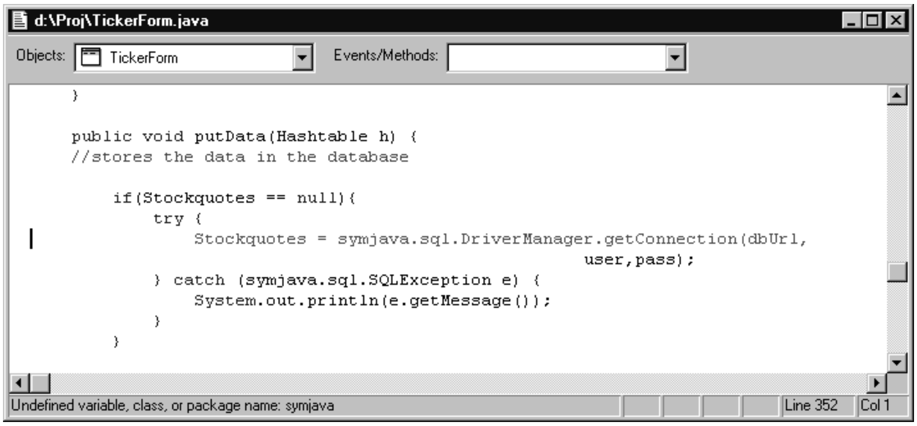
Correcting syntax errors

If your source code contains syntax errors, Visual Cafe flags them in the Messages window after a compile. (See the previous section for information on catching syntax errors as you write code.) You can go directly to each error from the Messages window.

To go to a syntax error from the Messages window:

1. From the View menu, choose Messages to bring the Messages window to the front.
2. Double-click on any error message to go to that error.

   The file containing the error opens in a Source window at the offending line. Once the file opens, you can edit your source code.



```
    }

    public void putData(Hashtable h) {
    //stores the data in the database

        if(Stockquotes == null){
            try {
                Stockquotes = symjava.sql.DriverManager.getConnection(dbUrl,
                                                      user,pass);
            } catch (symjava.sql.SQLException e) {
                System.out.println(e.getMessage());
            }
        }
```

Undefined variable, class, or package name: symjava   Line 352   Col 1

.

If you're using incremental debugging (see "Using incremental debugging" on page 6-40 for details), compile-time errors are reported in the Messages window when you change your code.

Setting text formatting for a single file

By default, the Source window automatically indents a new line to the same depth as the previous line. You can set several indentation options, such as automatic indentation, tab width, and indent/unindent after braces, from the Format Options dialog box. These options apply to a single file.

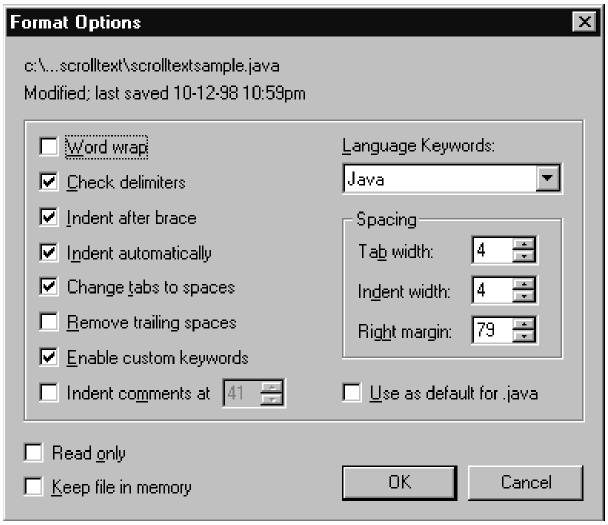
Because the Source window is designed to display source files, word wrap is not enabled by default. When you're typing, you must press the ENTER key to start a new line. When you type past the right edge of the window, the text scrolls horizontally. You can enable word wrap and set a right margin either for a particular file or for the entire Visual Cafe environment using Visual Cafe's text formatting options.

To set text formatting options for the Visual Cafe environment, see the following section, "Setting text formatting for the Visual Cafe environment" described on page 4-54.

To set format options for the Source window of a file:

1 Choose Format Options from the Source menu.

The Format Options dialog box appears.



c:\...scrolltext\scrolltextsample.java
Modified; last saved 10-12-98 10:59pm

- [ ] Word wrap
- [x] Check delimiters
- [x] Indent after brace
- [x] Indent automatically
- [x] Change tabs to spaces
- [ ] Remove trailing spaces
- [x] Enable custom keywords
- [ ] Indent comments at 41

Language Keywords: Java

Spacing
Tab width: 4
Indent width: 4
Right margin: 79

- [ ] Use as default for .java
- [ ] Read only
- [ ] Keep file in memory

2 Set the text formatting options you want to use.

The formatting style options in this dialog box are the same as those in the Format tab of the Environment Options dialog box but apply only to the current file. See “Setting format options for files with a certain extension” on page 4-55.

3 Select the remaining options as needed. These options are described in the following table:

Option	**Description**
Use as default for .java	Assign the current settings as the default setting.
Read only	Sets the current file so that it can’t be edited.
Keep file in memory	No prompts are given for saving on close or changes. Permanently stores file in memory.

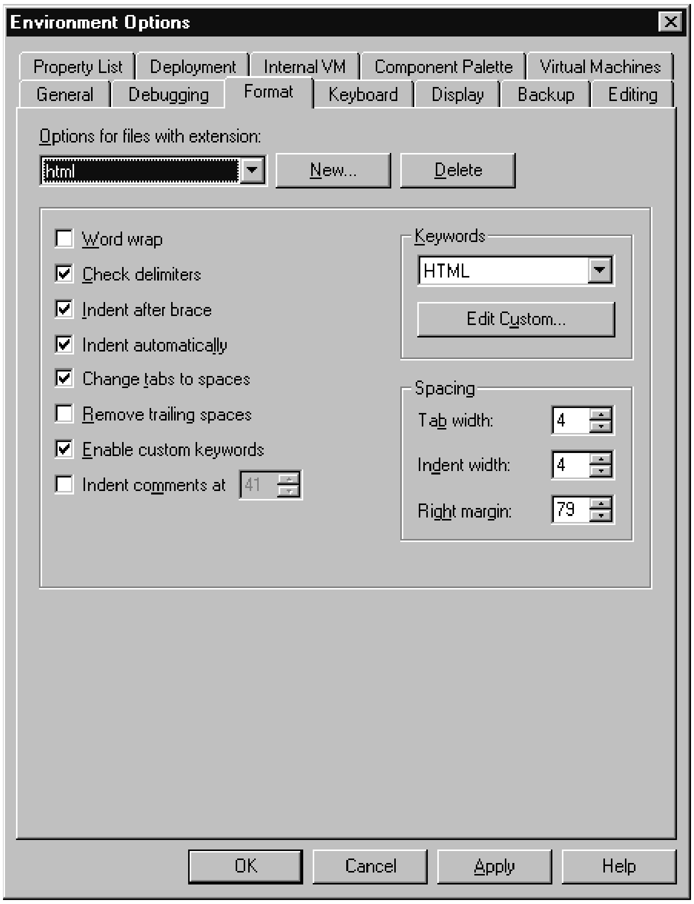
4 Click OK.

The changes take effect immediately.

Setting text formatting for the Visual Cafe environment

You can define a formatting style for different file types displayed in the Visual Cafe windows. The initial file types that are available are Java and HTML.

To open the Format view of the Environment Options dialog box, from the Tools menu, choose Environment Options, then click the Format tab.


Environment Options

Options for files with extension: html

- [ ] Word wrap
- [x] Check delimiters
- [x] Indent after brace
- [x] Indent automatically
- [x] Change tabs to spaces
- [ ] Remove trailing spaces
- [x] Enable custom keywords
- [ ] Indent comments at 41

Keywords: HTML

Spacing
Tab width: 4
Indent width: 4
Right margin: 79

To set text formatting in Visual Cafe, you can do the following:

- Modify extension file types for formatting
- Set format options for files with a certain extension
- Specify custom keywords

Setting format options for files with a certain extension

You can specify how text is formatted in different kinds of files, as determined by their file type extension (`*.java`, `*.html`, and so forth).

You can specify a different combination of formatting options for each kind of file.

To set format options for files with a certain extension:

1. From the Tools menu, choose Environment Options, then click the Format tab.
2. Choose the file extension you want to customize.
   - ❖ To create a new file extension entry, click New and enter the extension in the dialog box.
   - ❖ To remove an extension from the list, choose the extension, and click Delete.

   <Untitled> is the same type you get after choosing New File from the File menu.

   <Unknown> is a file extension that does not have specific format options set for it. This is the type of file that you get when you save a file as an "unknown" type.
3. Set the options you want to apply to files with the designated extension. To highlight language keywords, choose the language from the Keywords drop-down list.

Option	Description
Word wrap	Enables word wrap. While you're typing, lines that extend beyond the right margin are automatically broken at the last word boundary before the margin. By default, this option is not selected.
Check delimiters	If you type a right parenthesis), square bracket] or brace }, the editor briefly highlights the corresponding left delimiter. If no matching delimiter is found, an error message displays in the status bar. By default, this option is selected.
Indent after brace	If the last character you type on a line is a left brace, the next line is automatically indented by an extra tab stop. This option only works if Indent Automatically is selected. By default, this option is selected.

Option	Description
Indent automatically	Automatically indents the first character of a new line when you press ENTER. Use this option to specify indented paragraphs. By default, this option is selected.
Change tabs to spaces	Tabs are inserted into the text as an appropriate number of spaces, rather than as tab characters. By default, this option is selected.
Remove trailing spaces	Trailing spaces and tabs are removed from the end of each line when a file is saved. By default, this option is not selected.
Enable custom keywords	You can maintain a set of custom keywords that are highlighted in a specified manner within the Source window. There is one custom keywords list that applies to all file types that have the Enable custom keywords option. Select a keyword type in the Keywords list, or click Edit Custom to specify custom keywords. To set highlighting for custom keywords, click the Display tab. See “Specifying custom keyword formatting” on page 4-58. By default, this option is selected.
Indent comments at	The Source window automatically indents the comment to a specified column when you type `//` or `/*` to start a comment. You can specify the alignment column in the adjacent text box. By default, this option is not selected.
Tab width	Specifies the number of columns between tab stops (1–16). The default is four character widths. This value may be overridden locally in the format options for a file.
Indent width	Specifies the number of columns for each indent. The default is four character widths.

Option	Description
Right margin	Specifies the column that acts as the right margin for word wrapping (1–512). This value may be overridden for an individual file. The default is 79. This value may be overridden locally in the format options for a file.

Modifying extension file types for formatting

You can change which extension file types are available by adding or removing entries. Each file type has its own set of formatting options that you can specify.

To add file extension types for formatting:

1. From the Tools menu, choose Environment Options, then click the Format tab.
2. Click New.

   The Enter New File Extension dialog box displays.
3. Enter the file extension you want to add to the format type list, and click OK.

   When the extension is available from the list, you can define a format style for the file type.

To remove file extension types for formatting:

1. From the Tools menu, choose Environment Options, then click the Format tab.
2. From the drop-down list, select the file type you wish to remove.
3. Click Delete.

   The file extension type is removed from the list.

Specifying custom keyword formatting

Custom keywords are recognized in the Source window or pane. They are highlighted in red by default. You can change the display attributes of custom keywords in the Environment Options Format tab, and you can add or remove entries to the custom keyword list. These keywords are recognized in the Source window and pane. There is one custom keywords list that applies to all file types that have the Enable custom keywords option.

To specify custom keyword formatting:

1. From the Tools menu, choose Environment Options, then click the Format tab.
2. Click Edit Custom.

   The Custom Keywords dialog box displays.
3. Add or remove keywords, as you wish.
   - ❖ To add a new keyword, type the keyword into the text box and click Add.
   - ❖ To remove a keyword from the list, click the keyword in the list and click Remove.
4. Click OK.

   You can now specify how these custom keywords will display. See "Setting format options for files with a certain extension" on page 4-55.

About Javadoc

You can document your source code by including special documentation comments called **Javadoc comments**. You can include these comments before each class or interface declaration and before each method, constructor, or field declaration. Use **Javadoc tags** to describe the author and version of a class, as well as a method's parameters, what a method returns, any related classes, and if a method throws any exceptions. There is also a special tag that you can use to indicate a deprecated, or obsolete, class or member, so that the Java compiler can issue warnings when the class or member is used.

Once you've included your Javadoc comments, Visual Cafe can then scan the source code files and automatically generate these comments into HTML files. You can enhance the formatting of these HTML files by including HTML code in the Javadoc comments. For more information about using HTML in Visual Cafe, see "About HTML files in Visual Cafe" on page 3-50.

A Javadoc comment looks like this:

```
/**
 * This is the description part
 *
 * @tag      comment for the tag
 */
```

A Javadoc tag appears within a Javadoc comment and is of the form @*name* where *name* can be one of the following: `author`, `version`, `param`, `return`, `exception`, `since` or `see`.

For more information about how to style your Javadoc comments, see the Javadoc specification at Sun Microsystems' Web site at `http://java.sun.com.` The Javadoc utility in Visual Cafe is equivalent to Javadoc 1.1 by Sun Microsystems.

About Javadoc output

Javadoc generates the following files:

File	Description
`packages.html`	Lists the packages in the documentation set. Also the main point of entry into the documentation.
`tree.html`	Contains the class hierarchy of all the classes in the documentation set.
`AllNames.html`	Contains the complete index of all fields or members.
	If you specify the `-splitindex` option, `AllNames.html` contains just those fields and members that begin with the letter A. The other index files are written to `index-`*?*`.html` where *?* is B to Z. `Index-Other.html` is also written, which contains those fields or members with a leading underscore (_).
`Package-`*name*`.html`	*name* is the name of the package. Lists classes in a package.

Using Javadoc

You can instruct Visual Cafe to produce Javadoc documentation when your code is compiled. In addition, you can specify where Java API documentation is kept. You can create documentation every time you compile, or by selecting one or more Java files and choosing Produce Javadoc from the Project menu.

Use the Javadoc Editor to quickly enter Javadoc comments into your source code. Visual Cafe also provides a Javadoc Viewer, which you can use to quickly locate documentation for files, packages, JavaBeans in the Component Library, classes and methods in the Source Editor, and more.

Note: Remember that if an HTML file has links to other files, you need to keep the files in the same relative locations for the links to work. The graphics files must also be kept in the same relative location.

To produce Javadoc documentation for a project:

1. Activate the Project window of the project you want to work with.
2. Choose Produce Javadoc from the Project menu.

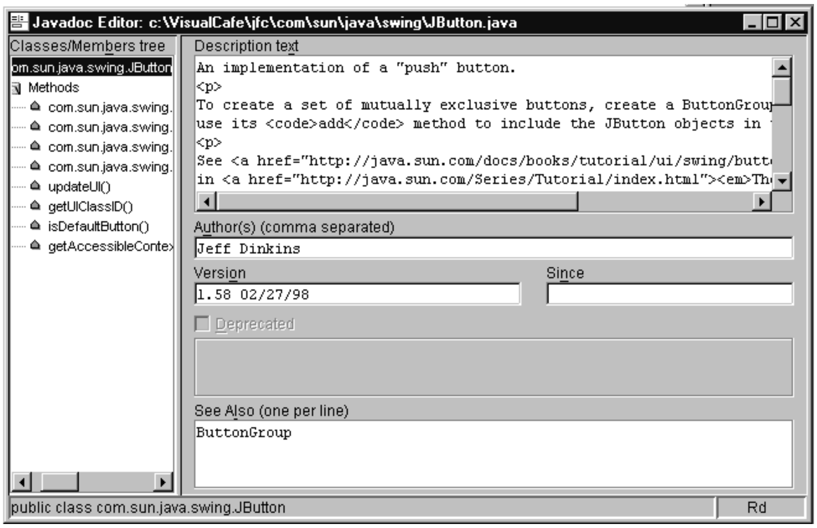
   Your Javadoc options determine what documentation is generated and where. For more information, see “Specifying Javadoc folders” on page 4-69 and “Setting Javadoc options” on page 4-70.

Using the Javadoc Editor

While working in the Source window, you can use the Javadoc Editor to quickly add Javadoc tags that document your code. This documentation will appear in HTML files when you generate Javadoc for the file.

Note: You can have only one Javadoc Editor open at a time. If you open the Javadoc Editor for another file, the editor switches to that file.

Here's what the Javadoc Editor looks like:



Description text

```
An implementation of a "push" button.
<p>
To create a set of mutually exclusive buttons, create a ButtonGrou
use its <code>add</code> method to include the JButton objects in
<p>
See <a href="http://java.sun.com/docs/books/tutorial/ui/swing/butt
in <a href="http://java.sun.com/Series/Tutorial/index.html"><em>Th
```

Author(s) (comma separated): Jeff Dinkins

Version: 1.58 02/27/98

Since:

Deprecated

See Also (one per line): ButtonGroup

public class com.sun.java.swing.JButton   Rd

You can add or view Javadoc comments for a class, method, or data item. If you select a specific member in the source code, when you invoke the Javadoc Editor it will display the Javadoc comments for that member by default.

Note: You cannot add Javadoc comments to a read-only file, although you can still browse the file in the Javadoc Viewer.

To add or view Javadoc comments:

1. Activate the Source window or the Class Browser's Source pane for the file you want to create Javadoc comments for. Or, if you've used the Javadoc Editor at least once, in the Files view of the Project window, right-click a file and choose Edit Javadoc Comments.

   Note: You can have only one Javadoc Editor open at a time. If you open the Javadoc Editor for another file, the editor switches to that file.

   ---

2. Choose Edit Javadoc Comments from the Source menu.

   The Javadoc Editor appears.

After you've already displayed the Javadoc Editor once, or if the file already has Javadoc comments, you can right-click and choose Edit Javadoc Comments.

3 In the left pane, choose a member from the tree.

In the tree, click the plus sign (+) to expand an item or the minus sign (-) to collapse it. You can also navigate using the Up, Down, Left, and Right Arrow keys, and expand or collapse a branch by pressing Enter or double-clicking.

The full declaration of the member appears in the status bar at the bottom of the window. Make sure the item you selected appears in the status bar.

The display changes, based on the member you selected.

Tip: You can adjust the size of the panels if needed by placing the cursor over the center divider and then dragging. You can also resize the window by dragging a side or corner.

4 In the right pane, type your Javadoc comments in the fields.

Note: You cannot enter Javadoc comments for the Classes, Methods, and Data folders, but you can enter comments for the items they contain.

Although you can enter Javadoc comments for all members, your Javadoc settings determine what HTML files are generated when you choose Produce Javadoc from the Project menu. For more information, see "Setting Javadoc options" on page 4-70.

Note: You can type HTML formatting tags in any field except the See Also field. See the following table for examples.

Here's a list of Javadoc comments:

Javadoc comment	**Description**
`Description`	Type the documentation for this member. Press Enter where appropriate (for example, to create line breaks or blank lines in the code). To create blank lines in the resulting HTML file, however, you need to use HTML tags such as <BR> or <P>. Note that the member name is automatically included.

Javadoc comment	Description
`Parameters`	Choose a parameter from the drop-down list, then enter a description of the parameter that you want to appear in the `@param` tag. For example, if you type `the listener to add` for the parameter "listener," the tag will be `@param listener the listener to add`. This field appears when you select a method that has parameters.
`Returns`	Type a description of the return value that you want to appear in the `@return` tag. This field appears when you select a method that has a return value.
`Author`	Type the information that you want to appear in the `@author` tag; for example, `David Jones`. Separate each author by a comma (,); this inserts separate `@author` tags for each author. This field appears when you select a class.
`Version`	Type the information that you want to appear in the `@version` tag; for example, `2.0a October 15, 1998`. This field appears when you select a class.
`Since`	Type the information that you want to appear in the `@since` tag; for example, `JDK 1.1`.
`Deprecated`	Type the information you want to appear in the `@deprecated` tag; for example, `This method was replaced by the newconvert method.`
`See Also`	Type the information that you want to appear in the `@see` tag; for example, `JScrollPane`. For multiple See Also references, press Enter at the end of a line to begin each new reference.

Javadoc comment	Description
`Exceptions`	Choose an exception from the drop-down list, then enter a description of this exception that you want to appear in the `@exception` tag. The exceptions are defined with a `throws` method, which is part of a method declaration. For example, if you type `something didn't happen` for the exception "myexception," the tag will be `@exception myexception something didn't happen`. This field appears when you select a method.

Here's a list of some HTML tags that you might want to use in your Javadoc comments. For more information, consult your HTML documentation.

HTML tag	Description
`<b>`*text*`</b>`	Place the `<b>` tag at the beginning of the text that you want to make bold, and the </b> tag at the end of the specified text.
`<i>`*text*`</i>`	Place the `<i>` tag at the beginning of the text that you want to italicize, and the `</i>` tag at the end of the specified text.
`<c>`*text*`</c>`	Place the `<c>` tag at the beginning of the text that you want to display as sample code, and the `</c>` tag at the end of the specified text.
`<p>`*text*`</p>`	Start a new paragraph with the `<p>` tag, and end it with the `</p> tag`.
`<br>`	Create a line break (a blank line).
`<blockquote>` *text* `</ blockquote>`	Place the `<blockquote>` tag at the beginning of the text that you want to indent, and the `</blockquote>` tag at the end of the specified text.

5 To add the comments to the source file, click another member in the tree or close the editor by clicking the Windows close box or right-clicking and choosing Close.

To cancel changes, right-click and choose Revert.

Mod appears in the bottom-right corner if you have made changes that aren't reflected in the source file.

Warning: The Javadoc comments appear in the file, but you must save the file in the Source Editor to save the changes made to the file.

To go to the currently displayed Javadoc comment in the source file:

- ◆ Right-click and choose Go to Definition.

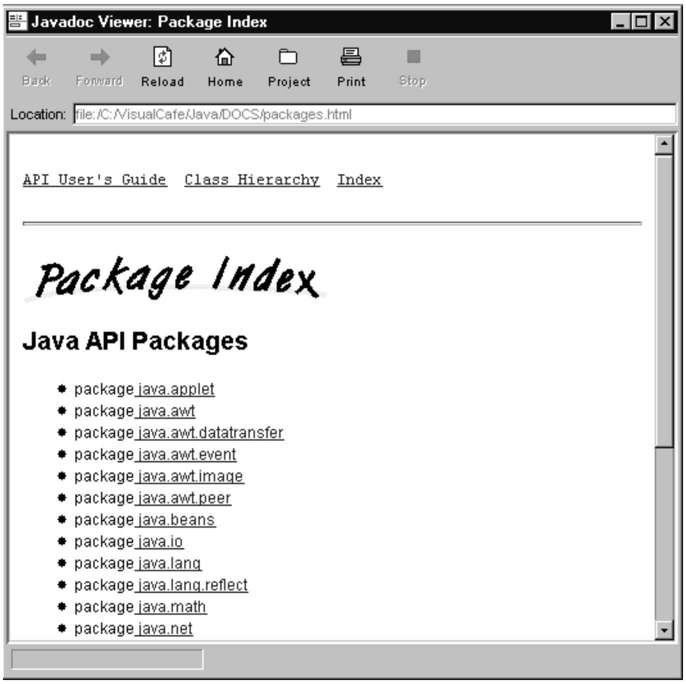
To update the Javadoc Editor display after making changes in the source file:

- ◆ Click on the Javadoc Editor to bring it to the front; the contents will be automatically updated.

Using the Javadoc Viewer

Use the Javadoc Viewer to quickly locate Javadoc documentation for files, packages, JavaBeans in the Component Library, classes and methods in the Source window, and more.

Any time you choose the Javadoc command from the View menu, Visual Cafe generates or updates the Javadoc for that file, if it belongs to an open project. If Visual Cafe has never generated Javadoc for a Java file, Visual Cafe saves the Java file and generates the Javadoc. If Visual Cafe has already generated Javadoc for a Java file, and you've modified the Java file since then, Visual Cafe asks you to save it first; you must save the file to get updated Javadoc output.



file:/C:/VisualCafe/Java/DOCS/packages.html

API User's Guide Class Hierarchy Index

Package Index

Java API Packages

- package java.applet
- package java.awt
- package java.awt.datatransfer
- package java.awt.event
- package java.awt.image
- package java.awt.peer
- package java.beans
- package java.io
- package java.lang
- package java.lang.reflect
- package java.math
- package java.net

For files you are working on in a project, you need to choose Produce Javadoc from the Project menu before you can view the documentation.

Tip: If you want to save time when generating an index, tree, or both for a project with multiple source files, you can select one file in your project and choose Produce Javadoc from the Project menu. This file should use the classes for the entire project, such as a file with a main method that calls everything else in your project, so that `AllNames.html`, `tree.html`, and `packages.html` are generated completely. The Javadoc documentation at this stage will include all referenced classes (except system classes) for the source file, but not Javadoc comments for the other source files. Then, to get all Javadoc comments, you should do another build with the option Always Produce Javadoc When Compiling selected, and with both Generate Tree and Generate Index selected.

To open the Javadoc Viewer:

◆ From the View menu, choose Javadoc Viewer.

To view the Javadoc for a file:

◆ Right-click a file in the Files view of the Project window and choose View Javadoc.

To view the Javadoc for a class:

◆ Right-click an object in the Objects view of the Project window and choose View Javadoc.

To view the Javadoc for a package:

◆ Right-click a package in the Packages view of the Project window and choose View Javadoc.

To view the Javadoc for items in the Source window:

◆ Right-click in the body or a Javadoc comment of a class or method, and choose View Javadoc.

 or

◆ Right-click a field or a Javadoc comment for a field, and choose View Javadoc.

 If you choose View Javadoc while you are positioned at a class, method, or field, the Javadoc Viewer displays the Javadoc for the file displayed in the Source window, at the location where the documentation for the class, method, or field resides. If there is no documentation for an item, such as a private method when you didn't generate Javadoc for private methods, the viewer displays the HTML file at the top of the file.

To view the Javadoc for a component in the Component Library:

◆ Right-click a component in the Component Library and choose View Javadoc.

 For Beans you have added to the Component Library, you can view Javadoc if the Javadoc exists in the path set in the Visual Cafe `sc.ini` file with the `JAVADOC_VIEWER_PATH` variable. The viewer uses the first matching HTML file it finds in the path. The Javadoc cannot be in a JAR file.

 For more information about working with the `sc.ini` file, see "Setting environment variables in the sc.ini file" on page 3-72.

To go to the Javadoc home for the Visual Cafe environment:

- ◆ In the Javadoc Viewer, click the Home button.

  The viewer displays Javadoc for the JDK packages, as well as other API information. By default, the viewer looks in `VisualCafe\Java\docs` for the Java API documentation. Visual Cafe uses the `JAVADOC_VIEWER_HOME` variable in the `sc.ini` file to set the location of the Javadoc system home. For more information, see “Setting environment variables in the sc.ini file” on page 3-72.

To go to the Javadoc project home:

- ◆ In the Javadoc Viewer, click the Project button.

  The viewer displays a package index for the project. The Javadoc Documentation Directory (which you can set in the Project Options dialog box), holds the Javadoc documentation for your project. This directory is by default *Projectname*`\api`. If you didn’t generate Javadoc for a project, it’s automatically generated, which also means that any unsaved files are saved.

To navigate forward and back between pages:

- ◆ In the Javadoc Viewer, click the Back and Forward buttons.

To reload the Javadoc HTML file:

- ◆ In the Javadoc Viewer, click the Reload button.

To stop loading an HTML file:

- ◆ In the Javadoc Viewer, click the Stop button.

To print the Javadoc file:

- ◆ In the Javadoc Viewer, click the Print button.

Specifying Javadoc folders

Your project’s Javadoc documentation will by default reside in the `api` folder below your project folder. However, you can can change this setting if you like. Also by default, the Javadoc documentation resides in a different folder, `VisualCafe\Javadocs`. Although in most cases you won’t need to change this setting, you can do so if you wish.

To specify Javadoc folders:

1 Choose Options from the Project menu and click the Directories tab.

2 In the Show Directories For list, choose Output Files.

3 In the Javadoc fields, type a directory and full path in the field, or select a directory by clicking the Browse (...) button that displays in the field.

  The Javadoc System Documentation Directory is where the Java API documentation is located. The Javadoc Documentation Directory is where the documentation for your project is placed. If the directories are different, links are generated to existing Visual Cafe Javadoc documentation in the system directory. If you make both directories the same, then full index, tree, and package files are generated in the documentation directory, which can take more time but may be what you need for deployment.

4 Click OK.

  The change takes effect the next time you produce Javadoc.

Setting Javadoc options

Producing Javadoc documentation is usually very fast. However, producing the index and tree can take more time and memory.

Tip: make sure that you have enough disk space to store the documentation.

To set Javadoc options:

1 Choose Options from the Project menu and click the Compiler tab.

2 In the Compiler Category list, choose Javadoc.

3 From the Javadoc view, set the following options as needed:

Select...	**To specify this...**
Always Produce Javadoc When Compiling	Produce Javadoc documentation every time you compile your project and place the HTML files in the folder specified in the Javadoc Documentation Directory field (Directories tab). Visual Cafe generates Javadoc only for files that have changed; this reduces compilation time. In addition, the index, tree, and packages files are not generated.
Include Version	Include the Javadoc information specified with the `@version` tag.
Include Author	Include the Javadoc information specified with the `@author` tag.
Generate Index	Generate `AllNames.html`, which is an index of all fields and methods. `packages.html` is also generated; it lists all the packages and has links to other HTML files that list the classes in each package. If the system and project Javadoc directories are not in the same location, the links are generated relative to the system directory; otherwise, all information is copied into the Javadoc directory, including the graphics files. This option is used only when you choose Produce Javadoc from the Project menu.
Split Index	Split `AllNames.html` into individual index files, `index-B.html` through `index-Z.html` and `AllNames.html` for A; `index-Other.html` is for items beginning with an underscore (_). You can choose this option only if Generate Index is selected.

Select...	To specify this...
Generate Tree	Generate `tree.html`, which is a class hierarchy display with links to the class documentation. `packages.html` is also generated; it lists all of the packages and has links to other HTML files that list the classes in each package. If the system and project directories are not in the same location, the links are generated relative to the system directory; otherwise, all information is copied into the Javadoc directory, including the graphics files. This option is used only when you choose Produce Javadoc from the Project menu.
Include Imported System Classes	Include imported system classes (`java.*`, `sun.*`, and `com.*`) in the index, tree, and package files that are generated. If you choose to include imported system classes, you'll find them in the Javadoc System Documentation Directory.
Use Sun's Javadoc	Use the Sun Microsystems Javadoc compiler.
Classes to Document	Choose the option that you need. The lower the option is in the list, the more is included. For example, selecting And Private means that Public, Protected, Package, and Private files are included.

You can set the name and path of the Javadoc executable in the Virtual Machines tab of the Environment Options dialog box. For more information, see "Using different Java virtual machines in Visual Cafe" on page 5-22.

You can use the Sun Microsystems Javadoc compiler, but you'll find that the one in Visual Cafe is faster. In addition, the Javadoc compiler in Visual Cafe:

- ❖ Supports relative links from the project documentation to the system documentation directory.
- ❖ If necessary, will automatically copy the `.gif` image files referenced in the documentation to where the documentation is generated.

- ❖ Includes a list of a class's inner classes (if any) at the top of each class's HTML file.
- ❖ Emits warnings if it finds unknown Javadoc tags.
- ❖ Includes a default package showing classes which don't have a package statement.

4 Click OK.

The change takes effect the next time that you produce Javadoc.

Searching one or more files

Visual Cafe has powerful search capabilities that make editing your work easier. This section describes the commands in Visual Cafe's Search menu. In addition, the options in the Find dialog box, which opens when you choose Find from the Search menu, are outlined.

Using wildcards in searches

You can use regular expressions to search for more specific items. Regular expressions are wildcard characters. The pattern you search for can be a text string or a regular expression.

Use any of the following wildcard characters to further customize your searches:

Use this wildcard...	To search for this...
??	Any character
*	Zero or more occurrences of any character
@	Zero or more occurrences of the previous character or expression
% or <	Beginning of a line
$ or >	End of a line

Use this wildcard...	To search for this...
[...]	Any of the characters listed between the brackets [and]. You can use a hypen (-) to indicate a range of characters. For example, [abc] matches a, b, or c; [a-z] matches any lowercase letter; [A-Za-z] matches any upper- or lowercase letter.
[~...]	Any character except those listed between the left bracket and tilde [~ and the right bracket]. You can use a hypen (-) to indicate a range of characters. For example, [~A] matches any character but A; [~abc] matches any character except a, b, or c; [~A-Za-z] matches any non-alphabetic character.
\	Take the following character literally instead of using it as a wildcard character. For example, you can use * to search for an asterisk (*) or \\ to search for a backslash character (\).
\t	Tab character
\f	Formfeed character

Searching and replacing

The Source window offers text-based search-and-replace functions that let you search the active window for a string and replace one string with another. In addition, the global find feature allows you to locate a string in any set of files.

You can also jump to specific points in a file by using Search menu commands. These commands are described later in this section.

To search for a string in a file:

1. (Optional) Select some text in the Source window.

   The selected text becomes the "Find what" criterion.
2. While the Source window is active, choose Find from the Search menu.

3 Type the string you want to search for in the Find what field or choose a previous string from the drop-down list, and select the options you want.

Option	Description
Match case	Only find text strings that match the search criteria exactly, including case.
Match whole words only	Search for any string that matches the search criteria and is preceded and appended by a blank space. Text is a match only if it appears exactly like the search string, not as a portion of a larger string. For example, `in` matches `in`, but not `include`.
Regular Expression	Accepts regular expression wildcards in the expression syntax. Click the right-arrow button to display a list of valid special characters. When you select from this list, the Regular Expression option is automatically selected.

Tip: Leave the Regular Expression option deselected (for speed) if you simply want to locate a string.

4 Click Next to search forward or Previous to search backward.

To repeat the search in the same direction:

◆ Choose Find Again from the Search menu.

To replace a string with a different string:

1 (Optional) Select some text in the Source window.

The selected text becomes the “Find what” criterion.

2 While the Source window is active, choose Replace from the Search menu.

3 Type the string you want to search for in the Find what field or choose a previous string from the drop-down list.

4 Type the replacement string in the Replace field or choose a previous string from the drop-down list.

Note: You cannot use wildcard characters in a replacement string.

5 Select any options you want.

Option	Description
None	Search for any string matching the search criteria. Case is not considered.
Match case	Only find text strings that match the search criteria exactly, including case.
Match whole words only	Search for any string that matches the search criteria and is preceded and appended by a blank space.
Regular expression	Allows you to define a search string using regular expression wildcard syntax. Click the Right-Arrow button to display a list of valid special characters.
Confirm changes	You're prompted before each replacement is performed. If this option is not selected, the editor replaces all occurrences of the search string without confirmation messages.
Search only in selection	This option is valid if you have a block of text selected in the editing window. This option limits the scope of the search to the text in the selected block.

6 Click Replace.

The file is sequentially scanned for matching strings, and the matching strings are replaced with the replacement text.

Tip: Choose Undo Search from the Edit menu to undo the entire set of search-string replacements.

To search for a string in multiple files:

1 From the Search menu, choose Find in Files. You can choose to search:

❖ All source files in the current project

- ❖ All files listed in the Find in Files window (which opens after the first search)
- ❖ All files matching the criteria you specify, including file name, directory, date, time, and file attributes

2 Choose which files to search.

For more information, see “Specifying the search file type and location” on page 4-78.

3 Set advanced search options as needed.

For more information, see “Setting advanced search criteria” on page 4-79.

Comparing two files

You can compare two text files line by line. Upon completion of the comparison, the two files display in separate windows where you can scroll through the lines that are different.

To compare two files:

1 Choose Compare Files from the Tools menu.

The Compare Files dialog box appears.

2 Select the file to be used as the base text, File 1.

If either file is currently open, Visual Cafe uses the version that is in memory, rather than reading the file from the disk. If you want to pinpoint recent changes, first save the open file under a different name, then compare it to the file on the disk.

Tip: Click the Down-Arrow button to show a drop-down list containing the names of files that were recently compared. Click the Browse button to display an Open File dialog box.

3 Select the file, File 2, that is mapped against File 1.

4 Specify the line number that the search should start from in both files. These numbers can be different.

5 Specify the arrangement of the source windows that display the results. Choose from the following options:

Option	Description
Horizontal	Displays one window above the other.

Option	Description
Vertical	Displays one window to the side of the other.

Visual Cafe then performs the comparison on a line-by-line basis.

6 When a mismatch is found, the lines in both files are highlighted and Visual Cafe reports where the mismatch was found.

- ❖ Click Next Match to resynchronize the comparison. The next set of matching lines is then highlighted, and the Compare Files dialog box reports where the match occurs.
- ❖ Click Next Difference to find the next mismatched line. You can continue the comparison until no more differences are found.

Specifying the search file type and location

You can specify the search criteria, the type of files to search, and the file's location in the Name & Location tab of the Find in Files dialog box.

To specify search criteria:

1 Select the appropriate search criteria options at the bottom of the window.

Option	Description
Search using regular expressions	Choose what you want to search for from the pop-up menu of regular expressions. This option is the default.
Search with wildcard symbol	Select the Match Wildcards option.
Search with exact case matching	Select the Match case option.
Search for an exact match to a whole word	Select the Match whole word only option. This option limits matches to files that contain the search criteria string preceded and followed by a space, tab, or punctuation character, or a search string at the beginning or end of a line.

2 Specify the search pattern in the Find what field. The drop-down list displays the previous 16 search strings. If Regular expression is

selected, you can use the more button (pictured at left) to select valid regular-expression characters.

3 Specify the types of files to search by entering file extensions or selecting extensions from the drop-down list.

4 Select the scope of the search. You can specify a folder, the current project, or the last set of files found in a previous search.

5 To expand a search into subfolders, select the Search subfolders option.

Setting advanced search criteria

You can specify file-attribute and modification search criteria from the Advanced tab of the Find in Files dialog box.

1 Set attribute criteria by enabling the appropriate attribute options. This narrows the search scope.

File attributes are Archive, Read Only, System, and Hidden.

Note: The Attributes checkboxes are three-state checkboxes. If an attribute is enabled, files with the given attribute are searched. If an attribute is cleared, files without the given attribute are searched. If an attribute is dimmed, the attribute is ignored when Visual Cafe decides which files to search.

2 Specify modification criteria by selecting the Files created or modified option.

Set the appropriate values in the Date and Time fields. The field value ignore disables the associated Date or Time field.

Date: Select Ignore to ignore the date. Otherwise, specify a date and one of the options. For instance, specify Is and `11/6/94` to search files last modified on November 6, 1994, or Greater and `4/1/90` to search files last modified after April 1, 1990.

Time: Select Ignore to ignore the time. Otherwise, specify a time and one of the options.

Jumping to a specific location

When searching a file, you can choose to go to a specific line, a particular function, an event or method, a buffer, a bookmark, a definition, or an error. A **buffer** is an open file or Class Browser window that is in temporary memory.

To go to a specific line:

1 While the Source window is active, choose Go To Line from the Search menu.

  The Go To Line dialog box appears.

2 Type the line number in the text box and click OK.

  The insertion point moves to the beginning of the specified line. If any text is currently selected, the selection is extended to include that line.

To go to a function:

1 From the Search menu, choose Go to Function.

  The Go to Function dialog box appears.

2 Select a function name from the scrolling list or type a function name.

  You can change how function names display in the list with the Show member name first option. This option shows the member name first in the display. The default is the form name first.

3 Click OK.

  The insertion point moves to the beginning of the specified function.

To jump to an event or method:

1 Choose an object from the Source window's Objects drop-down list.

2 In the Events/Methods drop-down list, choose the event or method.

  Existing events and methods are shown in bold. If you choose an event or method that's not bold, it's created for you.

To go to a buffer:

1 Choose Go to Buffer from the Search menu.

  The Go to Buffer dialog box appears, where you can change the options for the current edit buffer or those of another.

  The Go to Buffer dialog box allows you to switch to an editing buffer. A buffer is created for each editing window.

2 Select the category of buffers you want to view:

Option	Description
File Buffer	A Source window to edit an entire file.
Member Buffer	A Source pane to edit a particular member definition

3 Select a buffer from the list.

4 Perform any appropriate task(s), as follows:

Option	Description
Go To	Make the selected buffer's window current.
Options	Display the Current Buffer Options dialog box to define automatic buffer formatting and maintenance.
Save	Save the buffer's content with its current file name.
Save As	Save the buffer's content with a new name.
Close	Close the selected buffer's window and prompt you if there are unsaved changes.

To set a bookmark:

1 Choose Bookmarks from the Search menu.

The Bookmark dialog box appears, where you can add, remove, or go to a bookmark. You can set and move to as many as ten locations in your source files.

Use the Bookmarks dialog box to view or edit bookmarks. Bookmarks are specific to the text in which the bookmark was dropped. Adding text above the bookmark pushes the line number of the bookmark automatically.

2 The bookmark list shows the locations of the ten bookmarks, by file, line, and column. Click an entry to select it; double-click to go to it.

Button	Description
Go to	Moves the insertion point to the selected bookmark. You an also double-click the bookmark in the list.
Clear	Clears the current (highlighted) bookmark.
Drop	Sets the selected bookmark to the current insertion point. The entry in the bookmark list is updated to show the file, line, and column.

Note: Bookmarks are saved through the current Visual Cafe session only.

To go to a definition:

1 With the Source window active, select or click on a class or member, then choose Go to Definition from the Search menu, or right-click and choose Go to Definition.

A definition could be a class, member, or variable definition, for example.

2 If a Members window displays, select the member you want to view.

A Class Browser window appears. For more information, see “About classes, members, and the Class Browser” on page 4-1.

To search for an error:

With the Source window active, click the Search menu and select one of the following options:

- ❖ Go to Current Error
- ❖ Go to First Error
- ❖ Go to Previous Error
- ❖ Go to Next Error

By selecting one of the above options, you can search for an error in your source code and navigate easily among errors.

For more information about errors in source code, see Chapter 6, "Debugging Your Program."

Searching for a matching delimiter

You can search for matching delimiters in a file. Delimiters separate your code into segments, and can be parentheses ( ), brackets [ ], or braces { }. Search for a matching delimiter to quickly determine the scope of the current code segment.

To search for a matching delimiter:

1 In the Source window, position the insertion point in front of a delimiter (a parenthesis or bracket, for example; a common problem in source code is parentheses ( ), brackets [ ], or braces { } that don't match).

2 Choose Go to Matching Delimiter from the Search menu.

The insertion point moves to the other half of the pair.

To check delimiters in a file:

◆ Choose Format Options from the Source menu, then Check Delimiters from the submenu.

When you select Check Delimiters, this specifies that if you type a right parenthesis), square bracket], or brace }, the editor briefly highlights the corresponding left delimiter. If no matching delimiter is found, an error message displays in the status bar.

For more information about setting formatting options for a file, see "Setting text formatting for a single file" on page 4-53.

Tip: If you've chosen the Check Delimiters option, delimiter checking occurs automatically, including checking for text in strings and comments.

To check delimiters globally:

◆ Choose Environment Options from the Tools menu, then Format from the submenu.

When you select Check Delimiters, this specifies that if you type a right parenthesis), square bracket], or brace }, the editor briefly highlights the corresponding left delimiter. If no matching delimiter is found, an error message displays in the status bar.

For more information about setting formatting options for all files, see “Setting text formatting for the Visual Cafe environment” on page 4-54.

Working with imported Java code

Java source code for all kinds of applets and applications is already written and available for you to download from the Internet, compile, and execute. Also, many books written on Java and Visual Cafe contain sample programs for you to use right away.

Importing source code

If you have some source code that you’ve created with a text editor, you can use that source code in Visual Cafe by pulling it into a Visual Cafe project. Here’s how you can insert a `.java` file into Visual Cafe.

To import a .java file from outside of Visual Cafe:

1. Make sure the source file is an extension of the Applet class if you are importing applet code. If so, then create an AWT Applet project.
2. Click the Files tab of the Project window.
3. Right-click in the Project window and choose Insert Files from the pop-up menu.
4. Navigate to the file, select it, and click Add.
5. Click OK.

   The file is added.

Importing Visual J++ 1.1 projects

Your Visual J++ 1.1 project is converted automatically when you open the workspace (`.dsw`) or project (`.dsp`) file with Visual Cafe. If you open the workspace file, some of your workspace options (stored in an `.opt` file) are preserved; these settings might be lost if you opened the project file directly.

Note: You cannot import Visual J++ 6.0 projects into Visual Cafe.

When Visual Cafe converts a project through the workspace file, it gives the project the same name as the workspace and saves it immediately in the same folder as the workspace. Therefore, if your workspace contains multiple projects, you should first convert the projects that are in their own folders, then convert the project stored in the same folder as the workspace file.

Note: It's best to work from copies of your Visual J++ files instead of the originals.

Considerations when importing Visual J++ projects

While Visual Cafe has projects, Visual J++ has workspaces that contain projects. A J++ workspace is made up of a workspace file and an options (`.opt`) file. Information about the projects in a workspace is stored in a project file (one file for every project) and in the options file. Because the name of the options file is not stored in the project file, it's usually better to import projects through the workspace file rather than the project file.

The following options are preserved from the Visual J++ `.opt` file when you import a project through the `.dsw` file, or through a `.dsp` file that has the same name as the `.opt` file (and the `.opt` file is in the same folder.):

Visual Cafe project option	Visual J++ configuration that sets it
Project type field in Project tab	The Debug/Execute project under Browser and Stand-alone interpreter options determine the setting.
Start with Web page field in Project tab	The Use parameters from HTML page value is used; or the Enter parameters below option tells Visual Cafe to use Automatic.
Main class field in Project tab	The Class for debugging/executing value is used.
Runtime arguments field in Project tab	The Program arguments value is used.
Class Files list in Directories tab	The Class path directories setting is used. Note that in Visual Cafe the Append class path and Autogenerate class path options are selected by default.

The following options are preserved from the `.dsp` project file, whether you open a workspace or project file:

Visual Cafe project option	Visual J++ configuration that sets it
Output directory field in Directories tab	The Output directory value is used.
Show compiler warnings option in Compiler tab	A Warning Level of None tells Visual Cafe to deselect the option; any other value means the option is selected.
Generate debug information option in Compiler tab	The Generate Debug info value is used.

Here are some additional considerations when importing Visual J++ projects:

- If you want to maintain your Visual J++ workspace and projects, you should copy the files to another folder before conversion.
- Converted projects cannot be saved back into Visual J++ format.
- Only Visual J++ version 1.1 workspace and project files can be imported. However, you can add Java source files created in another product to a Visual Cafe project. For instructions, see "Adding an existing file to a project" on page 3-44.
- Visual Cafe can read Visual J++ `.java` files that are 100% pure Java, without proprietary Java extensions implemented by Microsoft.
- `.class` files compiled with Visual Cafe might not be compatible with Visual J++, so you should make sure your output folders are different for each product.
- You can't import projects that use variable persistence, such as having the location of a file be stored as `d:\mysource\$(SRCDIR)\a.java`, where `SRCDIR` is an environment variable and `a.java` is the file. In Visual Cafe, you can't use environment variables to control where Visual Cafe looks for a `.java` file. If you want to import a project that uses variable persistence, you need to remove variable persistence before importing the project into Visual Cafe.

While Visual J++ lets you specify a different project type — applet or application — for different configurations, Visual Cafe supports one project type for every project. If the project type is different for the debug and final option sets, Visual Cafe uses the project type specified for debug.

Notes: You can choose to use Visual J++ keyboard shortcuts by specifying them in your Visual Cafe environment options. See “Mapping Visual Cafe commands to key sequences” on page 3-73 for more information.

In Visual Cafe, a project closes when you close the Project window. In Visual J++, just the window closes when you close the workspace (a project).

Importing a Visual J++ project by way of the .dsw or .dsp file

You can import a Visual J++ 1.1 file by way fo the `.dsw` workspace file or the `.dsp` project file.

To import a Visual J++ project via the .dsw workspace file:

1. From the File menu, choose Open.

   The Open dialog box appears.
2. Navigate to the folder that contains the `.dsw` file for the project you want to import.
3. Choose Visual J++ Workspace Files in the Files of type field.

   The `.dsw` and `.dsp` files appear.
4. Select the `.dsw` file from the list, then click Open.
5. If there’s more than one project in the workspace, a dialog box appears that asks which project you want to convert. Select the project, then click OK.
6. If you have more than two configurations defined for the project, or if you have two configurations and neither of them in named “Debug,” a dialog box appears that asks what configuration you want to use for the Visual Cafe Debug and Final option sets. Select the configuration and click OK in each dialog box.

   The project is converted and opens in a Visual Cafe Project window.
7. Review the Visual Cafe project options and change them as needed.

   For more information, see “Setting environment variables in the sc.ini file” on page 3-72.

8 If the Visual Cafe project was in a different folder than the `.dsw` file, choose Save As from the File menu to save your new project to the folder where the project `.dsp` file is located. If you wish, you can rename the project at the same time.

See "Saving a project" on page 3-39 for more information.

Tip: Remember that a Visual J++ project is given the same name as the Visual J++ workspace if the project file is in the same folder as the workspace file.

If you save the project to another folder, you can delete the Visual Cafe project files in the same folder as the workspace. Or you can just let them be overwritten when you import the next project.

To import a project via the .dsp workspace file:

1 From the File menu, choose Open.

The Open dialog box appears.

2 Navigate to the folder that contains the `.dsp` file for the project you want to import.

3 Choose Visual J++ Workspace Files in the Files of type field.

The `.dsw` and `.dsp` files appear.

4 Select the `.dsp` file from the list, then click Open.

5 If you have more than two configurations defined for the project, or if you have two configurations and neither of them is named "Debug," a dialog box appears that asks what configuration you want to use for the Visual Cafe Debug and Final option sets. Select the configuration and click OK in each dialog box.

The project is converted and opens in a Visual Cafe Project window.

6 Review the Visual Cafe project options and change them as needed.

For more information, see "Setting environment variables in the sc.ini file" on page 3-72.

7 Save your new project.

See "Saving a project" on page 3-39 for more information.

C H A P T E R 5

Compiling and Deploying Your Project

Visual Cafe applets and applications are cross-platform Java programs that you design, develop, and build using the Visual Cafe development environment. Both are executed by a Java Virtual Machine, but applets run only within a Web page, while applications run on their own. You can use Visual Cafe to compile a Java program to bytecode, run it to verify its behavior, debug it as necessary, and eventually deploy it to users.

Compiling your Java program

You can compile and run a project at any time during its development cycle. Visual Cafe automatically saves files in the project before running. For information about running your program within the Visual Cafe environment, see “About files in a project” on page 3-42.

Running a project

Projects speed development by compiling only the source files that have changed since the last time the project was built. Visual Cafe manages the project for you by automatically analyzing the dependencies of the source files and updating the project information each time you build the project.

So, if you make changes to only two files in a project and then direct Visual Cafe to build the project, it recompiles only those two files.

To run a project:

◆ Do one of the following:

  ❖ Choose Execute from the Project menu to run the project without the debugger.

  ❖ Choose Run in Debugger from the Project menu to run the project and start the debugger. For more information, see Chapter 6, “Debugging Your Program.”

Important: Before running in the debugger, make sure your project options are set to debug. When you’re ready to compile your final program, make sure the project options are set to final. See “Specifying whether builds are debug or final” on page 5-16 for more information

---

For information on running applets, see these topics in this section:

◆ Making applets run in the AppletViewer or a browser

◆ Specifying an applet’s HTML file

For information about running applications, see these topics in this section:

◆ Specifying the main class to run for an application

◆ Specifying arguments for application execution

Making applets run in the AppletViewer or a browser

When running applets from Visual Cafe, you can launch your applets in the AppletViewer associated with the Visual Cafe environment or in the Web browser of your choice.

Running applets in the AppletViewer can be faster, because the browser does not have to start up. However, it’s a good idea to test your applets in popular browsers before deployment.

Note: When you run an applet in the debugger, it is always run in the AppletViewer by default.

To specify where applets are to run:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

3 In the Project Options dialog box, click the Project tab.

4 Select Applet as the Project Type, if needed, then select Execute applet in default Web browser if you want to run the applet in a browser; deselect it if you want to run the applet in the AppletViewer.

5 Click OK.

The change takes effect the next time you run your project.

Tip: When this option is selected, Visual Cafe looks for the application you have associated with the file extensions `.htm` and `.html` (Hypertext Document file type). If an association does not exist, you must define one.

To set file associations (including the open action) in Windows 95, 98, and NT 4.0, from a Windows file system window (such as the Explorer), choose View, then Options (In Windows 98 it's Folder Options), then click the File Types tab.

A browser might have already set the association for you (for example, the file type Netscape Hypertext Document).

Specifying an applet's HTML file

An applet is launched from an HTML file that has an applet tag. Visual Cafe can automatically create an HTML file with applet tags for all the applets in your project. When you run your project from Visual Cafe, you can specify what HTML file to use to display your applets. Here are some scenarios:

- ◆ If you run your project with the automatically generated HTML file in the AppletViewer, all of your applets appear in separate windows.
- ◆ If you run your project with the automatically generated HTML file in a Web browser, all of your applets appear in an otherwise blank browser window.
- ◆ If you run your project with your own HTML file in the AppletViewer, each applet that has an applet tag will appear in a separate window.
- ◆ If you run your project with your own HTML file in a Web browser, the HTML file appears in a browser window, including the applets as specified in the file.

If you want to test the files for a Web site, you could specify the home page as the starter HTML and run the page from a Web browser. Then you could access the other pages from this page to make sure your applets work.

To test the files for a Web site:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.
3. In the Project Options dialog box, click the Project tab.
4. Select Applet, if needed, then specify an HTML file in one of these ways:
   - ❖ Choose (Automatic) to run all of the applets from an automatically generated HTML file that is blank.
   - ❖ Choose one of your own HTML files from the pop-up menu. HTML files that you've added to your project automatically appear in the pop-up menu.
   - ❖ Click the Browse button (...) to browse for an HTML file.
5. Click OK.

   The change takes effect the next time you run your project.

Configuring an application to run in Visual Cafe

In order to configure an application to run in Visual Cafe, you need to specify some options. You need to specify the class that has a `main` method in it. Also, if your application accepts command-line arguments, you need to specify them.

About the main class in bytecode and native applications

The **main class** is the name of the class with a `main` method. For both a bytecode Java application and a native Win32 Java application, the main class is the starting point of execution.

To run a Java application from within the Visual Cafe environment, you must specify the starting point of your application (the Java class file) so Visual Cafe knows how to run your application. If you started a project with the AWT Application template or inserted a Java file that already had a `main` method in it, the main class was already specified for you. If there is no entry in this field, Visual Cafe tries to use the project name.

You should note these differences between bytecode and native applications:

- When you run a bytecode application from the command line, you type the name of the Java file that contains the main method as the argument to `java.exe`, after any switches such as for the class path, and so on. For a native Win32 application, you run the application outside of the Visual Cafe environment, as you would any other executable, and use the application name rather than the main class name. See "Specifying the main class to run for an application" on page 5-5 and "Specifying the name of a native application or DLL" on page 11-7 for more information.
- An application must have a main class that contains a main method with this signature:

  ```
  static public void main(String args[])
  ```

  If a bytecode application does not have a method of this format, the application can compile but will not run. If a native application does not have a method of this format, the application can't compile or run.

If your application also accepts arguments on the command line (the main method takes arguments), you need to specify those. See "Specifying arguments for application execution" on page 5-6.

Specifying the main class to run for an application

You can specify the main class for an application; when executing an application, the main class is where execution begins.

To specify the main class to run for an application:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.
3. In the Project Options dialog box, click the Project tab.
4. Select Application, if needed, then type the name of the class file in the Main Class field; for example, `Frame1` or `Frame1.class` are both acceptable. If the class is inside a package other than the Default Package, you need to type ***`package_name.class_name`*** in this field.

   You can enter one name only.
5. In the main( ) class pop-up menu, select the name of the class file.

Note: The main( ) class pop-up menu is not populated until after compiling. After compiling, the pop-up menu contains all the classes that have a suitable `main( )`.

6 Click OK.

The change takes effect the next time you run your project.

Note: If you rename a class that appears in this field, Visual Cafe updates the field for you.

Specifying arguments for application execution

If your application accepts arguments on the command line (the `main` method takes arguments), you need to specify them so Visual Cafe can run your application from its environment. For example, the Sun Java compiler is written in Java and takes command-line arguments.

You also need to specify the main class. See “Specifying the main class to run for an application” on page 5-5.

To specify command-line arguments:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

3 In the Project Options dialog box, click the Project tab.

4 Select Application, if needed. In the Program arguments field, type any arguments that should be passed to the program when you run it. Delimit the arguments with a space.

5 Select Application, if needed, then type the name of the class file in the Main Class field; for example, `Frame1` or `Frame1.class` are both acceptable. If the class is inside a package other than the Default Package, you need to type ***`package_name.class_name`*** in this field.

You can enter one name only.

6 Click OK.

The change takes effect the next time you run your project.

Compiling from the SJ command line

If you want to save memory when compiling, you can use the Symantec Javac (SJ) utility to compile your programs from a DOS window, and thus out of the Visual Cafe IDE. If you have a favorite editor and make utility, you can use the SJ tool to quickly compile programs. You can use the SJ utility when using batch programs, and you can also redirect error and warning messages to a file for easy viewing.

When you compile your project using the SJ command line utility, use the following format for your command:

```
sj { { switches } { file.java } { @file } }
```

The braces { } mean "repeated zero or more times." The only required parameter is a Java source file. For native Win32 applications or DLLs, you also need to specify either the `obj` or `link` switch.

SJ takes arguments in any order; if any arguments conflict, the rightmost argument takes precedence. SJ with no arguments prints a short help file to `stdout`.

@file means that *file* is searched for as an environment variable name, and if not found, as a file name. If the file is found, the text of the environment variable or file name is inserted in the command line as if it were part of the command line. In this way, you can give command line arguments from a file (for example, if you need to circumvent the command line length limit of an operating system). If the file is not found, no text is inserted.

SJ takes the following switches:

Switch	**Description**
`-cdb` *file*`.cdb`	Generate compilation database *file*`.cdb`. This database stores dependency information and is needed by the `make` switch.
`-make[:r|:w]`	Build only out-of-date files by checking dependencies between all files that were passed on the command line. You must supply a `.cdb` file with the `cdb` switch.
`-make:r`	Check dependencies on all imported files and rebuild them as needed.

Switch	Description
`-make:w`	Check dependencies on all imported files and issue a warning if they're out of date.
`-classpath` *path*	Use *path* instead of the setting of the environment variable `CLASSPATH`.
`-compact`	Removes line number information, which helps reduce file size.
`-d` *outputdir*	Set the output directory. The default is to put class files with their respective `.java` files.
`-debug`	When the compiler is Sun's compiler, `javac`, using this option causes the compiler to report diagnostic messages about its own execution. When the compiler is SJ, the switch is ignored.
`-depend` *file*`.dep` or *file*`.dar`	Generate dependencies into *file*`.dep`. You can read it to see what classes your files need in order to deploy. The `.dar` file also includes this list of classes, as well as the locations of the classes.
`-g`	Add debug information to output files so they can be debugged.
`-gl`	Adds line number information to `.class` files. This helps you to easily step through your code.
`-help`	Prints help on compiler switches.

Switch	Description
`-j` *codepage*	Override the system codepage default. Asian language character sets include double-byte characters, where certain prefix bytes mean that the following byte forms part of the character. Asian language characters can appear in `" "` strings, in `"` character literals, and in comments. The codepage guides the compiler in converting from the ASCII character set used in the source `java` files into Unicode. The compiler normally uses the system default codepage as a guide for translating double-byte character sets. However, this can be overridden with the `j` switch. Some common values are: `no -j` — Use system default locale and codepage `-j.932` — Japanese `-j.936` — Chinese `-j.949` — Korean `-j C` — Use “C” locale
`-noinline`	The `O` (optimize) switch will normally enable function inlining. To have optimization without function inlining, use the `noinline` switch. *Inlining* means that Visual Cafe takes a function’s code and embeds it in the calling function instead of calling the function. Inlining increases execution speed but also increases executable size.
`-nowarn`	Turn off warnings.
`-nowrite`	Compile to look for errors, but do not write out any files.
`-O`	Optimize.
`-verbose`	Display progress reports as compilation progresses. Without this option, messages are reported once per source file. With this option, additional messages are produced that specify the number of milliseconds to parse each Java source file.

Switch	Description
`-xdepend` *filename*`.dep` or *filename*`.dar`	Generate dependency information to `stdout` that's compatible with `javac`. Pass in a file name with either the `.dep` or `.dar` extension into which the information goes. The `.dep` file is a list of classes that you need to have in order to run the program. The `.dar` file also includes this list of classes, as well as the locations of the classes.

Javadoc-related switches

Some switches are specific to using the Javadoc utility. The -`classpath`, -`verbose`, and -`nowrite` switches, as described in the preceding table, also apply to the Javadoc utility.

Use the `-nowrite` switch to generate only HTML and no class files.

For more information about Javadoc, see "About Javadoc" on page 4-59. Also see the Sun Microsystems Javadoc home page at `http://java.sun.com/products/jdk/javadoc/index.html`.

The following command-line options apply only to Javadoc features:

Switch	Description
`-Javadoc`	Enable Javadoc to produce HTML documentation. This switch is unique to SJ.
`-public`	Include only public classes and members.
`-docdir` *dir*	*dir* specifies the destination directory where Javadoc stores the generated HTML files, also called the HTML Documentation output directory. The directory can be absolute or relative to the current working directory. The default is the `api` directory for new projects in Visual Cafe. When used from the command line, the default is the current directory. This switch is unique to SJ.
`-systemdocdir` *dir*	*dir* is the HTML System Documentation directory. This will default to `VisualCafe\java\docs`. Use this option to work out the relative links that are generated for referencing the system documentation. This switch is unique to SJ.
`-noauthor`	Omit the information specified by the `@author` tag, which is included by default. This switch is unique to SJ.

Switch	Description
`-noversion`	Omits the information specified by the `@version` tag, which is included by default. This switch is unique to SJ.
`-includesystem`	Generate documentation for all imported classes, including links to the system documentation directory for packages beginning with `java.*`, `sun.*`, and `com.*`. However, if the documentation directory and system documentation directory are the same, then full documentation is generated for these packages as well.
`-index`	Generate the package index (`AllNames.html`), which is not produced by default. `AllNames.html` is an index of all fields and methods. `Packages.html` is also generated; it lists all of the packages and has links to other HTML files that list the classes in each package. If the system and project Javadoc directories aren't in the same location, the links are generated relative to the system directory; otherwise, all information is copied into the Javadoc directory, including the graphics files.
`-splitindex`	Split the index file into smaller parts. Visual Cafe splits `AllNames.html` into individual index files, `index-B.html` through `index-Z.html` and `AllNames.html` for A; `index-Other.html` is for items beginning with an underscore (_). You can choose this option only if you've specified to generate the index files. This switch is unique to SJ.
`-tree`	Generate the class and interface hierarchy, which is not produced by default. Visual Cafe generates `tree.html`, which is a class hierarchy display with links to the class documentation. `packages.html` is also generated (by default); it lists all of the packages (including the Default package) and has links to other HTML files that list the classes in each package. If the system and project Javadoc directories aren't in the same location, the links are generated relative to the system directory; otherwise, all information is copied into the Javadoc directory, including the graphics files.
`-public`	Include only public classes and members.
`-protected`	Include only protected and public classes and members. This is the default.

Switch	Description
`-package`	Include only package, protected, and public classes and members.
`-nodeprecated`	Excludes paragraphs with the `@deprecated` tag. **Note:** Turning off deprecated warnings can help to speed up build times for large projects because the processing of Javadoc comments is skipped.

Native Win32 switches

This section lists the command line options that apply only to native applications and DLLs (not available in Visual Cafe Standard Edition). Either the `obj` or `link` switch is required; the rest are optional.

The command line options are as follows:

Option	Description
`-obj`	Generate native x86 object files.
`-link` *outputname*	Link generated object files into *outputname*`.[exe|dll]`. This command supplies the same functionality as the `obj` switch, and runs the linker as well.
`-main` *mainclass*	Specify the main class for the native executable. *mainclass* is a fully qualified class name. (If no main switch is specified, it will default to *outputname*.)
`-W`	Generate Windows-only executable (no console).
`-g`	Add debug information to output files so that the result can be debugged. Debug information is in CV4 format, so you can use it with debuggers other than the Visual Cafe debugger.
`-export` *package*|*class*	Specify classes or packages to be exported from the DLL, so they are accessible from other executables. *package*|*class* can be either a fully qualified class name or a package specification (for example, `java.awt.*`).
`-g6` *file*`.tdb`	Add debug information to output files so the result can be debugged. Use *file*`.tdb` as the debug "type database" file. This is the format you should use to debug in the Symantec Visual Cafe environment.
`-5|-6`	Generate Pentium (p5) or Pentium Pro (p6) code.

Option	Description
`-profile`	Do performance profiling.
`-L/`*switch*	Pass `/`*switch* to the linker. To see the valid options, type `link /?`.
`-`*file*`.lib`	Link in the library file. Used with the `link` switch.
`-`*file*`.res`	Link in the resource file. Used with the `link` switch.
`-`*file*`.def`	Link in the module definition file. Used with the `link` switch.

Environment variables

SJ uses the following environment variables, either set at the console command-line prompt or in the `SC.INI` file:

Variable	Description
`CLASSPATH`	A semicolon-separated list of paths (similar to the `PATH` environment variable) where SJ looks for classes. The default `CLASSPATH` is the current directory. This can be overridden using the `classpath` switch.
`PATH`	Search path for executable files if they are not found in the same folder where SJ resides.

How SJ searches for programs

To search for programs, SJ first searches in the directory where `SJ.exe` was found. If the programs are not found there, the `PATH` is searched.

How SJ searches for imports in SC.INI

SJ looks for the imports your program requires through the `CLASSPATH` or `PATH` settings. It's important that SJ find the correct imports if you want your build to be correct.

SJ looks in the folder where `sj.exe` resides for the `SC.INI` file. `SC.INI` is a text file that contains environment variable settings that are similar to settings you might find in `AUTOEXEC.BAT`:

```
;Comments are lines where the first non-blank character
; is a ';'
 [Environment]
```

```
CLASSPATH=C:\VisualCAFE\BIN

;Note that %PATH% gets replaced by the previous value of
; the environment variable PATH.
PATH=C:\VisualCAFE\BIN;%PATH%
```

The special environment variable `@P` is replaced with the path to the `SC.INI` file. For example, the previous lines can be replaced with the following lines if `SC.INI` is in `C:\SC\BIN`:

```
[Environment]
CLASSPATH=%@P%..\LIB
PATH=%@P%..\BIN;%PATH%
```

This makes the settings in `SC.INI` independent of where the SJ directory tree is installed.

If `SC.INI` isn't there, no error results. This feature helps you avoid cluttering up `AUTOEXEC.BAT` with environment variable settings. It also makes running SJ independent of any existing environment variables set for other tools.

The environment settings in `SC.INI` do not prefix, augment, or append any existing settings in the environment. They replace the environment settings for the duration of running the IDDE or the compiler. For example, to use `SC.INI` to append a `CLASSPATH` path to the existing `CLASSPATH` path, you can use:

```
[Environment]
CLASSPATH=%CLASSPATH%;C:\VisualCAFE\LIB
```

Viewing compiler messages

When an error or warning occurs while the compiler compiles or recompiles a source code file, Visual Cafe displays the compiler's message in the Messages window. The Messages window lists compilation errors and warnings for all files in a project.

Each error or warning is displayed on two lines: the first lists the file and line number where the error or warning occurred, and the second gives the message itself.

The most recent messages are displayed at the bottom of the list. If a file that had errors or warning messages is recompiled, the existing messages are deleted from the window and any new messages are added at the end of the list.

The Messages window opens automatically any time errors are detected during compilation. You can also open the window by choosing Messages from the View menu. The contents of the Messages window are saved when the window is closed and are displayed again when the window is next opened.

Compiler errors

A compiler error occurs when the compiler does not understand a portion of your source code. Even in the simplest program, it's easy to write code that will result in a compiler error.

There is a difference between compiler errors and programming errors. For example, if source code fails to compile successfully, you have a compiler error. If the source code compiles successfully but does not run as you expect, you have a programming error. It is possible for a program to compile successfully and still not run. To fix the logic of your program to get it to run the way you want, you must debug the program (for details, see Chapter 6, "Debugging Your Program").

The most common compiler errors result from improper syntax. The compiler can only understand source code that has been formatted correctly. Some of the most common mistakes are:

- Misspelled words (keywords, methods)
- Missing or incorrect separators (periods, braces, semicolons)
- Improper use of case (capitalization)

One compiler error can generate multiple error messages. For example, you could misspell the name of a method that you call from your program. Although you misspelled it only once, your program may reference the misspelled method many times. The result is multiple compiler errors from a single typing error.

Sometimes the compiler error message will indicate exactly what caused the problem. At other times, you may have to inspect the troublesome line of code character-by-character to locate the source of the problem.

Using Visual Cafe to locate compiler errors

When the compiler encounters an error in the source code, the error is displayed in the Messages window (see “About the Messages window” on page 6-5 for details). Double-click the error message to jump to the line that contains the error. The error is highlighted in the Source window. The integration between the Messages window and the Source window saves you from searching through every line of your code to find an error.

Specifying whether builds are debug or final

You can specify how you want to compile your Java code, either for a debug build or for a final build.

By default, the compiler includes debug information for debug builds, while no debugging information is included and Java optimizations for speed and compactness are performed for final builds. Debugging information enables you to use all Visual Cafe debugging features when you debug your Java programs, but makes the compiled code larger.

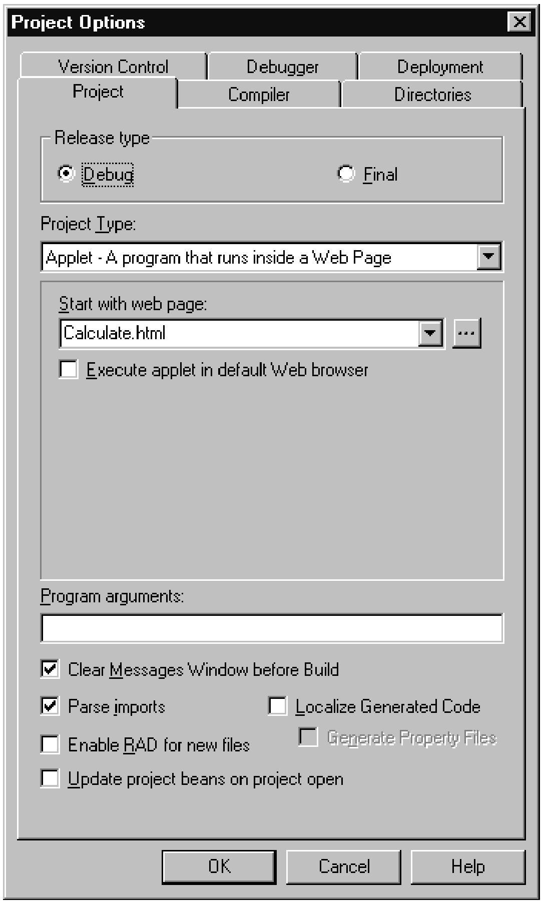
In the Project Options dialog box, the Compiler and Directories tabs show the release type option sets. For more information on the options, see “Setting compiler options” on page 5-57, “Specifying source-file search paths for a project” on page 3-62, and “Specifying class-file search paths for a project” on page 3-60.

To set the release type to debug or final:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.

3 In the Project Options dialog box, click the Project tab (if it's not already active).



Release type: [✓ Debug] Final

Project Type: Applet - A program that runs inside a Web Page

Start with web page: Calculate.html

☐ Execute applet in default Web browser

Program arguments:

☑ Clear Messages Window before Build

☑ Parse imports

☐ Localize Generated Code

☐ Enable RAD for new files

☐ Generate Property Files

☐ Update project beans on project open

4 Select one of the following options:

Option	Description
Debug	Builds an executable that contains debugging information.
Final	Builds a more compact executable that is optimized and contains no debugging information.

You can change the default options for each type of release, as described in the next procedure.

5 Click OK.

The change takes effect the next time you compile your Java program.

You can set two different categories of project-level options: one set of options for debug builds, and another set of options for final builds.

To change options for debug and final releases:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Project tab (if it's not already active).
4. Select either Debug or Final to be the release type.
5. Click the Compiler and Directories tabs and set the options for that release type.

   For more information, see "Setting compiler options" on page 5-57 and "Specifying source-file search paths for a project" on page 3-62.
6. Click OK.

   The changes take effect the next time you compile your Java program.

Specifying whether to parse imports

Visual Cafe ships with the standard Java package imports from the Sun Microsystems Java Development Kit (JDK) in a preparsed form. These imports might be required by applets and applications in order to execute.

By default, if you import other packages (including the Symantec packages), Visual Cafe will parse them so you can, for example, look at them in the Class Browser, view them in the Project window, and see them in the Form Designer, if applicable. Not parsing imports requires less computer resources and makes Visual Cafe run faster. You might want to disable import parsing if, for example, you import a lot of third-party packages and your computer runs very slowly as a result of the parsing.

To specify whether to parse imports:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.
3. In the Project Options dialog box, click the Project tab.

4. Select Parse imports to specify that imports be parsed automatically as you work with the project. Or deselect it to not parse these imports.
5. Click OK.

The change takes effect immediately if you select the option. If you deselect the option, no more imports are parsed.

Specifying whether to clear messages before a build

By default, Visual Cafe clears the Messages window before each build. This makes it easier to see what messages apply to the current build. However, you can specify that the window not be cleared so you can see messages from previous builds and compare build messages.

To specify that the Messages window not be cleared before each build:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.
3. In the Project Options dialog box, click the Project tab.
4. Select Clear Messages window before build to clear the Messages window before each project build. Deselect this option to not clear the window.
5. Click OK.

The change takes effect the next time you compile your project.

Specifying the output folder for a project

You can specify where Visual Cafe stores compiler output files, such as class files, for the current project. You can also specify the system Javadoc folder and the project documentation folder. For more information on Javadoc, see “About Javadoc” on page 4-59.

Note: The default output folder is now the project folder. Previous versions of Visual Cafe generated class files in the project folder, whereas now they are generated in the source folder.

For example, imagine that you have your project located in a folder `c:\projects`. Within that folder you have a package called `source`, and this package contains your source files (`c:\projects\source`). In previous versions of Visual Cafe, your compiled class files would be generated in the project folder (in this case, `c:\projects`). Now your compiled class files are generated in your source folder (in this case, `c:\projects\source`). So, if you were compiling a source file called `althea.java`, the old path to the class file would be `c:\projects\althea.class`, whereas now it would be `c:\projects\source\althea.class`.

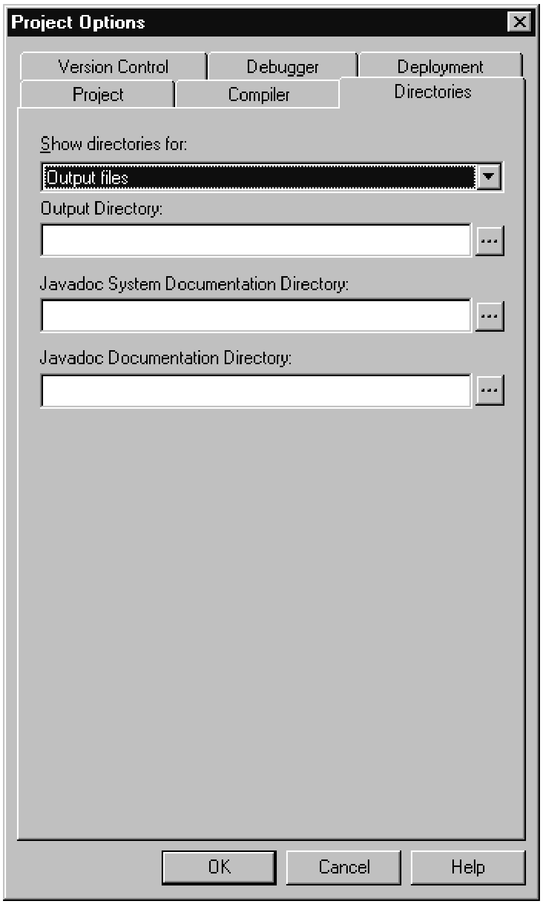
Note: When you run a project in the AppletViewer, the output folder is temporarily added to the class path.

To specify the output folder for a project:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Directories tab.

4 In the list under Show directories for, choose Output files.



Show directories for: Output files

Output Directory:

Javadoc System Documentation Directory:

Javadoc Documentation Directory:

The output directory appears. If no output directory is specified, Visual Cafe uses the default: a `.class` file is placed in the same location as the corresponding `.java` file. If you do specify an output folder, all output files are placed in this folder and any package file hierarchy structures are created as well.

5 Type a folder and full path in the field, or select a folder by clicking the Browse button (...) that appears in the field.

6 Click OK.

The change takes effect the next time you compile your project.

Using different Java virtual machines in Visual Cafe

Visual Cafe lets you use different Java virtual machines (Java VMs) so you can execute, compile, and debug with them from within the Visual Cafe environment. Note that the internal virtual machine used by Visual Cafe (for the Form Designer, for example) continues to be the default virtual machine supplied with Visual Cafe.

Caution: If your project's imported class files do not match the JDK version used by the virtual machine, your virtual machine can crash during debugging with a debugger supported by the Visual Cafe environment. If so, you can deselect the Supports Direct Debugging option, and optionally specify another debugger for the VM Other Debugger Executable parameter. In this case, when you choose Run in Debugger from the Project menu, Visual Cafe simply executes the Java program in the virtual machine you specified; if you specified an alternate debugger, it launches the debugger as a separate process.

To access the Environment Options Virtual Machines view:

From the Tools menu, choose Environment Options, and select the Virtual Machines tab.

To choose a virtual machine:

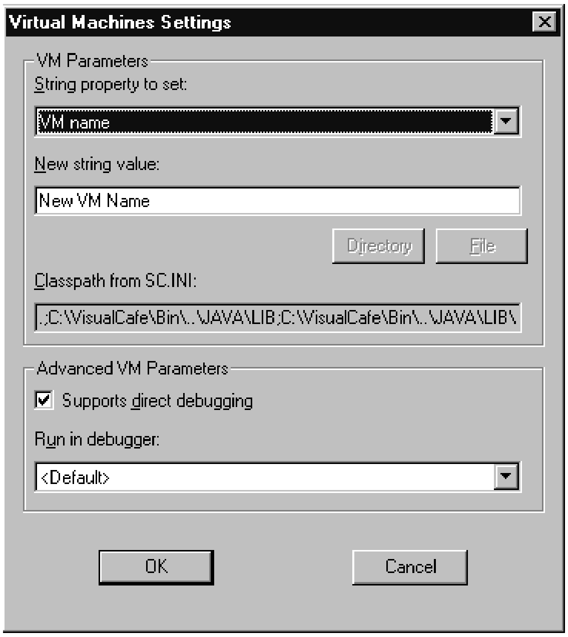
1. Choose the virtual machine from the drop-down list.
2. Click OK in the Virtual Machines view.

   The change takes effect immediately.

To add a new virtual machine:

1. Click New.

The Virtual Machines Settings dialog box displays.



2 Specify the parameters you want. To specify a parameter, choose an option from the String property to set drop-down list box. Only the VM name and the VM executable parameters are required, and it is recommended that you also supply a VM JDK Source Path.

Parameter	**Description**
VM name	A unique name you use for the virtual machine.
VM manufacturer	The name of the creator of the virtual machine. Currently used for informational purposes only.

Parameter	Description
Compiler classpath	Location of the files your program depends on for compilation (for example, the class files for JFC/Swing components). The compile class path is appended to the class path for the project. If the Compile Classpath field is left blank, the class path specified in the `sc.ini` file is used. **Note:** You need to add `symbeans.jar` to this class path if you use Symantec Beans. Otherwise, your source files might not compile. If you specify a virtual machine that does not support JAR files (such as the Microsoft virtual machine), you need to unjar `symbeans.jar` and add the directory location to the class path.
Use VM executable (fully specified path of the VM executable)	The virtual machine executable and the path to it.
Classpath specifier (VM executable classpath prepend string)	Command-line specifier for the class path, usually `-classpath`. The Microsoft virtual machine may use `/cp:a` (or `/cp:p` or `/cp`). Specify the class path in the VM Executable Classpath field.
Classpath (classpath for use by the VM executable)	The class path to be appended to the VM Executable Classpath Prepend String parameter. **Note:** You need to add `symbeans.jar` to this class path if you use Symantec beans. Otherwise, you might get Runtime Class Not Found exceptions during execution. If you specify a virtual machine that does not support JAR files (such as the Microsoft virtual machine), you need to unjar `symbeans.jar` and add the directory location to the classpath.

Parameter	Description
VM path	The search path for preparsed JDK information and a Javadoc executable. Visual Cafe ships with the standard Java package imports from the Sun Microsystems Java Development Kit (JDK) in a preparsed form (the files `jdk.ve2` and `jdk.vep` in the `Bin` directory). This information is used by Visual Cafe while you are programming in its environment (for example, so you can view the JDK in the Class Browser, Project window, and Form Designer). The JDK version used by a virtual machine might be different than that supplied with Visual Cafe. In this case, you want to supply the correct version of the JDK in preparsed form. If there is no preparsed information in the VM path, Visual Cafe asks if you want to initiate preparsing; if you do not, the default preparsed information present in the Visual Cafe environment is used. If you preparse, the `jdk.ve2` and `jdk.vep` files are stored in the VM path location. If you do not specify anything in the Javadoc Executable field, Visual Cafe searches for this executable on the VM path. If the executable is not found, the default `javadoc.exe` is used.
Other command options (other execution options for the VM)	Other command-line options.
Use debugger executable	The executable name and path to an alternate debugger that is used when the Supports Direct Debugging option is not selected.
Javadoc executable	The executable name and path to a Javadoc tool. If you do not specify anything in the Javadoc Executable field, Visual Cafe searches for this executable in the VM path. If the executable is not found, the default `Javadoc.exe` is used.

Parameter	Description
VM's JDK source path	The root of the source files for the JDK your virtual machine uses. You should specify this parameter, or you might have duplicate information in the Class Browser for the JDK classes.

3 In the Advanced VM Parameters area, specify the options that you want.

Parameter	Description
Supports direct debugging	Select this option if this virtual machine can be used with a debugger currently available from the Visual Cafe environment. Then specify the debugger you want in the Run In Debugger field. If you deselect the Supports Direct Debugging option, when you choose Run in Debugger from the Project menu, Visual Cafe simply executes the Java program in the virtual machine you specified. You can optionally specify another debugger in the Use Debugger Executable field. In this case, when you run in the debugger, Visual Cafe launches the debugger as a separate process. When you choose Run in Debugger from the Project menu, and the Supports Direct Debugging option is deselected, you receive a dialog box that lets you specify options for the debugger: specifically, the password for connecting to the virtual machine. The password is generated automatically by the virtual machine and is unique every time you run the virtual machine. The password appears in the Messages window.

Parameter	Description
Run in debugger	The debugger that is used when the Supports Direct Debugging option is selected. This list of debuggers is the same as those listed in the project options for applets. If you specify the default value and run an application, the AppletViewer is used. If you specify Netscape Navigator and run an application, Visual Cafe instead uses the default virtual machine. If you specify the default value and run an applet, the virtual machine specified in your project options is used; otherwise, the virtual machine specified in your project options is ignored.

4 Click OK in the Virtual Machine Settings dialog box, then click OK in the Virtual Machines view.

The change takes effect immediately.

To edit the settings of a virtual machine:

1 Choose the virtual machine from the drop-down list.

2 Click Modify.

3 Specify the options you want.

4 Click OK in the Virtual Machine Settings dialog box.

The change has been made.

5 When you're finished making changes, choose the virtual machine you want to use from the drop-down list in the Virtual Machines view, then click OK.

To remove a virtual machine:

1 Choose the virtual machine in the drop-down list, then click Delete.

2 Click OK.

The change takes effect immediately. Note that you cannot remove the virtual machine that was specified when you launched Visual Cafe.

Setting internal VM environment options

From the Internal VM tab of the Environments Options dialog box, you can enable, disable, and perform a stack trace on the Just-In-Time compiler (JIT) of the internal virtual machine of Visual Cafe. In addition, you can set the class path, Java heap, and stack size parameters here, instead of having to edit the `sc.ini` file.

Note: Making changes in the Internal VM tab causes Visual Cafe to rewrite your `sc.ini` file. Visual Cafe adds in the specified information, so existing information is not lost.

Most changes that you'll want to make to the `sc.ini` file you can do in the Internal VM tab, but you can also manually edit the file yourself. For more information, see "Setting environment variables in the sc.ini file" on page 3-72. All changes that you set in the Internal VM tab can also be specified in the `sc.ini` file.

Note: By default, Visual Cafe does not read the class path setting for Windows, as specified in the `autoexec.bat` file. This is a change from earlier versions of Visual Cafe. You can, however, add the setting yourself. For more information, see "Inheriting the class path from the Windows environment" on page 3-71.

The JIT in the VM converts Java bytecode to native machine code; the machine code provides performance comparable to that of C or C++. Turning on the JIT makes Visual Cafe run the fastest. However, when an exception occurs, Visual Cafe returns to you the type of exception only, and not where it happened.

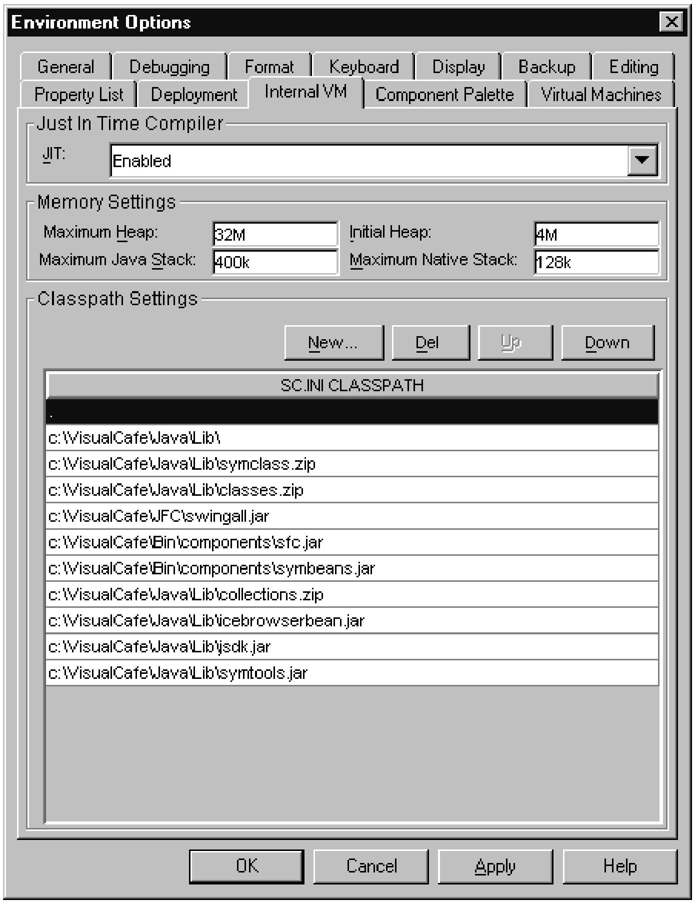
Realize that disabling the JIT makes Visual Cafe run more slowly. More importantly, when an exception is thrown, if the JIT is disabled you get a full stack trace, including the exact location where the exception occurred (file, class, and line number).

Turning on the stack trace means that the JIT is still enabled, but Visual Cafe runs a little more slowly. When an exception occurs, you get the same information as when the JIT is disabled, but without the file name and line number.

Note: You need to restart Visual Cafe for the changes made in the Internal VM tab to take effect.

To open the Environment Options Internal VM view:

Choose Environment Options from the Tools menu, then click the Internal VM tab.



Just In Time Compiler

JIT: Enabled

Memory Settings

Maximum Heap: 32M
Initial Heap: 4M
Maximum Java Stack: 400k
Maximum Native Stack: 128k

Classpath Settings

SC.INI CLASSPATH
.
c:\VisualCafe\Java\Lib\
c:\VisualCafe\Java\Lib\symclass.zip
c:\VisualCafe\Java\Lib\classes.zip
c:\VisualCafe\JFC\swingall.jar
c:\VisualCafe\Bin\components\sfc.jar
c:\VisualCafe\Bin\components\symbeans.jar
c:\VisualCafe\Java\Lib\collections.zip
c:\VisualCafe\Java\Lib\icebrowserbean.jar
c:\VisualCafe\Java\Lib\jsdk.jar
c:\VisualCafe\Java\Lib\symtools.jar

To enable, disable, or perform a stack trace on the JIT:

1 In the JIT list in the Internal VM view, choose Enable, Disable, or Stack Trace.

2 Click Apply or OK to save the change.

You need to restart Visual Cafe for the change to take effect.

To set the class path:

1 In the Internal VM view, modify the class path list as needed:

- ❖ To change the order in which directories are searched, select a directory and move it with the Up Arrow or Down Arrow buttons.
- ❖ To delete a directory from the list, select the directory and click Delete.
- ❖ To add a directory to the list, select the blank entry (marked by an empty box) at the bottom of the list and type the directory name, including the full path. Or, click the New button (located above the text box), then select a directory by clicking the Browse button (...) that displays in the field. You can also use the New button to insert a new entry above the selected entry.

Tip: You can specify environment variables, such as `%CLASSPATH%`, by choosing a directory and editing it.

2 Click Apply or OK to save the changes.

You need to restart Visual Cafe for the change to take effect.

To set heap and stack size parameters:

In the Internal VM view, modify variables as needed. (Instead, you could modify the `sc.ini` file.)

Field	Description	Variable in sc.ini
Maximum Heap	The maximum Java heap size (also called the memory allocation pool or garbage collected heap) for the internal VM. A larger value means that Visual Cafe runs faster and with fewer issues that can result from a small heap size. (Default: 32 MB; Minimum: 1000 bytes or Initial Heap Value, whichever is greater.)	`VCAFE_IVM_mx`*n*
Maximum Java Stack	The maximum Java stack size for any thread. (Default: 400 KB; Minimum: 1000 bytes.)	`VCAFE_IVM_oss`*n*

Field	Description	Variable in sc.ini
Initial Heap	The initial Java heap size (also called the memory allocation pool or garbage collected heap), which affects how much memory is taken from your operating system at startup. (Default: 4 MB; Minimum: 1000 bytes; Maximum: Maximum Heap value.)	`VCAFE_IVM_ms`*n*
Maximum Native Stack	The maximum native stack size for any thread (the stack size is sued by C code). (Default: 128 KB; Minimum: 1000 bytes.)	`VCAFE_IVM_ss`*n*

Note: The value (*n*) is in bytes. Append a *k* to specify kilobytes or an *m* for megabytes.

Deploying your project

You can easily deploy applets and applications in JAR, CAB, or ZIP files, or to a directory. A **CAB file** is a single file created to hold a number of compressed files, for use in Microsoft program development. Whether you're deploying to a JAR, CAB, ZIP, or directory, you're deploying to a **deployment target**. Use Visual Cafe's deployment features to simplify your deployment process. You can also easily figure out what class files your program needs, as well as configure a Web server where you'll put your files.

When you deploy to a deployment target, Visual Cafe will do so despite any compiler warnings.

The topics in this section include:

- Deploying your applet
- Deploying your application
- Configuring UNIX-based Web servers

For information on deploying native applications or DLLs, see "Linking native Win32 applications" on page 11-4, and "Registering DLLs using SNJREG" on page 11-19.

Note: You cannot deploy Swing components along with AWT-based components in the same archive file. To deploy Swing programs, you need to separately provide `swing.jar`. This will prevent your archive files from getting too large. For more information on using Swing in Visual Cafe, see Chapter 8, “Working with JFC/Swing Components.”

Deploying your applet

After you complete an applet in Visual Cafe, you’re ready to deploy it on a Web site. This section provides some general guidelines for deploying applets; you need to know how your particular Web site is set up to get your applet up and running. Because your project can contain more than one applet that appears in related Web pages, these guidelines are for setting up a project that contains one or more applets.

Caution: The classes needed by Visual Cafe components are stored in `symbeans.jar`, `symtools.jar`, and `sfc.jar`. You do not want to deploy using `symbeans.jar`, `symtools.jar` or `sfc.jar`, because it will affect the size and performance of your applets. Instead, you want to deploy using just the classes needed by the components in your applets.

`symbeans.jar`, `symtools.jar` and `sfc.jar` contain Beans by Symantec; `symbeans.jar` contains the older AWT-based Beans, and `sfc.jar` contains the newer and preferred Swing-based Beans. We recommend that you use `sfc.jar` rather than `symbeans.jar`, but for now Visual Cafe still needs both in order to work properly. `sfc.jar` also contains some useful generic, non-Bean classes that aren’t tied to Visual Cafe. `symtools.jar` contains some useful command-line utilities.

Netscape Communicator/Navigator (with or without the Java plug-in), Internet Explorer 4.0 (with or without the Java plug-in), and all versions of HotJava support the archive HTML tag.

To deploy your applets in a JAR, CAB, ZIP, or directory:

1. Use the Project menu’s Deploy command to create an archive, such as a JAR, ZIP, or CAB file, that contains your applets and all supporting files, including Symantec class files. See “Adding external files to a JAR” on page 5-55 for more information.

Tip: Your applets should use relative URLs for graphics files so you can easily move your applets to different computers.

2 Add the variable `ARCHIVE="`*name*`.jar"` or `"`*name*`.zip"`to the applet tags in your HTML files. You can specify multiple JAR files by delimiting them with a comma. If you're using CAB files, make sure you use the correct tag for Internet Explorer.

  If you're deploying to a directory, you don't need to perform this step.

Note: Not all Web browsers support multiple JAR files; however, the most recent version of Netscape Communicator/Navigator, Microsoft Internet Explorer, and Sun Microsystems HotJava do support multiple JAR files.

Tip: Remember that the applet tags specify where your applets are located relative to the location of the HTML files. Therefore, you must place the archive file in the same relative location.

3 On your local computer, test your Web pages by opening them in a Web browser from outside of the Visual Cafe environment.

4 Put your archive file and HTML files in a directory on the Web server, as they appeared in your Visual Cafe project directory.

5 After you've completed the setup of your Web site, test your Web pages from remote computers.

  You can also test your applet with different operating systems and Web browsers.

Deploying your application

After you complete an application in Visual Cafe, you're ready to deploy it.

Mainly, the difference in deploying an application, as compared to an applet, is that you have to make sure that the associated JAR file is in your class path, and that you run your application outside of Visual Cafe differently than you would an applet. Also, with an applet you have to specify an HTML file and the associated HTML tags — you don't have to do this when deploying an application.

For information about deploying native applications, see “Deploying native Win32 applications, DLLs and libraries” on page 11-5.

Requirements for different applications vary. However, some general guidelines are provided below.

To deploy your application in a JAR, CAB, ZIP, or directory:

1 Use the Project menu’s Deploy command to create a JAR file containing your application and all supporting files, including Symantec class files. See “Adding external files to a JAR” on page 5-55 for more information.

Tip: Your application should use relative URLs for graphics files so you can easily move your application to different computers.

2 On your local computer, test your application by running it from outside the Visual Cafe environment.

To start your application, the JAR file must be in the class path, as set in the `sc.ini` file. You can set your Visual Cafe class path in the Environment Options dialog box; for more information, see “Setting internal VM environment options” on page 5-28.

Then you can run your application:

`java` *application-name*

`java` invokes `java.exe` and, if necessary, includes the complete path (for example, `\visualcafe\java\bin\java)`. Your application name is the same as the name of the frame for the main application window, without the .`class` extension; note that the name is case-sensitive and you might need to include the complete path.

For example:

```
set classpath=%classpath%;Amazing.jar
\visualcafe\java\bin\java AmazingTour
```

3 Test your application from remote computers.

You can also test it with different operating systems.

Note: You need to provide — or your users need to obtain — the Java Virtual Machine and standard Java class files. The Symantec Java Virtual Machine must be licensed from Symantec.

Configuring UNIX-based Web servers

The following instructions for system administrators are guidelines only; they've been tested with Apache. After the Web server is configured correctly, all applets in the user's home HTML directory will have access to the Visual Cafe classes.

Caution: The classes needed by Visual Cafe components are stored in `symbeans.jar`, `sfc.jar`, and `symtools.jar`. You do not want to deploy using `symbeans.jar`, `sfc.jar`, or `symtools.jar` because it will affect the size and performance of your applets. Instead, you want to deploy using just the classes needed by the components in your applets. These classes can be packaged in a JAR file or not.

To configure a UNIX-based Web server:

1 Create a UNIX directory, such as `/home/symantecclasses`. Make sure all users have read access to the directory.

2 Expand `symbeans.jar`, preserving the directory structure, and copy it to this UNIX directory.

For more information, see "Expanding a JAR file" on page 5-56.

3 Create a symbolic link from `/home/symantecclasses/symantec` to the user's home HTML directory. For example:

```
ln -s /home/symantecclasses/symantec
/home/joeuser/public_html
```

`joeuser` can now run applets from Web pages without copying the full Symantec class structure to his `public_html` directory.

If you have the Visual Cafe Database Edition, see the *Visual Cafe Database Developer's Guide* for more information.

Deploying from the command line

You can deploy your programs from the command line using the `com.symantec.itools.tools.archive.Archiver` utility. You can run this program on any operating system, since it's written in Java. This is useful if you want to make use of Visual Cafe's deployment tools outside of the Visual Cafe environment, or if you're deploying from another operating system. With the Archiver tool, you can deploy to JAR files, CAB files, ZIP files, or directories.

Note: When using the Archiver tool to deploy, you'll deploy files to a local machine; with this tool you cannot deploy by way of FTP.

The source code for the archiver is included so that users can modify or subclass based on their particular requirements. The Archiver tool provides a simple wrapper around `JAR`, `CAB`, and `JAVAKEY` executables, thereby gathering all settings for deployment into one tool. The architecture is also extensible so that you can add new Archive types (for example, `tar.Z` support). The Archiver also has a programmatic API so it can be embedded into your applications that need to create archives. Using the Archiver tool is also helpful in case you're doing automated builds, so that you don't have to open Visual Cafe in order to create a deployment target.

To deploy to a JAR, ZIP, CAB, or directory by using the command-line Archiver tool:

1. Figure out what files or entries in the `CLASSPATH` will make up the contents of the deployment target.
2. Copy all of the files and entries to a temporary directory.
3. Run the Archiver tool to create the deployment target.

To use the Archiver tool by starting with a generated response file:

1. In Visual Cafe, deploy to an archive or directory (if deploying to a JAR, then generate a manifest file). For more information, see "Setting the archive type, signer tool, and protocol" on page 5-40.

   Select the option to create a response file. For more information, see "Setting advanced deployment options for a project" on page 5-51.
2. Modify the response file with options for the Archiver tool.
3. Run the Archiver tool with the response file. For example, at the command line, you'd type the following:

   `com.symantec.itools.tools.archive.Archiver`
   `@`*responsefilename*`.rsp`.

Setting command-line archiving options

The following are the options that you can use to configure the command-line archiving tool:

Option	Description
`-help`	Displays information on all command-line archiving options.
`-type [jar | zip| cab| dir]`	Specifies the type of deployment target to be created. Currently, JAR, ZIP, CAB, and directory are supported.
`-out` *filename*	Specifies the location of the resulting deployment target.
`-tempdir` *directoryname*	Specifies the location of the temporary directory to store the files that make up the archiver. This location is used by the Archiver tool to make the deployment target.
`-classpath {[`*directoryname* | *jarname* | *zipname* `|; ...}`	Specifies the CLASSPATH to search for entries. This can be a list of JAR files, directories, or ZIP files, and each item is separated by a semicolon (;).
`-files {`*filename*`, ...}`	Specifies a list of files to be archived, and each file is separated by a space. For example, `c:\projects\file.gif`.
`-entries {`*entryname*`, ...}`	Specifies a list of entries on the CLASSPATH that are to be archived. These entries are individual files. For example, you could specify `java/lang/Object.class.`
`-signer [sun]`	Specifies the signing tool to use, and its location. Currently Sun is supported. For example, you could specify `c:\VisualCafe\Java\Bin.`
`-overwrite [true |false]`	If the file specified by `-out` exists, then it gets overwritten. This prevents you from accidentally replacing a deployment target. Also, this is useful in case you are recreating a deployment target, and thus want to replace a pre-existing one.
`-debug [true|false]`	Logs any messages to the file specified by `-debuglog`.

Option	Description
`-debuglog` *filename*	Specifies the file to log messages to.
`-mstools` *directoryname*	Specifies the directory where the CAB tool (`CABARC.exe`) is located.
`-suntools` *directoryname*	Specifies the directory where the `JAR.exe` and `JAVAKEY.exe` are located.
`-JA` "*list of tool specific arguments*"	The command line specific to the archiver tool being used. Enter in arguments (in quotes) that work with the specified archiver tool.
`-JS` "*list of tool specific arguments*"	The command line specific to the signing tool being used. Enter in arguments (in quotes) that work with the specified signing tool.
`@ [`*responsefilename*`]`	A file that contains some or all of the command line options listed above. This is helpful in case you want to keep some of your settings each time you use the Archiver tool.

Here is an example of commands you would type to use the command line archiving tool:

```
java com.symantec.itools.tools.archive.Archiver -type jar
 -out c:\temp\foo.jar -tempter C:\temp\jardir -overwrite
 true -JA "-cf0m c:\temp\foo.jar c:\temp\foo.mf *.*" -signer
 sun -JS "-gs c:\temp\signing_directive c:\foo.jar"
 -suntools c:\VisualCafe\java\bin\ -entries mypackage/
 JApplet1.class
```

This example creates a JAR file called `foo.jar` in the `c:\temp` directory, uses `c:\temp\jardir` as the working, temporary directory, will overwrite `foo.jar` if it already exists, passes `-cf0m c:\temp\foo.jar c:\temp\foo.mf *.*` to Sun's command-line JAR tool, signs the JAR using Sun's signing tool, passes `-gs c:\temp\signing_directive c:\foo.jar` to Sun's signing tool, indicates that Sun's tools are located in the `c:\VisualCafe\java\bin` directory, and includes the `mypackage/JApplet1.class` entry from the class path.

Setting deployment options

After you complete an applet or application in Visual Cafe, you're ready to deploy it. You can specify deployment-related options from the

Deployment tab of the Project Options dialog box; these options affect deployment on a per-project basis.

You can also set deployment options that affect all projects. For more information, see “Setting deployment options for all projects” on page 5-52.

Setting deployment options for a project

You can specify deployment options for a project (discussed in this section), or for all projects. For more information, see “Setting deployment options for all projects” on page 5-52.

Use the Deployment tab in the Project Options dialog box to set deployment options for a single project. The project-level deployment options are divided into five sections: General, Files, Archiver, and Advanced. You can see these in the Deployment Category drop-down list box in the Deployment tab.

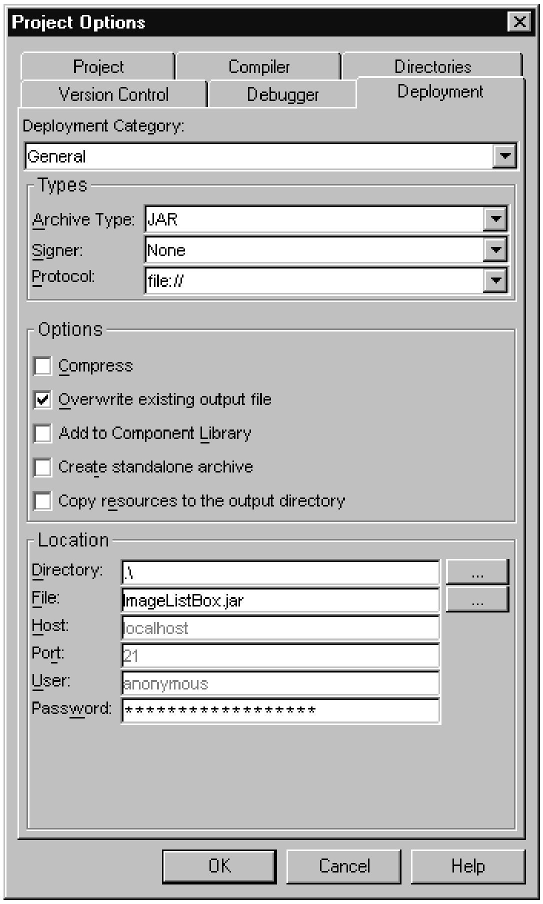
You can set the following project-level deployment options:

- Setting the archive type, signer tool, and protocol
- Specifying what files to include in your archive or directory
- Setting archiver options for JAR files
- Setting archiver options for CAB files
- Setting advanced deployment options for a project

These tasks are covered in this section.

Setting the archive type, signer tool, and protocol

When you select the deployment category of General in the Project Options dialog box, the Deployment page displays, as shown here:


Project Options

Project | Compiler | Directories | Version Control | Debugger | Deployment

Deployment Category: General

Types

Archive Type: JAR

Signer: None

Protocol: file://

Options

- [ ] Compress
- [x] Overwrite existing output file
- [ ] Add to Component Library
- [ ] Create standalone archive
- [ ] Copy resources to the output directory

Location

Directory: .\

File: ImageListBox.jar

Host: localhost

Port: 21

User: anonymous

Password: *******************

OK | Cancel | Help

The General option allows you to specify the most frequently-used deployment options that apply to a project

Signing allows the user to alter the permissions that a Java applet can have on the user’s system. Signing is currently available with tools from Sun Microsystems. For details, see Sun Microsystems’ documentation.

The user will need to set up program-specific information, such as the key and signer. The tool needs to work from the command line before it will work in the environment.

To set the archive type:

1. Under the Types field, select the type of archive you want to deploy to. Available choices include JAR, ZIP, Directory, or CAB.

   CAB files are only available if you've specified them as an option for all projects. For more information, see "Setting deployment options for all projects" on page 5-52.

2. If the archive can be signed you can choose from a list of available signers.

   Available signers are specified in the Environment Options dialog box. For more information, see "Setting deployment options for all projects" on page 5-52.

3. Select the protocol by which the archive will be deployed. You can deploy by way of FTP, or by way of the file protocol.

   If you're deploying to your local machine, select File as the protocol. If you're deploying to an FTP server, select FTP as the protocol.

   The FTP option will only be available if you've specified it in the Environment Options dialog box. For more information, see "Setting deployment options for all projects" on page 5-52.

4. In the Options area, select the options you want.

Select this...	To do this...
Compress	Compresses the archive. This option is available with JAR and CAB files.
Overwrite existing file	Overwrites a previous version of the archive or directory
Add to Component Library	Any JavaBeans that are deployed are also added to the Component Library.

Select this...	To do this...
Create stand alone archive	If this item is selected, Visual Cafe will deploy all files that your project generates, plus any additional files that are needed for the program to run. Visual Cafe looks to the `classpath` setting for these dependencies. The class path information is generated from the setting in the Environment Options dialog box (for more information, see “Setting internal VM environment options” on page 5-28), and also from the setting for input files in the Project Options deployment page (for more information, see “Specifying class-file search paths for a project” on page 3-60). If you know that the machine you’re deploying to has one or more of these dependencies, you can explicitly remove them from the archive or directory (for more information, see “Specifying what files to include in your archive or directory” on page 5-44). If this item is not selected, the archive will be deployed with only the class files that your project generates, plus any other files that are in your project (image, properties, or HTML files, for example).

Select this...	To do this...
Copy resources to the output directory	If this item is selected, Visual Cafe will copy non-class files to the same location as the archive, thus outside of the archive. This only applies if you use the `RelativeURL` class for these resource files. For deployment purposes, you want the ability to move your program to another location and still have files be found. The optimum method to do this is to use the `getClass().getResourceAsStream(ResourceName)` method. In code that Visual Cafe generates, the `RelativeURL` class is used. Both methods specify a file relative to the Java program. For example, if `Applet1` is in a Project directory and graphics files are in an `Images` subdirectory, specify just `Images/`*filename*. Unfortunately, `RelativeURL` doesn't allow you to access resource files inside of an archive. So we provide this checkbox here to copy the resources to the same location as the archive so that the `RelativeURL` class can still find the resource files.

5 In the Location area, specify the following information:

In the Directory field, specify the directory you want to deploy to. Click the Browse button (…) to navigate to a directory.

In the File field, specify the file name of your deployed archive or directory. Click the Browse button (…) to navigate to a file and directory. If you specified the archive type of Directory, then this field is not editable.

If you're deploying by way of FTP, you need to specify the Host, Port, User, and Password information. These fields are initially set with the default FTP settings that are specified in the Environment Options dialog box, but you can override these initial settings. For more information, see "Setting deployment options for all projects" on page 5-52.

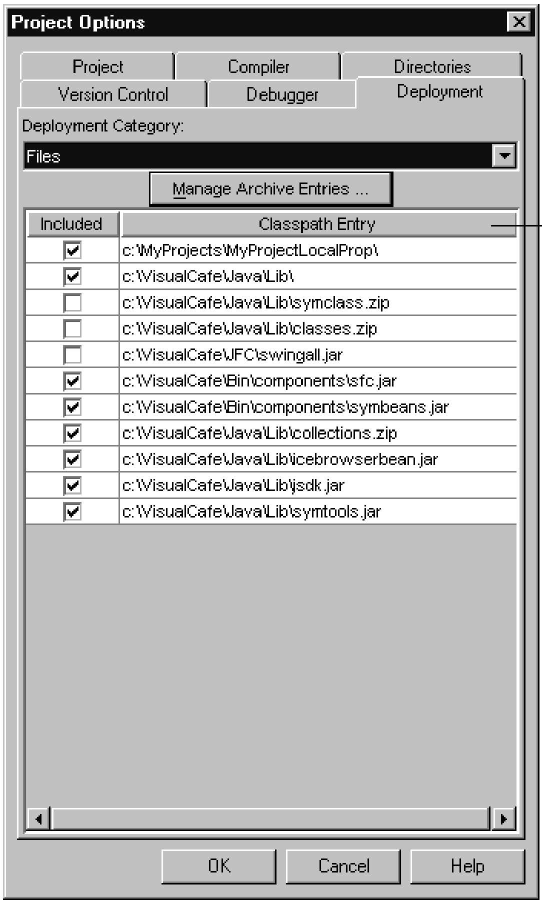
Note: The FTP password that you use here is not secure.

6 Click OK.

The changes take effect immediately.

Specifying what files to include in your archive or directory

When you select the deployment category of Files in the Project Options dialog box, the Deployment page displays:



Deployment Category: Files

Included	Classpath Entry
✓	c:\MyProjects\MyProjectLocalProp\
✓	c:\VisualCafe\Java\Lib\
☐	c:\VisualCafe\Java\Lib\symclass.zip
☐	c:\VisualCafe\Java\Lib\classes.zip
☐	c:\VisualCafe\JFC\swingall.jar
✓	c:\VisualCafe\Bin\components\sfc.jar
✓	c:\VisualCafe\Bin\components\symbeans.jar
✓	c:\VisualCafe\Java\Lib\collections.zip
✓	c:\VisualCafe\Java\Lib\icebrowserbean.jar
✓	c:\VisualCafe\Java\Lib\jsdk.jar
✓	c:\VisualCafe\Java\Lib\symtools.jar

This class path listing appears only if you selected Create stand alone archive *in the General deployment category page.*

Here you can choose to include or not include certain archives or directories, depending on what your target machine has installed.

Note: You cannot include `symclass.zip`, `classes.zip`, or `swingall.jar` in your deployment target. These are provided in the class path listing to reflect that they are listed in your class path. If you must deploy `swingall.jar`, you must do so separately.

If you know that the computer you're deploying to already has one of the archives listed, or if you're deploying one or more of these archives separately, you can choose to remove them from the current deployment archive or directory.

You can explicitly remove entries (like `jgl.jar`) if you know that the classes already exist on the target machine.

To manage files in an archive:

1. Open the Deployment page of the Project Options dialog box, and select Files as the deployment category.

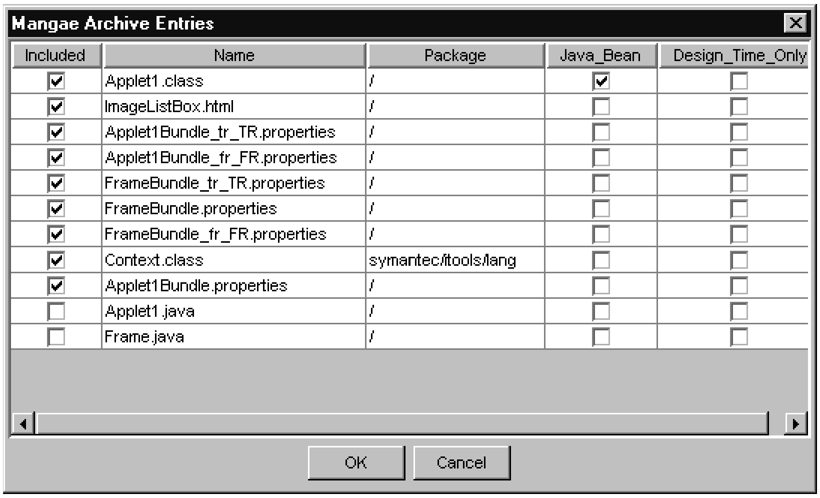
   The Deployment Files page displays.

2. If you selected the Create stand alone archive in the General deployment category page, then you'll also see a listing of archives listed in the class path.

   (Optional) To include or remove archives in your deployment target, select or deselect the checkbox in the Included column for each archive.

3. To include or remove specific files in your deployment target, click the Manage Archive Entries button.

   The Manage Archive Entries dialog box displays, which lists all the files that will be placed into the deployment target. You can choose to remove entries and set their manifest properties (manifest is for JAR only).



Mangae Archive Entries

Included	Name	Package	Java_Bean	Design_Time_Only
✓	Applet1.class	/	✓	☐
✓	ImageListBox.html	/	☐	☐
✓	Applet1Bundle_tr_TR.properties	/	☐	☐
✓	Applet1Bundle_fr_FR.properties	/	☐	☐
✓	FrameBundle_tr_TR.properties	/	☐	☐
✓	FrameBundle.properties	/	☐	☐
✓	FrameBundle_fr_FR.properties	/	☐	☐
✓	Context.class	symantec/itools/lang	☐	☐
✓	Applet1Bundle.properties	/	☐	☐
☐	Applet1.java	/	☐	☐
☐	Frame.java	/	☐	☐

To include a file in the deployment target, select the checkbox in the Included column. Deselect the checkbox to exclude a file from the deployment target.

The items in the Name and Package columns are provided for information only; the information in these columns can't be edited from this dialog box.

The `.java` files in the project are included in this listing, but are excluded from the deployment target by default. You can thereby choose to include your source files in with your deployment target, if you want.

4 (Optional) If a file is a Bean or is used only at design time, click the checkbox in the appropriate column. Specify these settings only if you're deploying to a JAR file.

If you're creating a JAR and would like to specify additional manifest tags (other than JavaBean and Design-Time Only), you can do so by providing your own initial manifest (see the following section, "Setting archiver options for JAR files," for more information).

Note: In order to add files to an archive you need to add the files to the project. This does not include entries of subprojects.

5 When you're done making changes to the file list, click OK.

Your changes to the deployment target's file list are saved.

6 In the Project Options dialog box, click OK.

Your deployment options take effect immediately.

How Visual Cafe figures out what files your program needs

You can learn which class files you need by creating a JAR file with the Visual Cafe Deployment project options or by using SJ.

Remember that when you deploy your project you need to keep the class directory structure intact; in addition, you must use the same case in the class names (class names are case-sensitive).

If you're interested in how Visual Cafe figures out which files are used by your program, read on. This information is provided for those of you want to know the details; you don't need to know this information in order to deploy your programs.

The `sj.exe` utility enables you to easily figure out which class files are used by your applet or application. After your Java applet or application is finished, enter the following command at the DOS prompt (make sure that `\visualcafe\bin` is in your path):

```
sj.exe -make -cdb mainclass.cdb -depend listname.dar
 filename.java filename.java filename.java
```

Include the names of all Java files in your project.

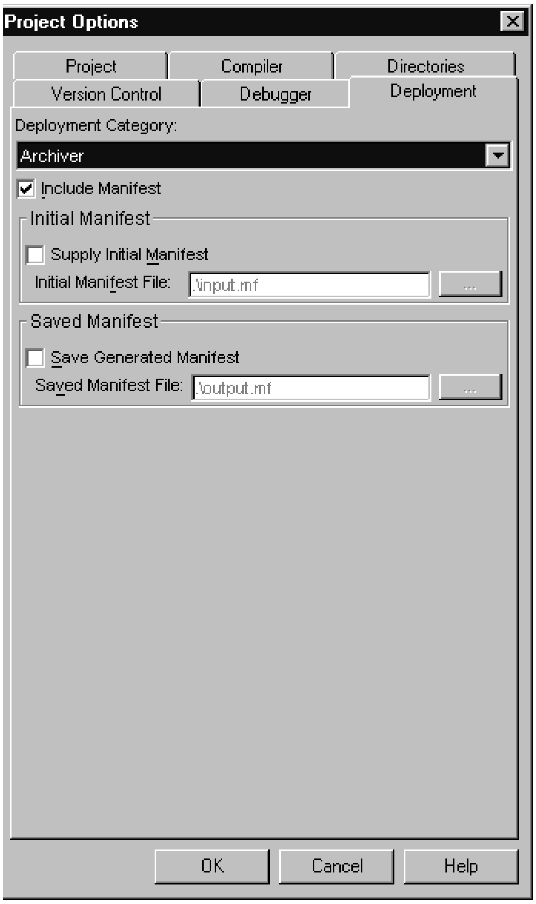
listname`.dar` is the file where the list of classes used by your applet or application and their locations will be generated. `sj.exe` also lists the name of the archive file or folder (if any) where the class file was found. This helps you figure out which class files you'd need to extract from each class archive file used by your applet or application. The file name must have a `.dar` extension.

Even though the standard `java.*` class files are logged in this file, you're not required to copy those to your Web server for your applets, as they're available with most Web browsers. For applications, the standard Java class files are part of Java Runtime Environment, which you can download from JavaSoft.

`sj.exe` takes additional command-line options, so you can add additional class paths and so on. See "Compiling from the SJ command line" on page 5-7 for more information.

Setting archiver options for JAR files

If you've previously set the archive type to JAR in the General category of the Project Options dialog box, when you select the deployment category of Archiver, the Deployment page displays:



Deployment Category: Archiver

[✓] Include Manifest

Initial Manifest

[ ] Supply Initial Manifest

Initial Manifest File: .\input.mf

Saved Manifest

[ ] Save Generated Manifest

Saved Manifest File: .\output.mf

Set options in the Archiver deployment category to specify whether you want to include a manifest in the JAR file, what you want to use as your base manifest (assuming you have one), and whether you want to save the generated manifest for use outside of Visual Cafe.

When creating a JAR, Visual Cafe uses the Sun JAR program, which is included with Visual Cafe. Symantec provides a command-line utility (`com.symantec.itools.tools.Archiver`) that allows you to easily create archives on the command line (by way of `make` on a Solaris machine, for instance).

You would specify an initial manifest to do the following:

- ◆ add tags other than `Java-Bean` and `Design-Time-Only` to an entry
- ◆ add files to your JAR without having to add them to the project

You can specify your own manifest file if you want to add specify additional tags. You might want to do this if you're using this as a project template.

If you supply an initial manifest, you can provide any information in that manifest that you'd like, and Visual Cafe will merge the manifest information that is generated into that file .

To specify manifest settings for a JAR:

1. Open the Deployment page of the Project Options dialog box, and in the General deployment category make sure the deployment category is set to JAR.
2. In the Deployment page of the Project Options dialog box, select Archiver as the deployment category.

   The Deployment Files page displays.
3. To include a manifest file with your JAR file, select Include manifest.

   You need to include a manifest file if you've set flags, such as JavaBean or Design-Time-Only.
4. To specify a manifest for Visual Cafe to start with, select Supply Initial Manifest.

   Enter the file name and path, or use the Browse button (…) to navigate to the manifest file (`*.mf`).
5. To make Visual Cafe save the manifest file it generates, select Save Generated Manifest.

   Enter the file name and path, or use the Browse button (…) to navigate to the path where you want the manifest file saved.

Note: If you're using the command-line archiving tool, you must have a manifest file.

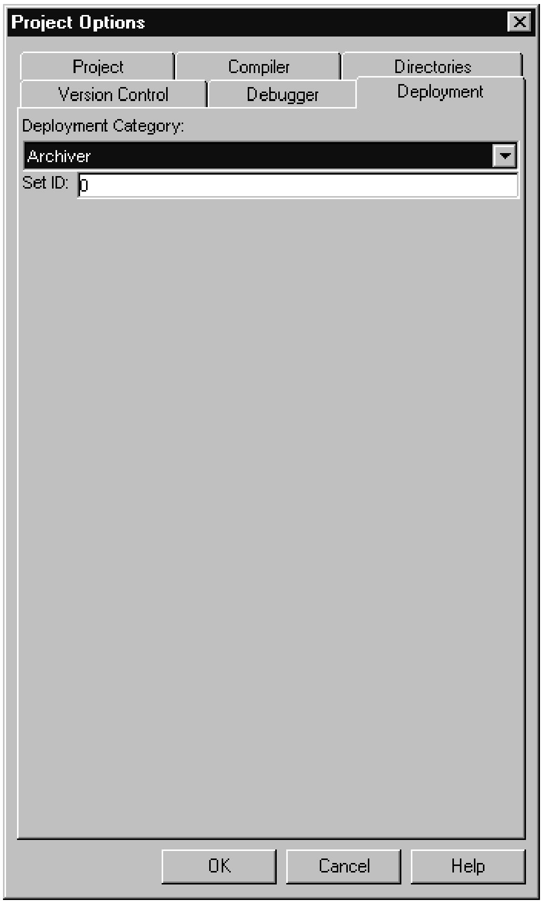
Tip: You can save a manifest the first time you create the JAR, then modify the manifest file by hand, and then include it as your initial manifest the next time you create the JAR.

6 Click OK.

The changes take effect immediately.

Setting archiver options for CAB files

If you've previously set the archive type to CAB in the General category of the Project Options dialog box, when you select the deployment category of Archiver, the Deployment Category page displays:



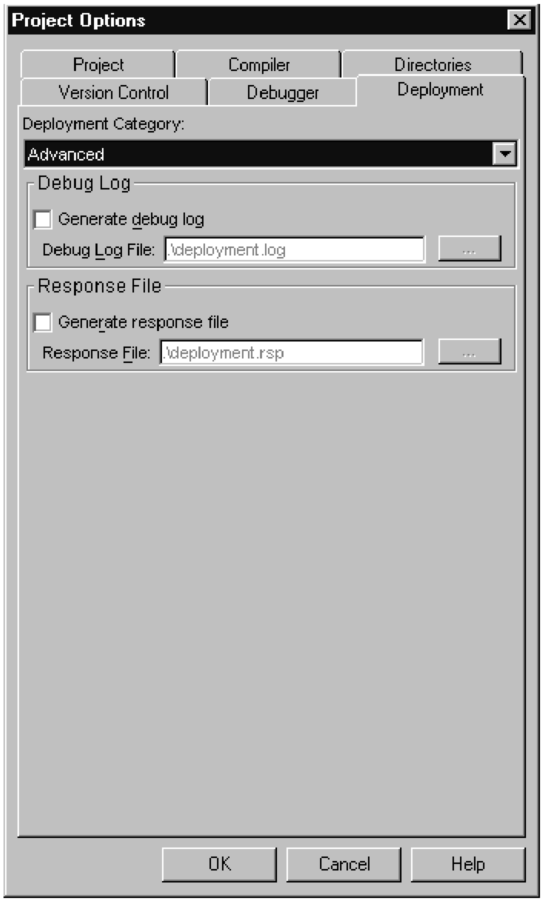
Deployment Category: Archiver

Set ID: 0

Here you can specify the ID of the CAB file. For more information, see Microsoft's documentation about CAB files.

Setting advanced deployment options for a project

When you select the deployment category of Advanced in the Project Options dialog box, the Deployment Category page displays:

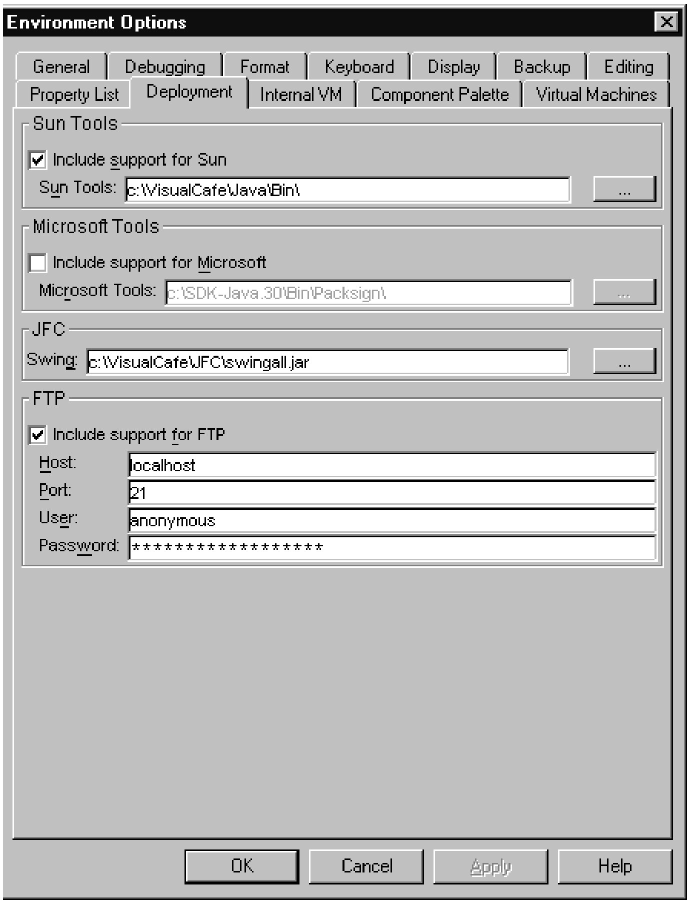


Here you can turn on the internal debug logging to help yourself debug any problems that you are having. Then, if you can't figure out what is wrong, Symantec's technical support team can use this information to further assist you. If you don't specify a name for the log file, the default setting is *outputdirectory*`\deployment.log`.

You can also choose to generate a response file. If you're using the command-line archiving tool, this file will provide you with the arguments you need to run the tool.

Setting deployment options for all projects

To set deployment options for all projects, click the Deployment tab in the Environment Options dialog box:



Sun Tools

☑ Include support for Sun

Sun Tools: c:\VisualCafe\Java\Bin\

Microsoft Tools

☐ Include support for Microsoft

Microsoft Tools: c:\SDK-Java.30\Bin\Packsign\

JFC

Swing: c:\VisualCafe\JFC\swingall.jar

FTP

☑ Include support for FTP

Host: localhost

Port: 21

User: anonymous

Password: ******************

In the Deployment page, you can specify paths to tools that you're using:

- If you're using JARs, you need to specify the path to `Jar.exe`; this program is provided with Visual Cafe.

- If you're using CABs, you need to specify the path to `Cabarc.exe`.
- If you're using Swing components, you need to specify the path to `Swingall.jar`; this JAR is provided with Visual Cafe.
- If you're deploying to an FTP site, you can specify the Host, Port, User, and Password. This is the default setting for all projects.

Here are some URLs you might want to use to obtain the tools you'll need. (Remember that URLs are subject to change.)

- Sun security and signed JAR tools:
  `http://web3.javasoft.com/products/jdk/1.1/docs/guide/security/index.html`
- Microsoft SDK for Java 3.1:
  `http://www.microsoft.com/java/download.htm`

Signing lets users alter the permissions that a Java program can have on their computer. Visual Cafe includes the Sun tools in the `\Java\Bin` folder.

Note: In order for Netscape Navigator/Communicator or Microsoft Internet Explorer to be able process signed JAR files, you must install the Java Plug-in for the respective browser. Visual Cafe does not currently support signing specifically for Netscape Navigator or Microsoft Internet Explorer.

After you download and install the tools, you need to specify the Paths in the Environment Options. You also need to set up any program-specific information, such as the key and signer. Basically, the tool needs to work from the command line before it will work in the Visual Cafe environment.

For information on setting deployment options for individual projects, see "Deploying your project" on page 5-31.

To specify paths in Environment Options:

1. Choose Environment Options from the Tools menu.

   The Environment Options dialog box appears.
2. Click the Deployment tab.

3 Select Include Support for Sun or Include Support for FTP, then specify the paths that you need by using one or more of the following options:

Field	**Tool**	**Description**
Sun Tools	`Jar.exe` and `Javakey.exe`	`Jar.exe` creates JAR files and `Javakey.exe` lets you sign JAR files. Signing allows downloaded applets in JAR files to run with the same full rights as local applications.
Microsoft Tools	`Cabarc.exe`	`Cabarc.exe` creates CAB files.
Swing	`Swingall.jar`	Contains Swing components.

4 Click Apply or OK to save the changes.

The changes take effect immediately.

About JAR files

JDK 1.1 supports two types of archives. The `.zip` file, which was also supported in Java 1.0, is a basic archive format that simply appends `.class` files together into one long file. The `.jar` format allows compaction, encryption, and signing. It also extends the archive format by allowing multiple file types to be included.

A **Java Archive (JAR)** file is an optionally compressed archive file that complies with the JavaBeans standard. It is the primary method for delivering JavaBeans components.

A JAR file contains one or more related Beans and any support files, including classes, icons, graphics, and sounds. A JAR tool, called `jar.exe` on Windows computers, archives and extracts JAR files and is provided with JDK 1.1.

In Visual Cafe, to use the JavaBeans components in a JAR file, you must first add the file to the Component Library. Then you can add the componetns to your projects. If HTML documentation was included in the JAR file for a Bean and you want to look at it, you need to expand the JAR by using `Jar.exe`.

If you're already shipping one set of resource files in another JAR, you can exclude those duplicate files from the current JAR. You can also specify whether a file within a JAR is a Bean or is used only at design time.

You can also deploy applets and applications from a JAR file.

About deployment and JAR files

Visual Cafe provides deployment features you can use within its environment to quickly create JAR files. To control how you will deploy your programs, you can set options for your project or for all projects in the Project Options and Environment Options dialog boxes.

You can also create JAR files with the AutoJAR command, as described in "Automatically updating Beans in the Component Library" on page 10-19. AutoJAR creates the JAR faster but does not give you as much control over the JAR output.

Using JAR files

In Visual Cafe, to use the JavaBeans components in a JAR file you must first add the file to your project's Component Library. Then you can add the components to your projects. If HTML documentation was included in the JAR for a Bean and you want to look at it, you need to expand the JAR by using `jar.exe`.

Visual Cafe's new Open by Name feature enables you to search for any `.jar` or `.class` file. You can type partial names, and in the case of multiple matches you can select the one you want to open.

Visual Cafe's drag-and-drop feature has been extended to dragging and dropping JAR files across projects. You can add JAR files to and remove JAR files from a project by dragging files and folders to and from Windows Explorer and other open projects.

Adding external files to a JAR

In the JAR Packager, the Add File command allows you to add files to the JAR that are not in your current project. These could be GIF or HTML files, as well as additional class files that may be dynamically loaded.

Expanding a JAR file

Visual Cafe provides a tool that you can use within its environment to quickly expand JAR files. To expand JAR files, use the `jar.exe` utility in the `java\bin` subdirectory. For example, enter the following at a DOS prompt:

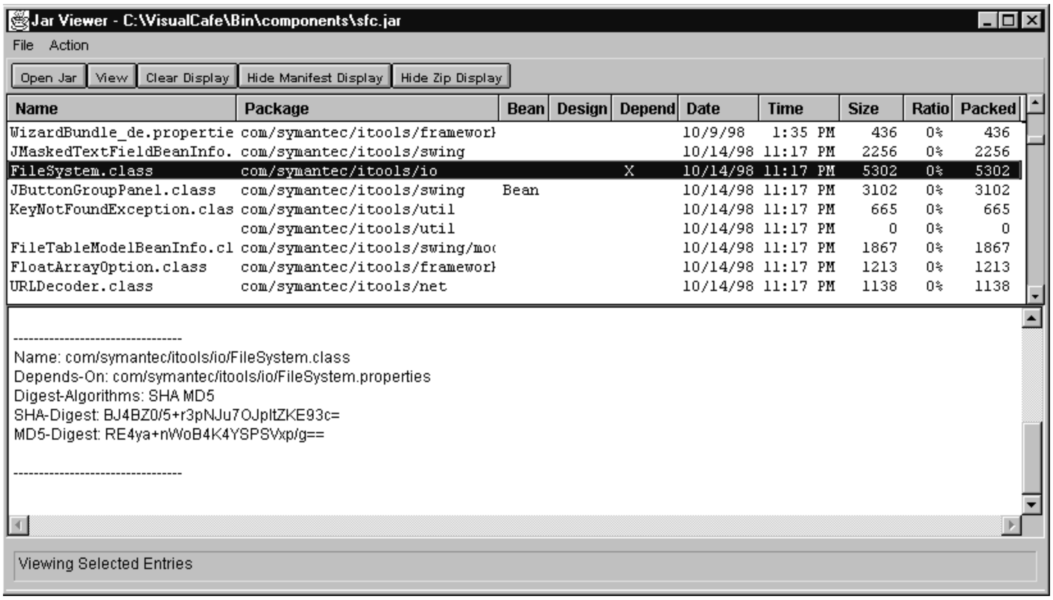
```
jar -xf filename.jar
```

Viewing a JAR file

Visual Cafe provides a JAR Viewer tool that you can use within its environment to view JAR files.

The **JAR Viewer** enables you to easily view the contents of a JAR file, just as you would a ZIP file. You can sort by date, type, and file name. Use the JAR Viewer to open an existing JAR or ZIP file, or to select a file in a JAR or ZIP file. You can also use the JAR Viewer to view the JAR's manifest file.

Here's an example of what the JAR Viewer looks like:



Name	Package	Bean	Design	Depend	Date	Time	Size	Ratio	Packed
WizardBundle_de.propertie	com/symantec/itools/framewor				10/9/98	1:35 PM	436	0%	436
JMaskedTextFieldBeanInfo.	com/symantec/itools/swing				10/14/98	11:17 PM	2256	0%	2256
FileSystem.class	com/symantec/itools/io			X	10/14/98	11:17 PM	5302	0%	5302
JButtonGroupPanel.class	com/symantec/itools/swing	Bean			10/14/98	11:17 PM	3102	0%	3102
KeyNotFoundException.clas	com/symantec/itools/util				10/14/98	11:17 PM	665	0%	665
	com/symantec/itools/util				10/14/98	11:17 PM	0	0%	0
FileTableModelBeanInfo.cl	com/symantec/itools/swing/mo				10/14/98	11:17 PM	1867	0%	1867
FloatArrayOption.class	com/symantec/itools/framewor				10/14/98	11:17 PM	1213	0%	1213
URLDecoder.class	com/symantec/itools/net				10/14/98	11:17 PM	1138	0%	1138

```
---------------------------------
Name: com/symantec/itools/io/FileSystem.class
Depends-On: com/symantec/itools/io/FileSystem.properties
Digest-Algorithms: SHA MD5
SHA-Digest: BJ4BZ0/5+r3pNJu7OJpItZKE93c=
MD5-Digest: RE4ya+nWoB4K4YSPSVxp/g==

---------------------------------
```

Viewing Selected Entries

To view the contents of a JAR file:

1. Choose JAR Viewer from the Tools menu.

   The JAR Viewer displays.
2. Click Open JAR, and browse to the JAR file you wish to view.

   The JAR Viewer opens the JAR file.

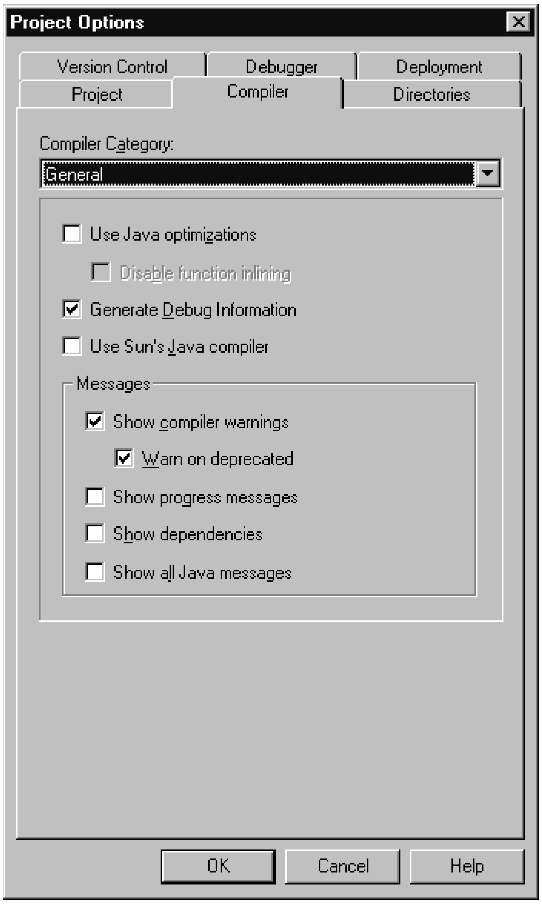
Within the JAR Viewer you can do the following:

- Sort the JAR contents by clicking the title of the column. Click the column title again to reverse the sort order.
- View the manifest of a class file in the JAR by double-clicking the selected file or clicking View.
- View the manifest of multiple files by selecting the files and then clicking View.
- Hide the manifest details by clicking Hide Manifest Display. Click Show Manifest Display to reveal the manifest details.
- Hide the Zip/Jar details by clicking Hide Zip Display. Click Show Zip Display to reveal the jar/zip details.

Setting compiler options

From the Compiler tab of the Project Options dialog box, you can control what compiler information is sent to the Messages window, which Java compiler to use, whether Java optimizations are performed, and more.

This view changes options for the currently selected option set, either the debug or final release type. For example, if you have the debug option selected and change an option in the compiler page, Visual Cafe changes that option for the debug option set, not the final option set. For more information, see "Specifying whether builds are debug or final" on page 5-16.



Project Options

Version Control | Debugger | Deployment | Project | Compiler | Directories

Compiler Category: General

- [ ] Use Java optimizations
  - [ ] Disable function inlining
- [✓] Generate Debug Information
- [ ] Use Sun's Java compiler

Messages
- [✓] Show compiler warnings
  - [✓] Warn on deprecated
- [ ] Show progress messages
- [ ] Show dependencies
- [ ] Show all Java messages

OK | Cancel | Help

Use the Compiler tab to set general compiler options, such as:

- Specifying Java optimizations
- Generating debug information
- Specifying the Sun Java compiler
- Showing compiler warning messages
- Showing progress messages
- Showing dependencies

- Showing all Java messages

For information about setting options that apply to Javadoc, see “Setting Javadoc options” on page 4-70.

You can also use the Compiler tab to set advanced compiler options, such as:

- Specifying Make settings
- Specifying custom compiler flags

These options are discussed in the following sections.

For information about compiler options for native programs, see “Setting project options for native programs” on page 11-7.

Specifying Java optimizations

You can optimize the Java executable to produce a more compact executable that runs faster.

You can also disable the function inlining, if needed. **Function inlining** means that Visual Cafe takes a function’s code and imbeds it in the calling function instead of calling the function. Inlining increases execution speed but also increases executable size. By default, the Disable Function Inlining option is not selected.

To specify Java optimizations:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Compiler tab.
4. In the Compiler Category drop-down list, choose General.
5. Select Use Java optimizations.
6. (Optional) Select Disable function inlining, if needed.
7. Click OK.

   The changes take effect the next time you compile your project.

Generating debug information

You can generate the debugging information used by the Visual Cafe debugger. For example, this option lets you see local variables during debugging. By default, the Generate Debug Information option is selected for debug release types and not selected for final release types.

To generate debug information:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Compiler tab.
4. In the Compiler Category drop-down list, choose General.
5. Select Generate Debug Information to have Visual Cafe create debugging information, or deselect it to turn this option off.
6. Click OK.

   The changes take effect the next time you compile your project.

Specifying the Sun Java compiler

You can instruct Visual Cafe use the Symantec Java compiler or the Sun Java compiler, `javac.exe`. When this option is deselected, the Symantec Java compiler, which is faster, is used. By default, the Use Sun's Java Compiler option is not selected.

Note: You cannot compile native Win32 applications and DLLs with the Sun Java compiler.

If you choose to use the Sun Java compiler, you can also o have Java diagnostic information displayed as well. See “Showing all Java messages” on page 5-62.

To select the Sun Java compiler:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose General.

5 Select Use Sun's Java compiler to have Visual Cafe use the Sun Java compiler, or deselect it so that the Symantec Java compiler is used.

6 Click OK.

The changes take effect the next time you compile your project.

Showing compiler warning messages

You can have Visual Cafe display compiler warnings, which appear in the Messages window. During development, you may want to look at items identified by warnings. By default, the Show Compiler Warnings option is selected.

To show compiler warning messages:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose General.

5 Select Show compiler warnings to have Visual Cafe display compiler warnings in the Messages window, or deselect it so that these messages are not shown.

6 Click OK.

The changes take effect the next time you compile your project.

Showing progress messages

You can have Visual Cafe display compiler progress messages, which appear in the Messages window. By default, the Show Progress Messages option is not selected.

To show progress messages:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose General.

5 Select Show progress messages to have Visual Cafe display progress messages in the Messages window, or deselect it so that these messages are not shown.

6 Click OK.

The changes take effect the next time you compile your project.

Showing dependencies

You can have Visual Cafe display file dependencies, such as imports. File dependency information appears in the Messages window. By default, the Show Dependencies option is not selected

To show dependencies:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose General.

5 Select Show dependencies to have Visual Cafe display file dependency information in the Messages window, or deselect it so that these messages are not shown.

6 Click OK.

The changes take effect the next time you compile your project.

Showing all Java messages

When using the Sun Java compiler, you can have the compiler report diagnostic messages about its own execution. (This option is ignored by the Symantec compiler.) By default, the Show All Java Messages option is deselected.

To show all Java messages when using the Sun Java compiler:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose General.

5 Select Show all Java messages to have Visual Cafe display Java diagnostic information in the Messages window, or deselect it so that these messages are not shown.

6 Click OK.

The changes take effect the next time that you compile your project.

Specifying Make settings

You can specify how imports are handled when you compile your project. You can instruct Visual Cafe to do the following:

- ◆ Not check dependencies on imports
- ◆ Warn you when imports are out of date
- ◆ Generate classes for out-of-date imports

If you select the Don't check dependencies on imports option, Visual Cafe does not check imports to determine if they are out-of-date. By default, this option is selected.

If you select Warn on out-of-date imports, Visual Cafe provides a warning message if an imported class is out of date. The class is not regenerated.

If you select Generate classes for out-of-date imports, Visual Cafe searches for the Java source code of imports and updates the classes to the most current version, if necessary.

To specify Make settings:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose Advanced.

5 Select one of the following options:

- ❖ Don't check dependencies on imports
- ❖ Warn on out-of-date imports

- ❖ Generate classes for out-of-date imports

6 Click OK.

The changes take effect the next time you compile your project.

Specifying custom compiler flags

You can choose to pass command-line options to the compiler. For information on SJ compiler options, see “Compiling from the SJ command line” on page 5-7.

To specify custom compiler flags:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose Advanced.

5 In the Custom compiler flags field, enter the options you want.

6 Click OK.

The changes take effect the next time you compile your project.

C H A P T E R 6

Debugging Your Program

This chapter shows you how to use Visual Cafe's integrated debugger to find and correct problems in your programs. You'll learn how to do the following:

- Use the debug workspace
- Use the Breakpoints, Call Stack, Messages, Threads, and Variables windows
- Debug code in the Source window
- Work with breakpoints
- Modify variables, expressions, and methods
- Debug threads
- Handle exceptions
- Use incremental debugging (Visual Cafe Professional and Database Editions)
- Debug your programs in a Web browser (Visual Cafe Professional and Database Editions)

For information about debugging native programs, see Chapter 11, "Creating Native Win32 Java Applications."

About the Visual Cafe debugger

After you write and execute a program, you may find that it doesn't work the way you expected it to. The fact that a program compiles without errors doesn't mean it will work correctly. When you run your program, you may encounter a variety of errors, including compile errors (code construction or syntax errors, for example), run-time errors that occur after you start the program (dividing by zero or writing to a file that doesn't exist, for example), and logic errors (the program doesn't do what you want it to do).

Visual Cafe's debugger lets you watch your program execute line by line. Using the debugger, you can monitor:

- the values stored in variables
- which methods are being called
- the order in which program events occur

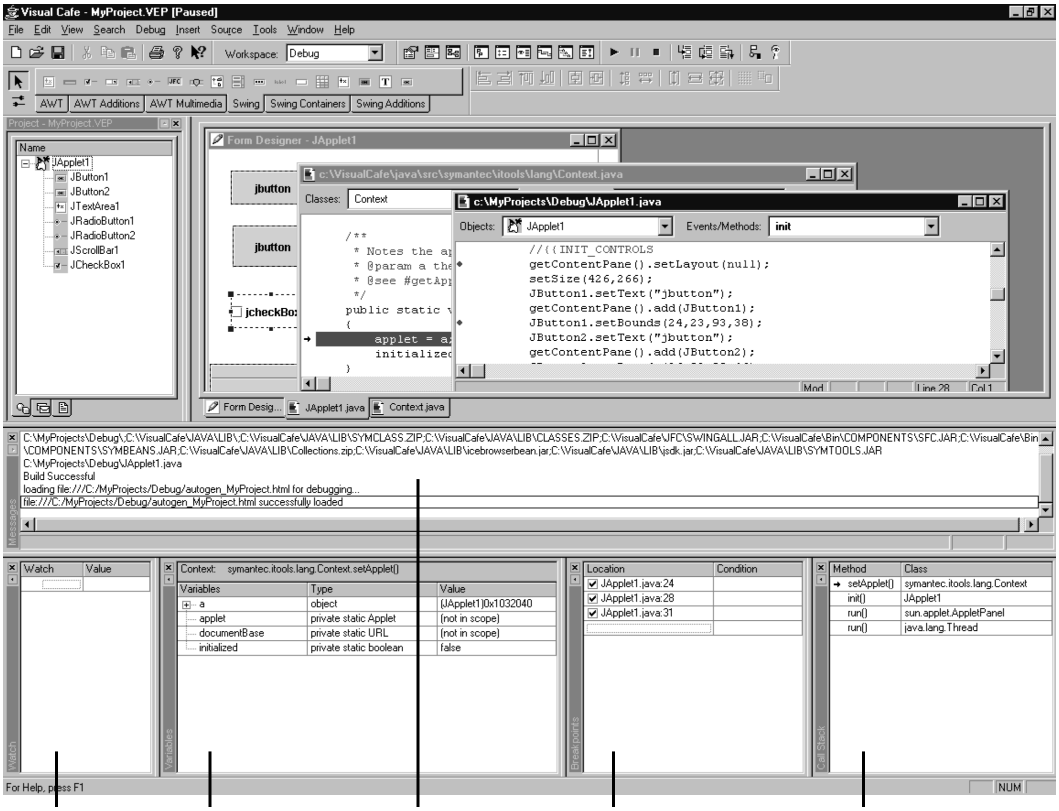
The Visual Cafe debugger:

- provides a Windows graphical user interface
- provides a graphical representation of data structures
- controls program execution
- supports breakpoints
- supports Java exceptions
- evaluates expressions
- lets you drag and drop to execute commands
- supports incremental debugging (Visual Cafe Professional and Database Editions), allowing you to make changes while a program is paused in the debugger and continue running the program with the new changes
- provides a Messages window, which is separate from the debugger, to display the status of your debugging session
- supports debugging in a Web browser (Visual Cafe Professional and Database Editions) such as Netscape Navigator or Internet Explorer

About the debug workspace

After you build your project you can enter **debug mode.** This mode offers several windows and menus that let you debug your program. While you're debugging, you can use the Source window (or the Class Browser's Source pane) and the Breakpoints, Variables, Call Stack, Threads, and Messages windows to help you isolate and resolve problems. The Source window, Source pane, Messages window, and Breakpoints window are available at all times; the other windows show information while you're debugging.

The debug workspace is automatically loaded and enabled by default. The Call Stack, Variables, Messages, and Breakpoints windows are opened automatically in the same size and position they had when you last ran the project.



```
    /**
     * Notes the a
     * @param a the
     * @see #getApp
     */
    public static v
    {
        applet = a;
        initialized
    }
```

```
        //{{INIT_CONTROLS
        getContentPane().setLayout(null);
        setSize(426,266);
        JButton1.setText("jbutton");
        getContentPane().add(JButton1);
        JButton1.setBounds(24,23,93,38);
        JButton2.setText("jbutton");
        getContentPane().add(JButton2);
```

```
C:\MyProjects\Debug\;C:\VisualCafe\JAVA\LIB\;C:\VisualCafe\JAVA\LIB\SYMCLASS.ZIP;C:\VisualCafe\JAVA\LIB\CLASSES.ZIP;C:\VisualCafe\JFC\SWINGALL.JAR;C:\VisualCafe\Bin\COMPONENTS\SFC.JAR;C:\VisualCafe\Bin
\COMPONENTS\SYMBEANS.JAR;C:\VisualCafe\JAVA\LIB\Collections.zip;C:\VisualCafe\JAVA\LIB\icebrowserbean.jar;C:\VisualCafe\JAVA\LIB\jsdk.jar;C:\VisualCafe\JAVA\LIB\SYMTOOLS.JAR
C:\MyProjects\Debug\JApplet1.java
Build Successful
loading file:///C:/MyProjects/Debug/autogen_MyProject.html for debugging...
file:///C:/MyProjects/Debug/autogen_MyProject.html successfully loaded
```

Watch	Value

Context: symantec.itools.lang.Context.setApplet()

Variables	Type	Value
⊞ a	object	(JApplet1)0x1032040
applet	private static Applet	(not in scope)
documentBase	private static URL	(not in scope)
initialized	private static boolean	false

Location	Condition
✓ JApplet1.java:24	
✓ JApplet1.java:28	
✓ JApplet1.java:31	

Method	Class
→ setApplet()	symantec.itools.lang.Context
init()	JApplet1
run()	sun.applet.AppletPanel
run()	java.lang.Thread

Watch window *Variables window* *Messages window* *Breakpoints window* *Call Stack window*

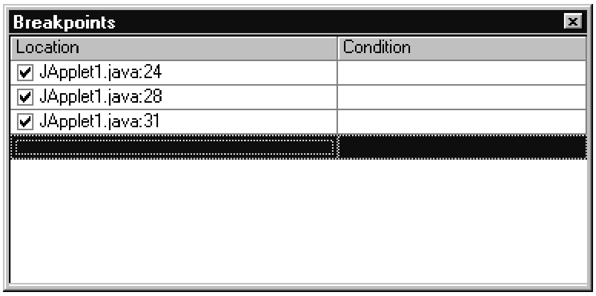
By default, Visual Cafe runs your Java program in the AppletViewer. By running your program in the AppletViewer, Visual Cafe eliminates the need for you to compile the `.class` file, open a Java-compatible Web browser, write the corresponding HTML code and load it into the Web browser, and reload the applet over and over again.

You can also debug in a Web browser if you wish, by selecting it in the Project Options dialog box. See "Debugging applets in a Web browser" on page 6-42 for more information.

The debugging windows and toolbar are introduced in this section. For information on working in these tools, see "Using the debugger" on page 6-10.

About the Breakpoints window

The Breakpoints window displays a list of breakpoints that are currently set in the source code. *Breakpoints* are flags you can insert into the code at specific points that cause program execution to pause (see "Working with breakpoints" on page 6-16 for more information). When the debugger goes through the instructions, it stops whenever it encounters a breakpoint. At that point you can check on the value of the data and other program conditions.



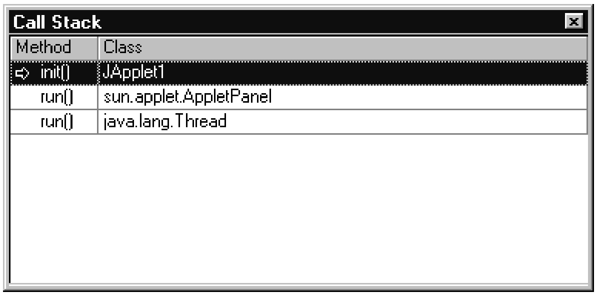
Location	Condition
✓ JApplet1.java:24	
✓ JApplet1.java:28	
✓ JApplet1.java:31	

The **Breakpoints window** allows you to view a list of all breakpoints in the currently active project. It lists the source file and line number of each breakpoint. You can use the checkbox that precedes each breakpoint to enable or disable it. Double-clicking on a breakpoint in the Breakpoints window goes to the source file and displays the selected breakpoint.

Use this window to add, remove, or modify breakpoints. For more information, see "Working with breakpoints" on page 6-16.

About the Call Stack window

The **Call Stack window** lists the method calls that have been made since the program began running. A **call** is a reference made from one class to methods in another class. This list of calls is known as the **call stack.** Each entry lists the name of the method, followed by the name of the class that contains the method. The Call Stack window shows all the method calls that have started, but have not completed execution.



Method	Class
⇨ init()	JApplet1
run()	sun.applet.AppletPanel
run()	java.lang.Thread

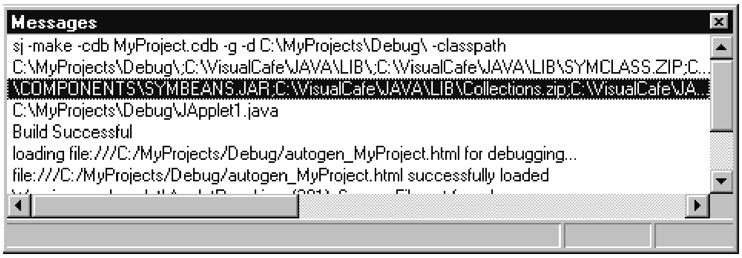
The **call chain** is the sequence of functions that were called to get to the current function. You can access the functions in the call chain through the Call Stack window.

See “Using the Call Stack window” on page 6-31 for more information.

About the Messages window

The **Messages window** displays error messages. When you’re debugging, it displays messages during compile time. For example, if an error is encountered during parsing, the error message displays in this window.

Double-click any error message to open the file in the Source window with focus on the selected error.



```
sj -make -cdb MyProject.cdb -g -d C:\MyProjects\Debug\ -classpath
C:\MyProjects\Debug\;C:\VisualCafe\JAVA\LIB\;C:\VisualCafe\JAVA\LIB\SYMCLASS.ZIP;C...
\COMPONENTS\SYMBEANS.JAR;C:\VisualCafe\JAVA\LIB\Collections.zip;C:\VisualCafe\JA...
C:\MyProjects\Debug\JApplet1.java
Build Successful
loading file:///C:/MyProjects/Debug/autogen_MyProject.html for debugging...
file:///C:/MyProjects/Debug/autogen_MyProject.html successfully loaded
```

Anything you write to `System.out` can also appear in the Messages window when you're running the program in the debugger. For example:

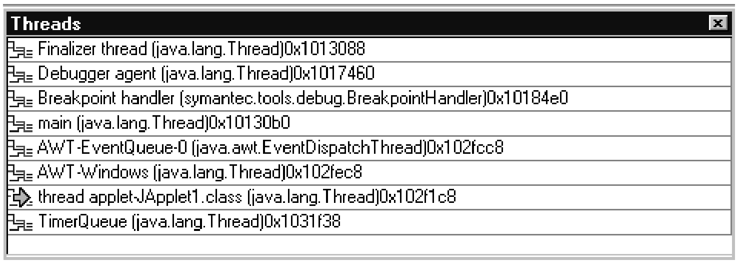
```
System.out.println("my message");
```

You can customize what kind of messages the Message window displays. To specify what's displayed in the Messages window, choose Options from the Project menu. For example, one of the options in the Options submenu lets you turn off Java compiler warnings.

See "Working with debugger messages" on page 6-15 for more information.

About the Threads window

The **Threads window** presents at a glance all the threads that your program has created, together with their states. A **thread** is a process in a program that has a beginning and an end. In applications, the `main` method is responsible for indicating the beginning and end of the program. In applets, the Web browser uses various methods to control the program flow.



```
Threads
Finalizer thread (java.lang.Thread)0x1013088
Debugger agent (java.lang.Thread)0x1017460
Breakpoint handler (symantec.tools.debug.BreakpointHandler)0x10184e0
main (java.lang.Thread)0x10130b0
AWT-EventQueue-0 (java.awt.EventDispatchThread)0x102fcc8
AWT-Windows (java.lang.Thread)0x102fec8
thread applet-JApplet1.class (java.lang.Thread)0x102f1c8
TimerQueue (java.lang.Thread)0x1031f38
```

Programs are not limited to performing a single process; Java programs can use threads to perform multiple processes simultaneously. This is called **multithreading**. For example, suppose you're using a text editor to type a letter and you want to save your changes before you continue. If the text editor program is single-threaded, when you save the file the rest of the program must wait until the file is completely written to the hard disk.

In a multithreaded application, the process that saves the file can be an independent thread with its own beginning and end. When you save the file, the file-saving thread starts and runs in parallel with the application's other processes. You can continue to type your letter as a copy of the file is written to disk in the background. Multithreaded programs run faster and are more convenient than single-threaded applications.

Note: The Threads window provides no benefits when you're debugging a single-threaded application. It's useful only when you're debugging a multithreaded application.

You use the Threads window to:

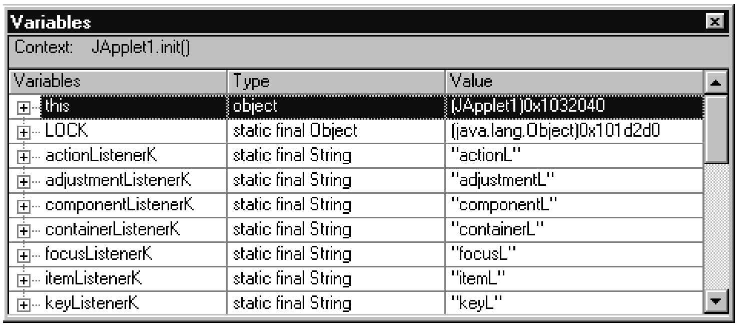
- easily switch between threads to debug
- view the source for a selected thread
- update the Source window with a thread's current location
- update the Call Stack window with a thread's call chain

See "Using the Threads window" on page 6-33 for more information.

About the Variables window

A **variable** is a structure in memory that holds a value that has been assigned to it. A variable that's defined in a class defines the class's structure.

Your program's variables are displayed in the **Variables window,** which displays local variables, global variables, and objects that are local to the current method. You can examine objects and array data elements as well as simple data types. For each variable, the Variables window displays the variable name, type, and value.



Variables

Context: JApplet1.init()

Variables	Type	Value
this	object	(JApplet1)0x1032040
LOCK	static final Object	(java.lang.Object)0x101d2d0
actionListenerK	static final String	"actionL"
adjustmentListenerK	static final String	"adjustmentL"
componentListenerK	static final String	"componentL"
containerListenerK	static final String	"containerL"
focusListenerK	static final String	"focusL"
itemListenerK	static final String	"itemL"
keyListenerK	static final String	"keyL"

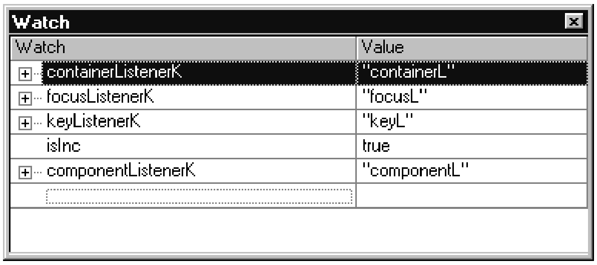
When you pause the execution of your program, you can modify the value of a variable and then continue execution with the new value in place.

You can modify the value of a variable from either the Variables window or the Watch window (the Watch window is described next).

For more information, see “Using the Variables window” on page 6-26.

About the Watch window

The **Watch window** lets you specify variables and expressions that you want to watch continuously while debugging your program. In this window, you can examine the contents of a class variable.



Watch	Value
⊞ containerListenerK	"containerL"
⊞ focusListenerK	"focusL"
⊞ keyListenerK	"keyL"
isInc	true
⊞ componentListenerK	"componentL"

The Watch window lets you:

- ◆ specify variables and expressions to watch
- ◆ modify variables or expressions
- ◆ delete variables or expressions

See “Using the Watch window” on page 6-28 for more information.

You can also modify the value of a variable using the Variables window (see “About the Variables window” on page 6-7 and “Using the Variables window” on page 6-26).

About the Source window

The **Source window** is the primary debugging window; it’s where you see your code at its current point of execution. When you debug your program, you can open the Source window containing the current line of code, if possible. (Sometimes Visual Cafe cannot open the Source window because you may not have all the source, for example.)

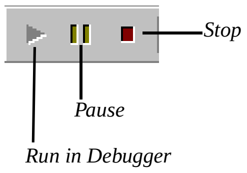
When you are in debug mode, the Source window provies extra functionality, allowing you to manipulate breakpoints and specify variables to watch in the Watch window.

You can also examine variables and evaluate expressions that you select in the Source window.

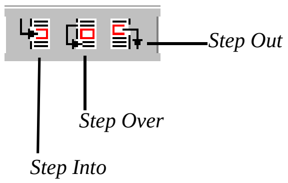
See Chapter 4, "Working with Source Code" for details about working with source code.

About the Debug toolbar

You can use the debug environment's Debug menu to select a variety of debugging commands. Alternatively, you can use the Debug toolbar, which provides easy access, in the form of icons, to the commands that you use most often. From the Debug toolbar, you can do the following:

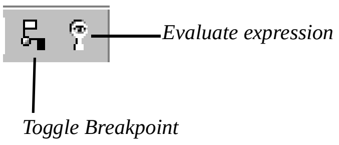


- Start a program
- Pause a program
- Stop a program



- Step into a method
- Step over a method

- Step out of a method



- Toggle a breakpoint
- Evaluate an expression

For more information about using these commands, see the appropriate sections in this chapter.

Keyboard shortcuts

In addition to the Debug menu and the Debug toolbar, Visual Cafe provides keyboard shortcuts for many debugging commands. These shortcuts are shown in the following table:

Debug Menu Command	Keystroke equivalent
Continue	F5
Step Into	F8
Step Out	SHIFT-F11
Step Over	F10
Restart	SHIFT-F5

Using the debugger

When you run your program in the debugger, you hand control over to it and interact with it as a user, but you can still debug it. When you run your program outside of the debugger, you are executing your program as a finished program and you can’t break into debug mode.

In Visual Cafe, you can choose to run your program in a number of ways. You can run your program:

- so it pauses as it’s about to execute the first line

- ◆ until it terminates normally or reaches a breakpoint or an exception (breakpoints and exceptions are discussed later in this chapter)

You can also run your program from its current debug point until it terminates, ignoring any breakpoints that are set. This allows you to check for any suspected exception conditions. Alternatively, you can run your program until execution reaches the cursor location in the Source window. This is a handy way to continue from a breakpoint to a line you're inspecting in the Source window. For more information, see "Ignoring all breakpoints" on page 6-22.

Starting a debugging session

You launch the debugger within Visual Cafe. Your project must be open in the Project window so the debugger can have access to symbolic debugger information.

Note: Your program must successfully compile into a `.class` file before the debugger can open it.

After creating your program you'll want to run it until it terminates normally or encounters a breakpoint.

To start a debugging session:

1. Open the project you want to debug.
2. Choose Run in Debugger from the Project menu (or press F5).

   Visual Cafe switches to debug mode, thereby running your program in the debugger.

Note: Before running in the debugger, make sure your project options are set to debug. See "Specifying whether builds are debug or final" on page 5-16 for more information.

You can run your program outside of the debugger by choosing Execute from the Project menu. For more information, see "About files in a project" on page 3-42.

If you've set breakpoints or exceptions in your program (see "Working with breakpoints" on page 6-16 and "Setting exceptions" on page 6-38 for details), your program will run until it encounters one, at which point it will pause execution.

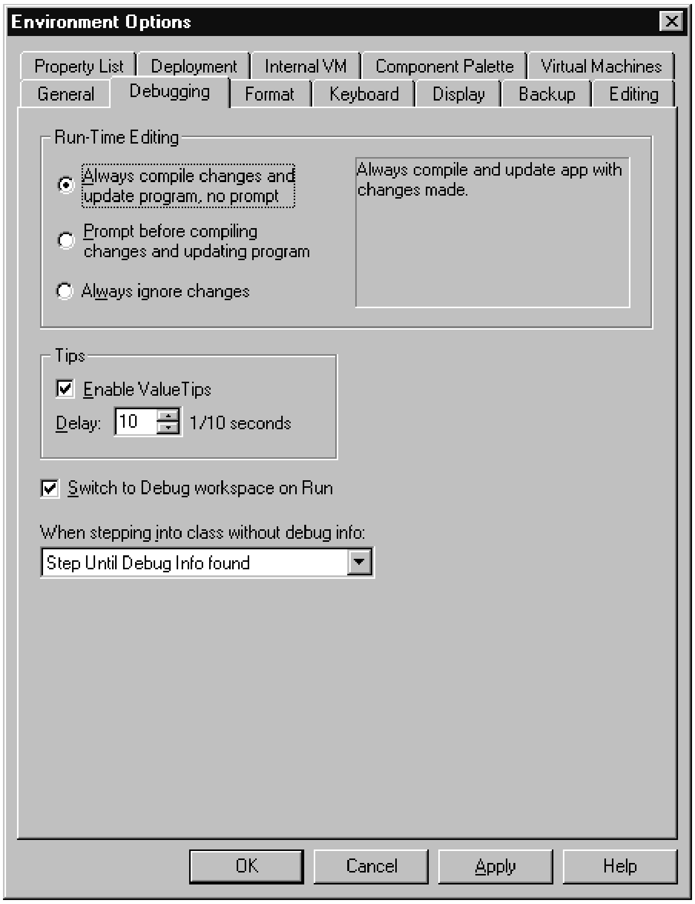
To run to the first line in your program, you can step into a method. See “Stepping into a method” on page 6-24 for more information.

Switching to the Debug workspace when running in the debugger

You can control whether Visual Cafe switches to the Debug workspace when running in the debugger. By default, this option is selected.

To switch to the Debug workspace when running in the debugger:

1 From the Tools menu, choose Environment Options, then click the Debugging tab.



Run-Time Editing

- (●) Always compile changes and update program, no prompt
- ( ) Prompt before compiling changes and updating program
- ( ) Always ignore changes

Always compile and update app with changes made.

Tips

- [✓] Enable ValueTips
- Delay: 10 1/10 seconds

[✓] Switch to Debug workspace on Run

When stepping into class without debug info: Step Until Debug Info found

2 If you want Visual Cafe to go to the Debug workspace when you run in the debugger, select Switch to Debug workspace on Run. If you want to remain in the current workspace when running in the debugger, deselect this option.

Ending a debugging session

You can completely stop execution of a program during a debugging session (as opposed to pausing temporarily, as described in the next section). Stopping a program terminates it completely so that you can continue working on it. Once you've stopped a program, you can't continue executing it. You can, however, edit your program as you debug it by using Visual Cafe's incremental debugging features (also called run-time editing). For more information, see "Using incremental debugging" on page 6-40.

To end a debugging session:

◆ Do either of the following:

❖ Quit the application by closing the project or Visual Cafe.

or

❖ Choose Stop from the Debug menu.

The debugging session ends, and the program is terminated.

Restarting a debugging session

After you stop a program's execution, you can restart it to run it again from the beginning. (This is different from continuing a program, where you resume execution after a breakpoint; see the next section, "Pausing a program to debug it.")

To restart debugging a program from the beginning:

◆ Choose Restart from the Debug menu. (Or press SHIFT-F5.)

If you choose Restart while a program is executing, it's the same as choosing Stop from the Debug menu, then Run in Debugger from the Project menu.

If you choose Restart when a program is stopped, it's the same as choosing Stop from the Debug menu, then Step Into from the Project menu.

Pausing a program to debug it

In Visual Cafe, you can pause an executing program and switch to debug mode. The effect on your program is the sameas hitting a breakpoint.

To pause a program:

◆ Choose Pause from the Debug menu.

The program's execution pauses.

Note: When an unhandled exception is encountered, the program pauses automatically at the line where the error occurred.

When you pause your program, it may be executing in a portion of code inside the Java AWT. If so, you're prompted to find that source file.

While your program is paused, you can examine its state by using the Call Stack and Variables windows, as well as by setting breakpoints See "Using the Call Stack window" on page 6-31, "Using the Variables window" on page 6-26, and "Working with breakpoints" on page 6-16 for more information.

Resuming a program

You can resume execution when your program is paused because of a breakpoint or an exception, or because you manually paused execution. When you resume your program, execution continues from the current location.

To resume debugging a program:

◆ Choose Continue from the Debug menu. (Or press F5.)

  The debugger stops executing the code at the line that's marked with a breakpoint.

Working with debugger messages

The Messages window is automatically displayed when you enter debug mode. If there are syntax errors in your source code, Visual Cafe flags them in the Messages window after a compile. You can easily navigate to each error from this window.

To open the Messages window:

◆ Choose Messages from the View menu.

To navigate to an error from the Messages window:

1. Choose Messages from the View menu to bring the Messages window to the front.

2 Double-click on any error message to open the related file in the Source window.

The file containing the error opens in the Source window at the offending line. Once the file opens, you can work on your source code.

If you're using incremental debugging, you can change your code and compile-time errors are reported in the Messages window the same as always. For more information, see "Using incremental debugging" on page 6-40.

Note: The Messages window does not show any user-created threads until a breakpoint is hit.

Using Messages window shortcut keys

You can use press keys to quickly navigate in the Messages window. Here's a list of the shortcut keystrokes (CTRL - = means press the CTRL and = keys simultaneously, and so on):

Press this...	To go to this message in the Messages window...
CTRL - =	Current error message
CTRL - SHIFT - =	First error message
CTRL - –	Next error message
CTRL - SHIFT - –	Previous error message

Working with breakpoints

Watching an entire program execute from beginning to end helps you understand the program flow. However, in many cases you'll want to observe only specific parts of the program. It makes sense to step through just the few lines of code that have behavior you want to observe, rather than the entire program. In these cases you can use breakpoints.

A **breakpoint** is a flag (a piece of code) placed in the bytecode that tells the debugger to pause execution of the program. This tiny piece of code is invisible to you, but it is represented in the Source window as a diamond to the left of the line where the breakpoint occurs.

Breakpoints allow you to examine your program line by line as it executes in the debugger. The compiler recognizes the breakpoint and treats it as a part of your program. Breakpoints are important to the compiler only when you're testing and debugging your program. Breakpoints are ignored when you compile your program into a finished `.class` file.

When you work with breakpoints, the debugger exectues all the statements in the program until it encounters a breakpoint. The program pauses at the breakpoint so you can check the state of the program at that point, including checking the value of variables. As you debug your code, you can use as many breakpoints as you want.

You can set breakpoints anywhere in your code in order to stop execution at a particular line and regain control after starting your program. When your program breaks on a breakpoint, you can examine variable values, single-step your program, or examine the state of your program in any way you like.

In Visual Cafe, you can set a breakpoint on:

- the current line in the Source window
- a specific line number
- a method name
- the condition of a variable or expression

These options are discussed in the following sections.

Managing breakpoints

Visual Cafe provides many ways to manage breakpoints, allowing you to stop program execution at a specific location or on a predetermined condition, such as a variable being assigned a particular value. You can manipulate breakpoints in either the Source window or the Breakpoints window. You can view or modify the different parameters of any breakpoint and enable or disable the currently defined breakpoints.

To open the Breakpoints window:

- Choose Breakpoints from the View menu.

Breakpoints are saved with the project and are made visible each time you open the project. The state of each breakpoint — enabled or disabled — is also saved with the project.

Setting a breakpoint

You set a breakpoint directly in a line of source code in the Source window.

To set a breakpoint:

1. Click the line where you want to set your breakpoint.
2. From the Source menu, choose Set Breakpoint. Or you can right-click and choose Set Breakpoint. You can also click the Set Breakpoint button on the Debug toolbar.

   The breakpoint is set, indicated by a diamond symbol in the left margin next to the line of code.

   Tip: You can also set and remove breakpoints on the current line by pressing the F9 function key.

Setting a breakpoint on a line number

You can have your program break every time a specific line in your current source file is about to be executed. You set breakpoints in the Source window, or in the Source pane of the Class Browser.

To set a breakpoint on a line number:

1. In the Source window, click the line where you want to set a breakpoint.
2. From the Source menu, choose Set Conditional Breakpoint.
3. Select Line number.
4. Enter a line number in the text box.
5. Click OK.

   The breakpoint is added to the list in the Breakpoints window.

Setting a conditional breakpoint

A **conditional breakpoint** pauses the execution of your program when a specified condition is met. When you set a conditional breakpoint, it's added to the breakpoint list in the Breakpoints window, along with the condition you specify.

To set a conditional breakpoint:

1. In the Source window, click on the line where the breakpoint should occur.
2. Choose Set Conditional Breakpoint from the Source menu.
3. Select If Expression Is True.
4. Type your breakpoint condition into the text box.

   The condition that triggers a breakpoint must be an expression that evaluates to true. When the expression is true, the breakpoint occurs.

   Note: The expression is evaluated, so make sure it evaluates to a Boolean. For example, *n* = 64 would set *n* to 64, while *n* == 64 would be true when *n* was 64 during the execution of your program.

5. Click Add.

   The breakpoint is added to the list in the Breakpoints window.

Modifying a conditional breakpoint

You modify a conditional breakpoint by changing the conditions of the breakpoint in the Breakpoints window.

To modify a conditional breakpoint:

1. Choose Breakpoints from the View menu to display the Breakpoints window.
2. Click on the Condition field of the breakpoint you want to change.
3. Type the new condition into the Condition field.
4. Press ENTER to save the change.

Setting a breakpoint at a method

You can have your program break every time a specific method is executed by setting a breakpoint at a method name.

To set a breakpoint on a method name:

1. Choose Set Conditional Breakpoint from the Source menu.
2. Select Method Name in the dialog box that appears.
3. Enter a method name in the text box.
4. Click OK.

   The breakpoint is added to the list in the Breakpoints window.

Enabling or disabling a breakpoint

You can tell the debugger to ignore a particular breakpoint by disabling that breakpoint. If you want to use that breakpoint later, you can enable it.

Turning breakpoints on and off in the Breakpoints window

The checkbox preceding each breakpoint in the Breakpoints window indicates whether or not that breakpoint is enabled. When the box is checked, the breakpoint is enabled. When the box is empty, the breakpoint is disabled.

To disable a breakpoint:

◆ Find the breakpoint in the Breakpoint window's list and click the checked box corresponding to that breakpoint.

  When the box is empty, the breakpoint is disabled.

Note: Disabling a breakpoint does not delete it from the breakpoint list. You can enable it later if you wish.

To enable a breakpoint:

◆ Find the breakpoint in the Breakpoint window's list and click the empty checkbox corresponding to that breakpoint.

  When the box is checked, the breakpoint is enabled.

Turning breakpoints on and off in the Source window

Toggling a breakpoint is a quick way to reverse the state of a breakpoint in any line of source code in the Source window. When you toggle a breakpoint, you're turning it on or off.

In the Source window, an enabled breakpoint is marked with a solid red diamond; a disabled breakpoint is marked with a hollow red diamond.

Note: Turning a breakpoint off does not remove it from Visual Cafe; it merely tells the debugger to ignore that breakpoint.

To toggle a breakpoint in the Source window:

1. In the Source window, click on the line where you want to toggle the breakpoint.
2. In the Debug toolbar, click the Toggle Breakpoint toolbar button to turn off the breakpoint. Click it again to turn the breakpoint back on.

Tip: You can also use the F9 function key to toggle breakpoints.

Clearing a breakpoint

If you no longer need a breakpoint at a certain line, you can clear it. Clearing a breakpoint deletes it from the set of breakpoints you've defined. (Clearing a breakpoint is different from disabling a breakpoint, which retains it in your breakpoint list in a temporarily disabled state; see the previous section, "Enabling or disabling a breakpoint.")

You can clear a breakpoint at the line of code in the Source window (or the Source pane of the Class Browser) or from the list in the Breakpoints window.

To clear a breakpoint in the Source window or pane:

1. Select the breakpoint by clicking on the line where the breakpoint is set (the breakpoint is denoted with a diamond symbol in the left margin).
2. Choose Clear Breakpoint from the Source menu, or right-click and choose Clear Breakpoint.

To clear a breakpoint in the Breakpoints window:

◆ Do either of the following:
  - ❖ Select one or more breakpoints in the list and press DELETE.
  - ❖ Select one or more breakpoints, then choose Clear from the Breakpoints menu, or right-click and choose Clear.

To clear all breakpoints in the Breakpoints window:

◆ Choose Clear All from the Breakpoints menu, or right-click in the Breakpoints window and choose Clear All.

  When the breakpoint is cleared, the diamond symbol is removed from the left margin of the line of source code, and the breakpoint is removed from the list in the Breakpoints window.

Ignoring all breakpoints

There's an easy way to ignore all breakpoints you've set without changing the state of any of the breakpoints in the Breakpoints window. You can do this by:

◆ Running to the end of the program
  or

◆ Running to the cursor location

Running to the end of the program

You can run your program from its current point until the end, ignoring all breakpoints.

To run to the end of the program:

◆ Choose Continue to End from the Debug menu.

  Your program will ignore all breakpoints and continue to run until it terminates, or until you cause it to terminate by closing it as you would outside of the debugger.

Note: If any kind of exception occurs, the program will pause at the point of the violation, unless otherwise instructed. You can specify options for handling exceptions by clicking the Debugger tab of the Project Options window, choosing Exceptions from the Category menu, then restarting the debugging session. For more information, see “Setting exceptions” on page 6-38.

Running to the cursor location

You can run your program until it reaches the cursor location, ignoring any breakpoints along the way.

To run to the cursor location:

◆ Choose Continue to Cursor from the Debug menu.

Your program will ignore all breakpoints and run until it reaches the cursor location.

Note: If the selected line does not get executed before the end of the program, your program will not pause.

Viewing the source code for a breakpoint

In Visual Cafe, you can select a breakpoint that’s listed in the Breakpoints window and view the associated source code.

To view the source code associated with a breakpoint:

1. From the View menu, choose Breakpoints to open the Breakpoints window.
2. Click the breakpoint for which you want to see the source code.
3. From the Breakpoint menu, choose Go to Source, or right-click and choose Go to Source.

   The source code associated with that breakpoint is displayed in the Source window.

Stepping through code

When you debug a program, you can execute the instructions in the program line by line and watch the result. This allows you to pinpoint the exact source of a problem in your code..

When your program hits a breakpoint, you can step through your lines of code one line at a time using three techniques: Step Into, Step Over, and Step Out. Each technique has an associated item on the Debug toolbar.

Stepping into a method

When you step into your code, the debugger executes one line of code, including any jumps to other methods, until it reaches the next line of code. This method allows you to step through a program, executing every statement completely.

When you step into a method call, the debugger jumps to the code for that method, and steps through it as well. If the code for the method is in a different file, the debugger opens that file and jumps right to the code for the method.

Stepping into every line of code for every method called by your program isn't usually the best way to debug a program. For example, let's say that your program calls the `println` method. We know the `println` method works, because it's part of the Java language. Stepping into method calls like `println` throughout a program can quickly result in a chaos of open windows. A more useful approach to debugging is to step into some parts of the code and to step over other parts. You can avoid stepping through code that you know works by stepping over method calls. However, if you stepped in where you should have stepped over, you could choose to step out.

If the program is running and you are paused in the program, this command steps to the next source code statement. If the current line is a method call, Step Into steps inside the method. If the method is in another source file, that file is opened to the next line of source code.

To step into a method:

◆ Choose Step Into from the Debug menu. (Or press F8.)

To start debugging and pause at the first line, use this command. When used on an applet, Step Into takes you to the applet constructor.

Note: If the next line to be executed does not contain a method call, Step Into performs a single-step of the line.

Stepping over a method

Step Over executes one line of code. However, if the current statement is a call to a method, the debugger executes the method behind the scenes and returns to the next statement following the method call. Step Over executes the program to the next statement, unless a breakpoint or an exception is encountered before execution reaches that point.

By using the Step Over command, you can bypass opening other files unnecessarily and stepping into every line of code each time a method is called.

You can setp over a method call contained in your code after the debugger has hit a breakpoint. This causes the single step to execute the called method in its entirety and land on the next line of code in your source.

To step over a method:

◆ Choose Step Over from the Debug menu. (Or press F10.)

Note: If the next line to be executed does not contain a method call, Step Over performs a single-step of the line.

Stepping out of a method

Step Out returns to the calling method at the point after the current method was called, unless a breakpoint or an exception is encountered before execution reaches that point.

Note: If a variable is declared but not used, the debugger will step over that declared statement.

If you hit a breakpoint and want to execute the rest of that method returning to the caller, you can choose to step out of the currently executing method.

To step out of a method:

- Choose Step Out from the Debug menu. (Or press SHIFT - F11.)

Viewing and modifying variables, expressions, and methods

You can examine any of the variables in your program while in debug mode. You can modify the value of a variable from the Variables window, the Watch window, or the Evaluate Expressions dialog box.

You can watch a variable or expression in the Watch window. You can also see method calls in the Call Stack window.

Using the Variables window

The Variables window shows the variables — such as local variables, global variables, and objects — that are active in the current context. A **context** is the particular portion of your program on which Visual Cafe is focusing. This window is useful for examining errors that occur when your program passes parameters to methods. You can modify these variables directly in this window. When you pause the execution of your program, you can modify the value of a variable and then continue execution with the new value in place. You can examine objects and array data elements as well as simple data types.

Use the Variables window to:

- view the value of a variable
- view type information for a variable
- modify a variable

To open the Variables window:

- Choose Variables from the View menu.

Viewing the value of a variable

You can view the value of a variable by selecting its name from the Variables window.

To view the value of a variable:

1. Choose Variables from the View menu.

   The Variables window opens, displaying all the variables in the current context of the program.
2. Click on the variable you want to view.
3. To expand an object to see its contents, click the plus sign (+) to the left of the object you want to expand.

Viewing type information for a variable

You can view a variable's type from the Variables window.

To view type information for a variable:

1. Choose Variables from the View menu.

   The Variables window opens, displaying all the variables in the current context of the program.
2. Click on the variable whose type you want to view.

   The variable's type is shown in the Type column.

Modifying a variable in the Variables window

You can modify the value of a variable by selecting it in the Variables window and typing a new value in the Values column. When you pause the execution of your program, you can modify the value of a variable and then continue execution with the new value in place.

Note: You can modify a variable in the Variables window only when your program is paused in the debugger.

To modify a variable in the Variables window:

1. Choose Variables from the View menu.

   The Variables window opens, displaying all the variables in the current context of the program.

2 Click the variable you want to change.

3 Click in the Value column.

4 Type the new value in the Value box.

5 Press ENTER to save the change.

Note: To modify the value of an array or any structured type, edit the individual array's fields or elements. You cannot edit an entire array or structure at once.

Enabling or disabling ValueTips at debug time

ValueTips display the value of a variable in the Source window when you place your cursor over the variable at debug time. The variable has to be in in scope, or the area where the variable is applicable.

To enable or disable ValueTips at debug time:

1 From the Tools menu, choose Environment Options, then click the Debugging tab.

2 Select or clear the Enable ValueTips option.

3 If Enable ValueTips is selected, set the amount of time, in units of 1/10 of a second, that goes by before a tip displays.

Using the Watch window

In Visual Cafe, you can specify variables and expressions that you want to watch while debugging your program. The variables that you elect to watch are displayed in the Watch window, which is available only when you pause a running program. You can also modify the value of a variable using the Watch window.

Caution: Since the program is paused, do not enter watch expressions that rely on calls to other components.

To open the Watch window:

◆ Choose Watch from the View menu.

Adding a variable or expression to watch

You can watch a variable or expression while debugging by adding the variable name to the Watch window. You can add variables and expressions in several ways, as described in this section.

Caution: Because the program is paused, you should not enter a watch expression that relies on calls to other components.

To add a variable or expression by dragging and dropping:

◆ Drag an item from the Variables window to the Watch window.

To add a variable or expression by typing the name:

1 Choose Watch from the View menu.

   The Watch window displays.

2 Type a variable name or expression in the Watch window's Watch field. As you begin to type, the window changes to edit mode for the column of the selected row.

3 Press ENTER to save or ESC to leave the item unchanged.

To add a variable or expression from the Evaluate Expression dialog box:

1 In the Variables window, right-click the variable you want to watch.

2 Choose Evaluate Expression from the pop-up menu.

3 In the Evaluate Expression dialog box, click Add Watch.

   The Watch window evaluates and immediately displays the value of the variable or expression.

Modifying a variable or expression in the Watch window

You can use the Watch window to change the variable name or expression that you elected to watch at run-time. You can also use the Watch window to modify variables in place while your program is in debug mode.

Caution: Because the program is paused, you should not enter watch expressions that rely on calls to other components.

To locate the variable you want to modify:

1 Choose Watch from the View menu.

All the variables and expressions you've chosen to watch are displayed in the Watch window.

2 Click on the variable or expression you want to change.

The variable is selected and ready to be modified.

To change the value of a variable:

1 In the Watch window, select the variable that you want to modify.

2 Click in the Value field.

3 Type the new value in the Value box.

The variable is changed to use the new value.

Only primitive types of variables can be modified by clicking in the Value cell and entering a new value. After modifying a variable, you can continue debugging from the current line without having to stop and restart the debugging session.

Note: To modify the value of an array or any structured type, edit the individual array's fields or elements. You can't edit an entire array or structure at once.

To change the variable or expression to watch:

1 In the Watch window, select the variable that you want to watch.

2 Click in the Watch field.

3 Edit the variable or expression.

4 Press ENTER to save the change.

Deleting a variable or expression from the Watch window

When you've finished watching a run-time variable or expression, you can delete it from the Watch window.

To delete a variable or expression:

1 Choose Watch from the View menu.

All the variables and expressions you have chosen to watch are displayed in the Watch window.

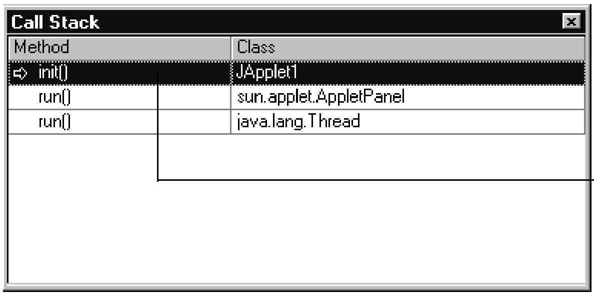
2. Click on the variable or expression that you want to delete from the list.
3. Press DELETE.

   The selected entry is deleted from the Watch window. You can also delete multiple entries.

Using the Call Stack window

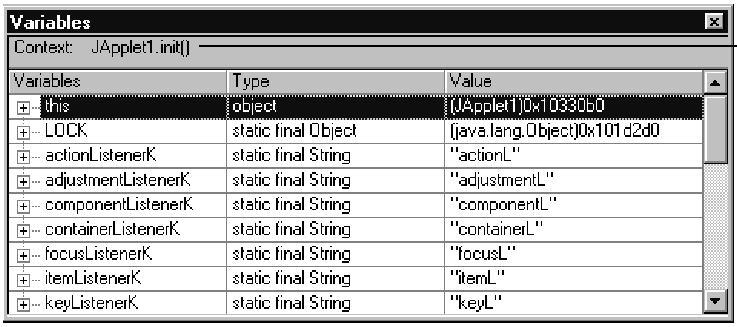
When debugging in Visual Cafe, you can view the stack of methods in your application that have started but not completed. These pending methods are displayed in the Call Stack window. The currently executing method appears at the top of the stack, and older function calls appear below that. You can also see the parameter types and values for each method on the call stack.

The active method is indicated by a black arrow next to it. By double-clicking an entry in the Call Stack window, you can change the context of the Variables window to display the variables for that method and its source.



Method	Class
⇨ init()	JApplet1
run()	sun.applet.AppletPanel
run()	java.lang.Thread

Click an entry in the Call Stack window...



Context: JApplet1.init()

Variables	Type	Value
this	object	(JApplet1)0x10330b0
LOCK	static final Object	(java.lang.Object)0x101d2d0
actionListenerK	static final String	"actionL"
adjustmentListenerK	static final String	"adjustmentL"
componentListenerK	static final String	"componentL"
containerListenerK	static final String	"containerL"
focusListenerK	static final String	"focusL"
itemListenerK	static final String	"itemL"
keyListenerK	static final String	"keyL"

...and the Variables window displays variables for that method.

To open the Call Stack window:

- ◆ Choose Call Stack from the View menu.

Viewing parameters for a method on the call stack

You can view the parameters passed to a method on the call stack when that method was called. Visual Cafe allows the display of both values and types for each parameter passed.

To view parameter types:

- ◆ Choose View Parameter Types from the Calls menu.

  This action toggles the display of parameter types in the Procedure column of the Call Stack window.

To view parameter values:

- ◆ Choose View Parameter Values from the Calls menu.

  This action toggles the display of parameter values in the Procedure column of the Call Stack window.

Viewing variables for a method on the call stack

You can view the values of variables for any method on the call stack. Only the variables in scope at the time the method entered the stack can be viewed.

To view the variables on the call stack:

1. Choose Call Stack from the View menu.

   The Call Stack window opens.
2. Click the method whose variables you want to view.
3. Choose Go to Variables from the Calls menu.

   This opens the Variables window, which lists the variables in the selected call.

Viewing source code for a method on the call stack

You can choose any method entered on the call stack and view the source code for that method.

To view the source for a method on the call stack:

1. Choose Call Stack from the View menu.

   The Call Stack window opens.
2. Click the method whose code you want to view.
3. Choose Go to Source from the Calls menu.

   This opens the Source window for the selected call.

Debugging threads

Java is a multithreaded language, which means that Java allows for more than one sequence of execution at a time. You might want to use threads to allow your Java program to talk to more than one client across networks, for example.

You use the Threads window to debug the threads in your program.

For an itroduction to the Threads window, see “About the Threads window” on page 6-6.

Using the Threads window

While you’re debugging, the Threads window lists the active threads, one per row. The currently selected thread is highlighted. You can change the selection by clicking a different row, or by using the Up and Down Arrow keys.

The *primary thread* is identified by a bold arrow in the left margin. This thread receives user input and is automatically created by the operating system when a process (an instance of ann application) is created.

The *active thread* — the one that was active when you entered break mode — is identified by a non-bold arrow in the left margin.

Note: If you’re using multiple threads in your program, each thread has its own call chain. To see the call chain for an individual thread, drag the thread from the Threads window and drop it on the Call Stack window.

When debugging a multithreaded program, you can select a single thread using the Threads window. The Threads window shows all the existing threads that your program has created, and the state of each thread. You can also update the Call Stack window with a single thread's call chain and update the Variables window to show only the variables within the current thread.

To open the Threads window:

- ◆ Choose Threads from the View menu.

Debugging a single thread

If you want to focus your debugging efforts on a specific thread to eliminate the behavior of others, you can set the focus to that thread and work on just that thread.

To debug a single thread:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread you want to work on.
3. From the Threads menu, choose Set Focus.

   The focus of the debugger is set to the selected thread.
4. Continue debugging that thread in the Call Stack window, Variables window, or Source window.

Suspending a thread

If you're working on a multithreaded program, you can suspend any thread if you suspect it of causing unwanted side effects while your program is running or while you're debugging.

To suspend a thread:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread you want to suspend.
3. Choose Suspend from the Threads menu.

   The selected thread is suspended.

Resuming a suspended thread

If you've suspended any threads to temporarily eliminate their behavior from your program, you can resume any one you choose from the list displayed in the Threads window.

To resume a suspended thread:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread you want to resume.
3. From the Threads menu, choose Resume.

   The suspended thread is resumed.

Suspending other threads

If you want to focus your debugging attention on a single thread, Visual Cafe allows you to suspend all the other threads to narrow down the behavior of your program.

When debugging multiple threads, the timing of the threads may be different than at run time because of the memory overhead used in debugging. For example, say you have a program that prints the numbers 1 through 10 in numerical order, with one thread responsible for printing the odd numbers and the other thread responsible for printing the even numbers. When you execute this program, the numbers will appear in correct sequence. When debugging, they might print as 1, 3, 5, 2, 4.

To suspend other threads:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread you want to want to work on.

   Note: You're causing all threads except the selected thread to be suspended.
3. Choose Suspend Others from the Threads menu.

   All threads in the debugger except the one that's selected are suspended.

Resuming other suspended threads

If you've suspended all but one thread in a multithreaded program (see the preceding section, "Suspending other threads"), you can resume all other threads at any time.

To resume other suspended threads:

1 Choose Threads from the View menu.

The Threads window opens.

2 Click the thread that you don't want to resume.

Note: You're causing all threads except the selected thread to resume.

3 Choose Resume Others from the Threads menu.

All threads in the debugger except the one that's selected are resumed.

Viewing the source code for a selected thread

If you want to look at the source code for a specific thread, you can do so from the Threads window.

To view the source code for a selected thread:

1 Choose Threads from the View menu.

The Threads window opens.

2 Click the thread you want to focus on.

3 Choose Set Focus from the Threads menu.

Set Focus updates the Source window.

4 If the Source window isn't already open, choose Source from the View menu.

The thread's source code is displayed in the Source window.

Viewing the active variables in a thread

You can view the values of variables for any thread that's currently executing.

To view the active variables in a thread:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread you want to focus on.
3. Choose Set Focus from the Threads menu.

   Set Focus updates the Variables window.
4. If the Variables window isn't already open, choose Variables from the View menu.

   The Variables window opens, displaying the chosen thread's variables.

Viewing the call stack for a thread

If you're working on a specific thread and want to see the call stack for just that thread, Visual Cafe allows you to do so.

To view the call stack for a thread:

1. Choose Threads from the View menu.

   The Threads window opens.
2. Click the thread whose call stack you want to view.
3. Choose Set Focus from the Threads menu.

   Set Focus updates the Call Stack window.
4. Choose Call Stack from the View menu.

   The Call Stack window opens on the thread you selected.
5. Click the method whose code you want to view.

Handling exceptions

Java uses exceptions to process program errors. An **exception** is an event that occurs during the execution of your program that interferes with, disrupts, or stops the normal flow of instructions.

Throwing exceptions

Many types of errors — from simple programming errors to a hard disk crash — force the Java run-time system to throw exceptions. When such an error occurs within a Java method, the method creates an exception component and passes it to the Java run-time system.

This exception component contains important information about the exception, including its type and the state of the program when the error occurred. The run-time system tries to find a piece of code to handle the error. This process of creating an exception component and passing it to the Java run-time system is called **throwing an exception.**

Catching exceptions

After a method throws an exception, the Java run-time system finds a way to handle the exception. One place to handle exceptions is the set of methods in the call stack of the method where the error occurred. The Java run-time system scans backwards through the call stack, starting in the method where the error occurred, until it locates a method that has a suitable **exception handler**. An exception handler is suitable if the type of the exception thrown is the same as the type of the exception handled by the exception handler.

The exception moves up through the call stack until a suitable handler is found and one of the calling methods handles the exception. This is called **catching an exception.** If the run-time system searches all of the methods on the call stack without encountering a suitable exception handler, the run-time system and the Java program terminate.

Setting exceptions

When you run your program, you can choose to have all exceptions break into the debugger, to stop at a particular expression, or to have only unhandled exceptions break into the debugger.

To stop whenever a particular exception occurs:

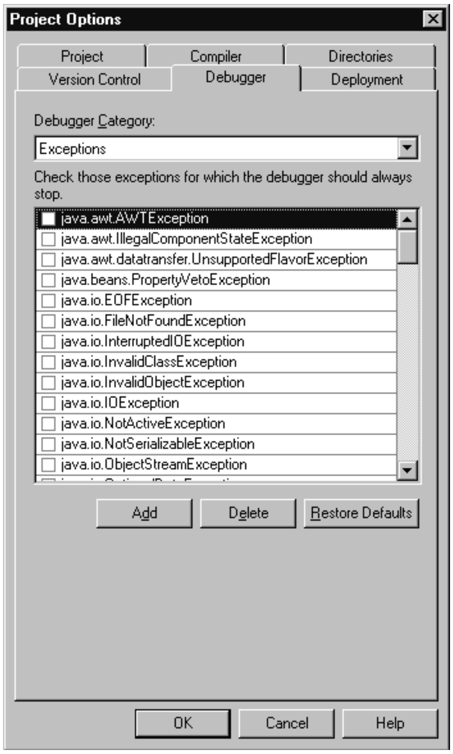
1 Choose Options from the Project menu.

The Project Options dialog box appears.

2 Click the Debugger tab.

3 From the Debugger Category drop-down list, choose Exceptions.

A list of possible exceptions displays.



Debugger Category:
Exceptions

Check those exceptions for which the debugger should always stop.

- ☐ java.awt.AWTException
- ☐ java.awt.IllegalComponentStateException
- ☐ java.awt.datatransfer.UnsupportedFlavorException
- ☐ java.beans.PropertyVetoException
- ☐ java.io.EOFException
- ☐ java.io.FileNotFoundException
- ☐ java.io.InterruptedIOException
- ☐ java.io.InvalidClassException
- ☐ java.io.InvalidObjectException
- ☐ java.io.IOException
- ☐ java.io.NotActiveException
- ☐ java.io.NotSerializableException
- ☐ java.io.ObjectStreamException

4 Check the box to the left of the exception name to select that exception.

The program stops whenever the exception is encountered, regardless of whether or not your program handles the error.

5 (Optional) To add an exception, click Add and then type in the name of the exception.

6 (Optional) To delete an exception, select the exception and click Delete.

7 (Optional) To restore the default settings, click Restore Defaults.

This restores the defaults for all exceptions, thereby disabling stopping at all exceptions except `java.lang.ArrayIndexOutOfBoundsException` (probably the most common exception error).

The default behavior for exceptions is for the debugger to stop only if a particular exception isn't handled in the source code.

Using incremental, browser, and remote debugging

You can check your program for errors or complications in addition to regular debugging features. If you're using Visual Cafe Professional or Database Edition, you can edit your program and immediately see the effects in the debugger. You can also run your applets in a Web browser, which can be helpful because not all Web browsers work with applets in the same way. Another useful option is that you can run your program on one computer and debug it on another, to further simulate the end user's experience with your program.

For information on debugging native programs, see "Debugging native programs" on page 11-6.

Using incremental debugging

Visual Cafe has several **incremental debugging** features that make debugging your Java programs more efficient. While your program is executing or paused in the debugger, you can edit it and immediately see the effects of the code change. This is also called **run-time editing**. The code is compiled and saved automatically.

In order to make use of this feature, you need to first enable Visual Cafe for incremental debugging.

To enable incremental debugging:

1. From the Tools menu, choose Environment Options, then click the Debugging tab.
2. Select the appropriate options:
   - ❖ Always compile changes and update program, no prompt – Enables run-time editing so that choosing a Save menu item causes new changes to be compiled and reflected in an executing program. If the program is paused, resuming it causes all changes to be applied and saved. (This is the default.)

- ❖ Prompt before compiling changes and updating program – Enables run-time editing like the previous option, except that you are prompted first so that you can choose to stop run-time editing.
- ❖ Always ignore changes – Disables the run-time editing feature. (Visual Cafe version 1.0 worked this way.)

To incrementally debug your program:

◆ While you have your program open in Debug mode, do any of the following:

- ❖ To save all the changes that you've made to your files, choose Update Now from the Debug menu. This forces an incremental update and saves all files.
- ❖ To go back to the method that called the currently active method, choose Restart Method from the Debug menu. You can then restart the current method from the beginning.

Note: You should not think of the Restart Method command as a type of undo command, because it cannot undo some edits, such as variable edits. It does not undo side effects of code that was run (if part of a program runs two times and causes an exit, for example).

If you change a portion of code that is active (anywhere on the call stack), the code is compiled and saved and you'll see a dialog box that asks what action you want to take:

- ❖ Restart the program – Start the program in the debugger again.
- ❖ Restart the active method – Start the active method again. (This option is valid only if you perform run-time editing while your program is paused.)
- ❖ Continue – Continue with the old code until the next time the code becomes active. Ignore breakpoints in this code until the code becomes active again.
- ❖ Stop debugging – Exit the debugger.

The recommended action is already selected for you in the dialog box.

Note: When you're debugging native code you can add new methods, while with bytecode you cannot. For more information, see "Debugging native programs" on page 11-6.

Debugging applets in a Web browser

Applets and applications may run differently in different Web browser Virtual Machines (VMs), so you can save time and effort by debugging in a browser to preview these differences. Debugging in a Web browser allows you to test your applet in the environment where the applet will be used. That way, you'll be able to refine your code to reflect differences in the virtual machines of the supported browsers.

Another advantage of running your programs in a Web browser is that you can make use of classes that are only supported by that particular Web browser's VM. For browser debugging to work, you must have either version 4.04 of Netscape Navigator (with the JDK 1.1 PR3 or newer plugin), or Internet Explorer 4.0 or higher.

Note: This feature is available only in Visual Cafe Professional and Database Editions.

To turn on debugging in a Web browser:

1. Activate the Project window of the project you want to work with.
2. Choose Options from the Project menu.
3. In the Project Options dialog box, click the Debugger tab.
4. In the Debugger Category drop-down list, choose General.
5. Select either Netscape Navigator or Microsoft Internet Explorer/Jview.

   The default is to debug using Sun Java vm (the Visual Cafe AppletViewer).

   See the following set of procedures for more information about debugging with Netscape Navigator.
6. Choose Run in Debugger or Step Into from the Project menu to start your debugging session in the selected Web browser.
7. Click OK.

   The changes take effect immediately.

To allow debugging in Netscape Navigator:

1. Install Netscape Navigator 4.04 or later.
2. Install the Netscape JDK 1.1 Support Patch for 4.03 or greater.

3. Change to your current Navigator user directory and edit your `prefs.js` file to allow local applet access to the hard drive. In `...\netscape\user\`*username*`\prefs.js`, add the following text:

   ```
   user_pref("signed.applets.low_security_for_local_classes
    ", true);
   ```
4. This allows Netscape Navigator to access the Visual Cafe applet on your hard disk.

Considerations for browser debugging

Here's some information that you should be aware of when debugging in a Web browser:

◆ Netscape Navigator can only debug applets; if you change the project type to Java application, then you'll notice that the debug using option will be set back to Sun Java VM and the Netscape Navigator option will be unavailable. Internet Explorer supports both applets and applications.

◆ If Visual Cafe can't detect the supported version of either Web browser, the Web browser option is unavailable.

◆ If you save a project that has the debug in Netscape Navigator or Microsoft Internet Explorer/Jview project option selected, and then you remove Netscape Navigator or Microsoft Internet Explorer and try to debug your project, you'll see a message stating that the browser is not installed and that the Sun Java VM will be used instead. This dialog box has a checkbox that allows you to permanently change the option for the project to Sun Java VM to prevent the message from appearing in the future.

◆ The following options are not allowed while debugging in Netscape Navigator and Internet Explorer: incremental debugging, expression evaluation, and remote debugging.

Debugging programs on a remote computer

With Visual Cafe, you can run a program on one machine and debug it remotely on another. To do this, Visual Cafe, the class files, and the HTML files need to be on the remote computer. You might find remote debugging useful, since you can see the program's user interface on one computer and the debugging information on the main computer.

Debugging a program on a remote computer works just as it does when you debug and run on the same computer. The only difference is that you cannot perform runtime editing or make any changes to the remote program while you're debugging it remotely.

Note: This feature is available only in Visual Cafe Professional and Database Editions.

Setting up for remote debugging

Before beginning a remote debugging session, you must properly configure the local and remote machines.

Note: You can debug an applet or application remotely on another machine. You cannot debug native applications or DLLs remotely.

To configure the local and remote machines, the following conditions must apply:

- ◆ Visual Cafe must be installed and running on both machines.
- ◆ The TCP/IP networking protocol must be installed and configured on both computers.
- ◆ Identical copies of the project must be on both machines. You must have the class files on the remote machine, although you don't necessarily need the source files.

The class files you're debugging reside on the remote computer, and not on the computer where you're running the debugger. Therefore, to have a successful debugging session you need to be sure that the remote computer has the debug build of the classes you need.

Note: Remote debugging first looks in the Visual Cafe `sc.ini` file for class path information, and then in the Windows class path. This information must be correctly configured in order for remote debugging to function properly. For more information, see "Setting environment variables in the sc.ini file" on page 3-72. Make sure the class path is set correctly in `autoexec.bat` for Windows 95 and 98 or in the Control Panel for Windows NT.

If you're going to remotely debug an applet, you need to specify the absolute path to the associated HTML file in the Project tab of the Project Options dialog box (unless the HTML file is located in the remote computer's root directory). Specify the absolute path to the HTML file in the Start with Web page option. For more information, see "Specifying an applet's HTML file" on page 5-3.

Note: You cannot perform remote debugging in a Web browser.

Starting remote debugging

When both computers are configured correctly, you're ready to begin your remote debugging session. Visual Cafe uses the `debugvm.exe` file to execute remote debugging.

To debug an applet or application on a remote computer:

1. Open a console session (DOS window) on the remote machine.
2. Navigate to the directory containing the class and HTML files of the program you intend to debug.
3. Execute `debugvm`.

   `Debugvm` displays some version information, and then a password and the machine's IP address.

   Remote debugging is now enabled for the remote machine.
4. On the local computer that's running the debugger, load the project that corresponds to the class and HTML files you're debugging on the remote computer.
5. Choose Debug in Waiting VM from the Project menu.

   The Remote Debug dialog box appears.
6. Enter the Host Address and Password as reported by `caferemote`.

Field	Description
Host	The IP address of the remote computer.
Password	The remote computer's password, which is returned by `debugvm.exe` run on the remote computer.

7 You can now begin to debug the remote computer's program from your local computer. You can click the standard debug commands Go to Breakpoint or Step Into to do so.

Ending remote debugging

Once you're finished debugging the remote program, you can stop the debugging session.

For information about native debugging, see "Debugging native programs" on page 11-6.

To terminate a remote debugging session:

◆ On the remote computer, select the `debugvm` console window and press CTRL–C.

  This terminates the debugging session that's currently in process.

To continue debugging, you need to reexecute `debugvm` in order to get a new password.

II

Using Components

C H A P T E R 7

Working with Components

This chapter shows you how to use components in Visual Cafe. Much of the information applies to all types of components you'll use in Visual Cafe: AWT, Swing, and JavaBeans components. Where indicated, the information here pertains only to AWT-based components. For information about working with Swing components in Visual Cafe, see Chapter 8, "Working with JFC/Swing Components." For information about using JavaBeans components, see Chapter 8, "Working with JFC/Swing Components."

Visual Cafe makes it easy to design your graphical user interface (GUI) by allowing you to visually lay out your applets and windows. To design your GUI, you must first create a Visual Cafe project. Then you can add components, such as windows, frames, menus, dialog boxes, text, buttons, and graphics, to the project. (See "About components" on page 7-2 for information.) Some components contain other components, and some components must be contained by another component. For example, a button must be contained by another component, such as an applet, window, or dialog. You can add components to a project from the Component Library or Component Palette (see "About the Component Library" on page 7-7 and "About the Component Palette" on page 7-11 for more informaton).

You can also open top-level components, or forms, in the Form Designer, then visually arrange components on a form. (See "About forms" on page 7-20 and "About the Form Designer" on page 7-20 for more information.)

In addition, you can:

◆ set component properties, such as color and text

- use customizers to edit components when you want a visual interface for changing properties
- create interactions between components

About the Java AWT

The **Java Abstract Windowing ToolKit,** or **AWT,** is a standard and portable GUI library that you can use to create visual "front ends," or user interfaces. This GUI library is cross-platform for developing and running applications and applets. Visual Cafe has integrated the AWT into its visual design environment so you can quickly create your user interfaces. You can view the AWT components by expanding `java.awt` in the Packages view of the Project window.

The AWT contains a set of pre-built components that encapsulate the core user-interface elements that can be used by your applet or application. The AWT is a cross-platform library; its methods and classes are abstracted to remove any dependencies on a particular operating system. It's a Java package that can be used in any Java program by importing `java.awt.*` using the `import` keyword. Visual Cafe automatically imports this package for you.

About components

Components allow users to interact with your program. **Components** are reusable objects that you add to your projects, such as:

- Scroll bars
- Buttons
- Checkboxes
- Text-entry fields
- Menus
- Graphics files

There are three types of components in Visual Cafe: visual components, non-visual components, and containers. A **visual component** is visible at run time and lets users interact with your applet or application; it has a screen position, a size, and a foreground and background color. Examples of visual components are forms, applets, and buttons. A **non-visual component** is not visible at run time (a timer, for example) displays differently at run time (a menu bar, for example). Non-visual components are also called **invisible components**. Some components can contain other components, such as an application window that contains a button; these components are called **containers.**

In Visual Cafe, you add components to forms to assemble applets and applications. You can drag and drop components into other components, creating a project in a visual manner.

Components can accept input from a user and perform specific actions (for example, a user could click a button that caused an animation to play). Visual components can also be used to display the results of an action (for example, clicking a button could display the results of a mathematical operation).

Visual components generally have the following attributes:

- A set of properties – Governs how components display and behave. Properties are accessible from the Property List.
- A visual element – Defines the appearance of a component at run time. The component is shown as an icon in the Project window; you can edit the component by double-clicking the icon to open it.
- One or more interactions – A relationship between two components is called an *interaction*. As you create interactions by connecting components, Visual Cafe automatically generates code for the relationship, allowing you to assemble interactive applets and applications without writing code. Interactions are not an integral part of all components. (For more information, see Chapter 9, “Working with Events and Interactions.”)
- A set of event method(s) – In Visual Cafe, interactions are implemented as methods and imply an event notification. You can access this Java code from the Source window.

Each of these component attributes has its own editor, from the Form Designer to the Interaction Editor, making it easy to manipulate visual components. You create cross-platform Java applets and applications by first designing the user interface components and then using the various Visual Cafe tools to automate the process of deriving classes, creating interactions, and mapping functions to visual components and messages.

You can do most of these tasks by setting properties to define, refine, and control the appearance and behavior of your visual components.

About top-level components

A **top-level component** is a component that appears at the top level of the Project window's Objects view. Top-level components include `Applet`, `Frame`, `Window`, and some dialog components. In the Component Library, the top-level components are all in the `Forms` group. You can position other components, such as text, buttons, and graphics, on the top-level component in the Form Designer, making the top-level component a container for these components.

A top-level component corresponds to a Java source file. Visual Cafe automatically creates the Java code and updates the code as you visually design your GUI.

The top-level component for an applet is the `Applet` component. The `init` method, which is called by another program (such as a Web browser), is the entry point of the applet. If you use the AWT Applet project template provided with Visual Cafe, the applet is already set up for you programmatically.

The top-level component for an application with a GUI is the `Frame` component. The `main` method, usually called from the command line, is the entry point of the application. If you use the AWT Application template provided with Visual Cafe, the main window is already set up for you programmatically.

`Component` is the parent class from which all visual components and containers are derived. This class defines objects with properties (size and position, for example), methods, and events that enable the objects to be rendered on the screen and respond to events.

All visual components in the Component Palette are component subclasses. `Component` is an abstract class and cannot be instantiated.

With Visual Cafe, you can define a component in several ways:

◆ You can subclass `Component` directly (thereby making a lightweight component; see "About lightweight and heavyweight components" on page 7-6 for more information), provided you define a public constructor

- You can subclass any standard AWT component subclasses
- You can subclass one of your own component subclasses, or one of the Beans in the Component Library

About containers

Java defines a **container** as a component that can hold any number of components, such as text, buttons, and graphics. A container can also hold other containers. Containers allow you to group related components and treat them as a single unit, reducing the amount of programming you have to do.

Containers hold and organize your components, but they may also contain code for event handling and many essential tasks such as changing the cursor's appearance and the program's icon. All containers support operations such as adding, removing, and painting the components they contain.

In general, while components are independent objects, there is a certain parent-child relationship that exists between containers and other components. To better understand the idea of containers, think of all visual components as children of the form on which they are displayed. Most components inherit the read-only parent property, which displays the form. The placement of visual components is also relative to the parent form. Visual components cannot be moved outside the boundaries of the parent; moving a form moves the child components as well.

The top-level containers, or forms, are `Applet`, `Frame`, `Window`, and some dialogs. Applications are built on `Frame` containers and applets are built on `Applet` containers. Other containers are panels, `MenuBar`, `Menu`, and some dialogs. (A **panel** is a container that you use to group a window into logical regions.) These containers can be contained by a form, but are not forms themselves.

When you select an applet template, Visual Cafe automatically provides you with an `Applet` component as the parent container for your project. When you select an application template, Visual Cafe automatically provides you with a `Frame` component as the parent container for your project.

The top-level containers are described in the following table:

Container	Function
`Window`	A window.
`Frame`	Extends `Window`. It supports a title bar and menu bars, can be minimized, and must be a parent container.
`Dialog`	Extends `Window`. A dialog box that can be shown and dismissed apart from its parent window.
`Applet`	A panel that can be embedded in a Web page. `Applet` containers are parents to applet programs. Applets cannot be nested.

In the AWT, components are added to containers and then arranged by layout managers (see "Arranging components" on page 7-37 for more information). In addition to components and containers, there are a variety of event handling, menu, fonts, and graphics classes. The AWT also works well in conjunction with the networking and threads classes.

About lightweight and heavyweight components

A direct subclass of `Component` is a lightweight component. A *lightweight component* is written in pure Java, and therefore isn't defined by a native-code peer component, as are *heavyweight components.* A lightweight component does not necessarily draw its background color, which means that by default lightweight components are transparent. A direct subclass of `Container` is also lightweight. A subclass of any other AWT component class is heavyweight.

Heavyweight components always display over lightweight components that are in the same container. This is because lightweight components use the drawing context of their nearest heavyweight container, while heavyweight components create their own drawing context. For information on mixing lightweight and heavyweight components, see "Mixing lightweight and heavyweight components" on page 8-26.

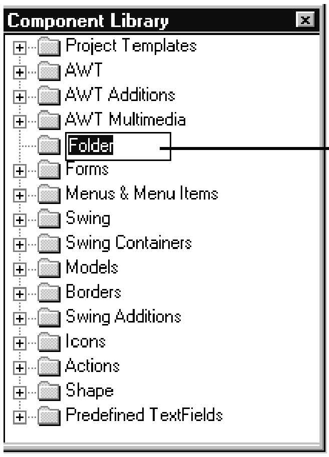
About the Component Library

The **Component Library** is a repository for storing, organizing, and displaying components and project templates. When you install Visual Cafe, all of the standard Java components plus many additional Symantec components and project templates are displayed in the Component Library.

The items in the Component Library are reusable. Once you design a form or component and place it in the Component Library, you can use it in other projects. For example, you can start a new project with a project template you designed, then drag components from the Component Library into the Project window or Form Designer to use them in the new project.

To display the Component Library:

◆ Choose Component Library from the View menu.



Create a new group of components just as you would create a new folder in Windows Explorer.

You can also easily rename or delete groups from the Component Library, just as you would in Windows Explorer.

Using the Component Library

You have a good deal of control over the components in the Component Library. In the following sections you'll learn how to add components – including custom components – to the Component Library, rearrange the components in the Library, and delete components from the Library.

Adding components to the Component Library

To easily access components, you can add them to the Component Library. For information about adding a JavaBeans component to the Component Library, see “Automatically updating Beans in the Component Library” on page 10-19 and “Adding an existing Bean to the Component Library” on page 10-21.

To add a component, you insert a `.class` or `.jar` file.

Note: Some components have more than one class file, such as if the Java source file for a component has inner classes. You need to make sure that these class files are in the class path. You do not need to add inner classes to the Component Library.

An **inner class** (or **nested class**) is a class that’s included within the body of another class, even within a method (called a local class). This feature is new with JDK 1.1 and is useful for creating adapter classes. After compilation, the inner class ends up in its own class file, which has a dollar sign ($) in its name.

To add components to the Component Library:

◆ Choose Add Component into Library from the File menu and insert a `.jar` or `.class` file.

or

◆ Drag and drop a `.jar` or `.class` file from a file system window, such as the Windows Explorer.

Caution: After you add a component, you should not move its corresponding `.class` or `.jar` file, because Visual Cafe looks for it in that location. If you move the component, you should re-add it to the library. Some components are made of more than one class file (for example, if the Java source file for a component has inner classes). You need to make sure these class files are in the class path. You do not need to add inner classes to the Component Library.

Creating a component template

Another way of adding a component to the Component Library is to create a component template. You can then use this component template as a starting point for other components you create.

To create a component template:

◆ Drag a component from the Project window (Objects view) to the Component Library.

  The source file is copied by Visual Cafe, so you don't have to keep the files in the same location. A component and a component template appear the same in the Component Library, and you add them to projects in the same way.

  For information about creating project templates, see "Creating a project template" on page 3-28.

Adding custom components

You can customize the Component Library by adding third-party components and your own custom components and project templates. For example, if there's a certain type of form you use often, you can create it and then add the form – including the components it contains and its properties – to the Component Library. You can also edit the default behaviors of a component by selecting it in the Component Library and modifying its properties in the Property List (see "About the Property List" on page 7-34 for more information).

If you've modified a Visual Cafe component (see "Modifying component properties" on page 7-36) and want to place it in the Component Library, all you have to do is drag the component from the Project window's Objects view into the Component Library. However, if you want to add custom components that are not based on the Visual Cafe components, you can add them to Visual Cafe as follows.

To add custom components to a project:

◆ Insert the components' source files into your project

or

◆ Add the custom components using the JavaBean Wizard. (See "Creating a Bean" on page 10-10 for more information.)

Note: You only need to add custom components to the Component Library if you want to visually select and use the custom component with Visual Cafe views. Custom components can be referenced in the program's Java code without being integrated into Visual Cafe.

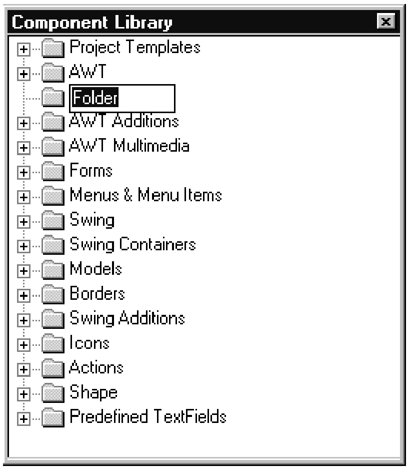
Adding a group to the Component Library

Components are stored in groups, represented by a folder icon, so you can organize your components.

To create a new group in the Component Library:

1. From the File menu, choose Window, then Component Library.
2. Choose Group from the Insert menu to add a new group to the Component Library.

   A new folder appears in the Component Library, as shown here:



3. Click on the new folder and enter a name for the new group.

   The new group is displayed in the Component Library.
4. Add components to the new group by dragging them into the folder.

Moving components within the Component Library

You can rearrange components within the Component Library. Simply select the component (in the Objects view of the Project window or in the Form Designer) and drag it to a new location in the Component Library.

To rearrange components and groups already in the Component Library, simply select a component or group and drag it to a new location.

Deleting components from the Component Library

As you're developing your project, you can delete user-created components from the Component Library. Deleting a component from the Library also removes the component from the Component Palette (see the following section).

To delete a component from the Component Library:

1 In the Component Library, select the component yu wish to delete.

2 Choose Cut from the Edit menu (or press the DELETE key).

Note: All JavaBeans components in a JAR file (as specified in the manifest file) are added to the Component Library. You cannot remove one component independently of the others in the JAR file. Instead, you need to recreate the JAR and not include the unwanted component. For more information, see "About JAR files" on page 5-54.

About the Component Palette

At the top of the main Visual Cafe window is the **Component Palette,** which presents a number of icons that you can click to work with components. You can think of the Component Palette as another view of the Component Library, but with easier access and more customization capabilities. The Component Library contains everything that the Component Palette contains, but the Component Library can contain more (such as project templates, which don't appear in the Component Palette).

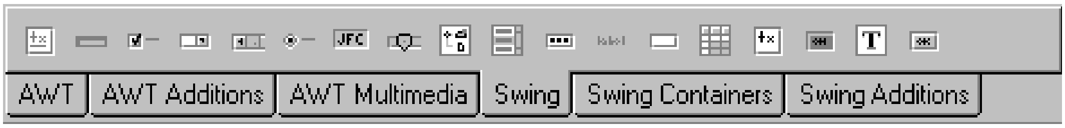
Like the Component Library, the Component Palette contains a variety of reuasable components and form templates. You can drag components from the Component Palette to the Form Designer or Project window to add them to a project.

Visual Cafe provides an extensive collection of custom and third-party components that you can quickly add to your forms from the Component Palette. You can add groups as well as individual components. When you add groups to the Component Palette, they are arranged into tabs.

You can add any component from the Component Library to the Component Palette and easily use the component in your applets and applications. You can add components in other ways as well; see “Adding a component or group to the Component Palette” on page 7-16 for further information.

If you wish, you can customize the Component Palette by using the Environment Options dialog box, which allows you to add and remove components, create new tabs, and reorganize components within groups. For more information, see “Customizing the Component Palette” on page 7-13.

Here’s an example of what the Component Palette looks like:



Similar components (such as Swing, AWT, and so on) are grouped together into tabs.

Also on the Component Palette are these buttons:

The **Selection Tool** is enabled by default. You can click this icon to return to component-selection mode if you’re not in it already. To select a component in the Form Designer, click on it with the Selection Tool arrow or use the Selection Tool to surround the entire component with a selection rectangle.

The **Interaction Tool** lets you create an interaction between two components by visually connecting them. After you complete an interaction, the Selection Tool (see above) is selected and the Interaction Tool is not selected. For more information about interactions, see Chapter 9, “Working with Events and Interactions.”

Component Palette display options

You can control the Component Palette's position and visibility by docking, floating, resizing, or hiding it.

To float the Component Palette:

◆ Drag the Component Palette from the top of the Visual Cafe window onto your desktop.
or

◆ Double-click somewhere in the Component Palettes's background.

To dock the Component Palette:

◆ Drag the Component Palette to the top of the Visual Cafe window.
or

◆ Double-click somewhere in the Component Palettes's background.

To show the Component Palette:

◆ Right-click in the background of the main Visual Cafe window and select Component Palette from the pop-up menu.

To hide the Component Palette:

◆ Right-click in the background of the main Visual Cafe window and deselect Component Palette in the pop-up menu.
or

◆ Click the Component Palette's close box.

Customizing the Component Palette

You can customize the Component Palette in a variety of ways, adding components and groups, moving components and groups, or renaming tabs.

You can also customize the Component Palette to contain the objects that you use most often. Visual Cafe provides a extensive collection of custom and third-party objects that you can quickly add to your forms, including grids and tabbed dialog boxes.

You customize the Component Palette from the Component Palette page of the Environment Options dialog box.

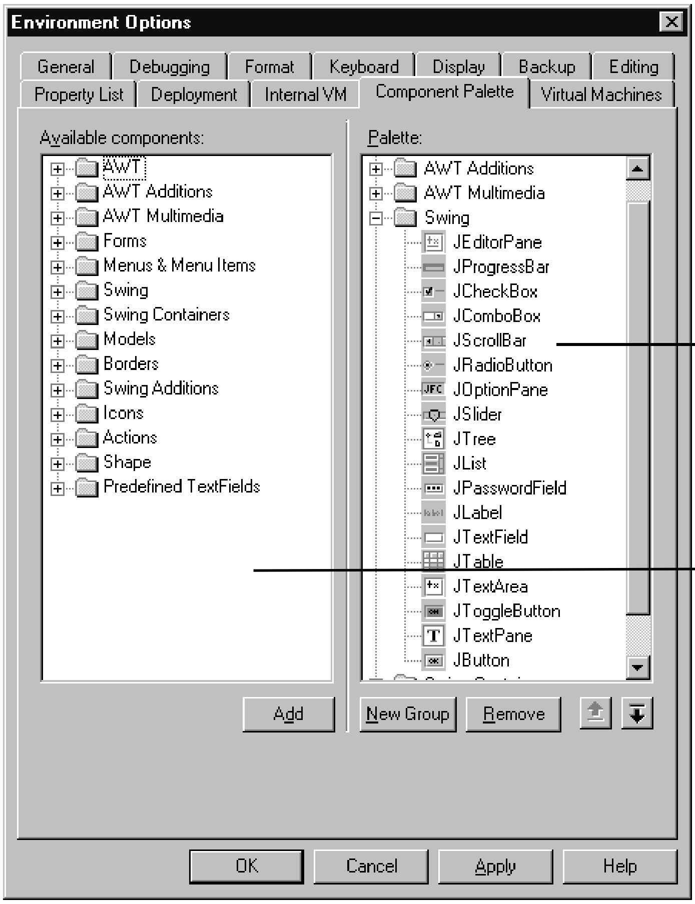
To access the Component Palette page in Environment Options:

- Choose Environment Options from the Tools menu, then click the Component Palette tab in the Environment Options dialog box.

  or

- Right-click the Component Palette, then choose Customize Palette.

The Component Palette page in the Environment Options dialog box appears. Here's an example of the Component Palette page:



In the right pane you'll see what items are in the Component Palette.

In the left pane you'll see the items available in the Component Library.

The Component Palette tab provides two panes for component display. The Available Components pane contains the same set of components as the Component Library. The Palette pane represents the Component Palette's tabbed toolbar.

Folders in the Palette pane represent groups, which are displayed in the Component Palette as tabs, and the components in each folder display as icons in the Palette pane.

To customize the Component Palette, you can:

- add components

- delete components
- group components in tabs
- create or remove a tab

These features are discussed in the following sections.

Adding a component or group to the Component Palette

You can add a component or a group – which will appear as a tab – to the Component Palette in one of two ways: by using drag-and-drop or by using the Add button. You can also drag components from the Project window to the Component Palette.

To add a component or group by dragging it from the Environment Options dialog box:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab in the Environment Options dialog box.

   The contents of the Component Palette are displayed in the Palette pane.
2. Select a component or group from the Available Components pane and drag it onto a Component Palette group or into the general Component Palette area.

   Dragging a component adds it to the current Component Palette group. Dropping the component onto another component adds the selected component to the same group as the target component.

   Dragging a group creates a new tab in the Component Palette.

To add a component or group by using the Add button in the Environment Options dialog box:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab.

   The contents of the Component Palette are displayed in the Palette pane.
2. Select a group in the Palette pane.
3. Select a component in the Available Components listing.
4. Click the Add button.

   The selected component is added to the selected group.

To add a group by using the New Group button in the Environment Options dialog box:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab.

   The contents of the Component Palette are displayed in the Palette pane.

2. Click the New Group button.

   Type a name for the group item.

Note: After you've added a new component to a new tab in the Component Palette, the group and the component are added to the Component Library.

To add components from the Project window:

1. Select the component in the Project window.
2. Drag-and-drop the component onto the tab of the Component Palette where you want the component to be stored.

Note: After you've added a new component to a new tab in the Component Palette, the group and the component are added to the Component Library.

To add components from the Component Library:

Use one of these methods:

- ◆ Drag the component from the Component Library and drop it on the tab of the Component Palette where you want the icon placed. Visual Cafe displays a message if the addition duplicates an existing component on the Component Palette. Duplicates are not allowed in the same tab.
- ◆ Select one or more components, then right-click and choose Add to Palette. This command is available only if the selected components are not currently on the Component Palette.
- ◆ Drag-and-drop a group from the Component Library to create a tab on the Component Palette. The tab contains all components in the group.

Moving components on tabs

You can move Component Palette components within their group or to another group. (Groups are represented by tabs.)

Within the Component Palette toolbar, you can drag and drop components within the same tab to reorganize their display.

To move components within the same tab:

1. Choose Tools, Environment Options, then select the Component Palette tab.
2. Select a component and use the Up and Down Arrow buttons to reorganize the display order.
3. Click OK.

   The changes take effect immediately.

Tip: You can also drag and drop components within the same group to reorganize their display.

To move components to another tab:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab.
2. Select a component and drag the component to a new location.
3. Use the Up and Down Arrow buttons to reposition the component.

   The component is now in its new location.
4. Click OK.

   The changes take effect immediately.

Deleting a component or group

As you use Visual Cafe, you may need to remove objects from the Component Palette if you no longer need them on a frequent basis.

You can delete components or group tabs from the Component Palette tab of the Environment Options dialog box, or you can delete tabs directly from the Component Palette itself.

Note: Deleting a tab deletes the group and all components within the tab. However, it does not delete them from the Component Library.

To delete a component from the Component Palette directly:

1. Click a component on the Component Palette.
2. Right-click and select Remove Component.

   The component is removed from the Component Palette.

To delete a tab from the Component Palette directly:

1. Right-click a tab on the Component Palette.
2. Select Remove Tab.

To delete a component or tab from the Component Palette by using the Environment Options dialog box:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab.
2. In the Palette pane, select the component or group to delete.
3. Click the Remove button or press the DELETE key.
4. Click OK or Apply.

   The component or tab is removed from the Component Palette.

Renaming tabs

As you organize components on the Component Palette, you may find that you need to rename a component tab.

To rename a tab:

1. Choose Environment Options from the Tools menu, then click the Component Palette tab.
2. Slowly double-click the tab (or select the tab and press F2), and type a new name.
3. Click OK or Apply.

   The changes take effect immediately.

About forms

Forms are the basis for creating a user interface to your applets and applications. Forms can be windows that display information and can receive user input. A form can contain labels that display text, or can contain components that provide interaction with the program. The most common types of forms are:

- Window
- Frame
- Dialog
- Applet

See "About top-level components" on page 7-4 for more information on these form types.

You can also use forms as containers for items that are not visible components in a user interface. For example, you can have a form in an application that serves as a container for other components that will be used in other programs. Or, you can have a container that contains code that handles events like mouse interactions. See "About containers" on page 7-5 for more information.

If you look in the Component Library, the forms provided with Visual Cafe are all in one Forms group.

About the Form Designer

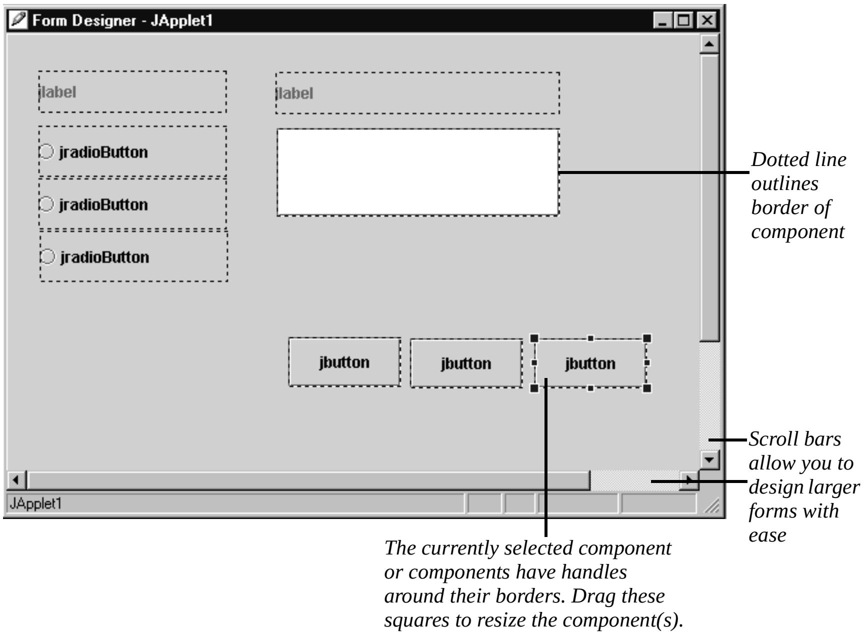
The **Form Designer** is the main tool for designing the interface to your applet or application. You use it to design windows, dialog boxes, message boxes, and other visual components. The Form Designer uses an integrated Java Virtual Machine (VM) to run Java code at design time. This allows the Form Designer to provide a true what-you-see-is-what-you-get (WYSIWYG) design environment, including the accurate representation of complex layouts using the various Java layout managers. It also allows the Form Designer to run Java components and applets at design time so that the effects of run-time factors (an on-screen animation, for example) can be previewed in the layout design.

The Form Designer is fully integrated into the Visual Cafe development environment. As you design your forms the source code, properties, and project are dynamically kept in sync.

Note: While you're working in Visual Cafe you can have multiple Form Designer windows open at once, one per form.

When you're working on a form you can drag components from the Component Library, Component Palette, or Project window onto the Form Designer. The components appear as dotted-line rectangles that contain the component name. You can drag components in the Form Designer to arrange them, as well as perform other operations such as resizing or deleting them. See "Working with forms and components" on page 7-26 for details.

Here's an example of what a form looks like in the Form Designer:


Form Designer - JApplet1
jlabel
jlabel
jradioButton
jradioButton
jradioButton
jbutton
jbutton
jbutton
JApplet1
Dotted line outlines border of component
Scroll bars allow you to design larger forms with ease
The currently selected component or components have handles around their borders. Drag these squares to resize the component(s).

The Form Designer has been improved in version 3.0 of Visual Cafe. In earlier versions, the Form Designer window size was the size of your form. Now the Form Designer window size is not dependent on the size of your form; if your form is larger than the Form Designer window, you can use scroll bars to move around the form.

Dragging and dropping into the Form Designer

You can use standard Windows operating system techniques for dragging and dropping components into a form displayed in the Form Designer.

When you drag a component, a plus sign (+) appears over the cursor when a copy operation is being performed.

Visual Cafe does not allow inappropriate copies and moves, such as dropping a top-leval component into a container. If you see a circle with a slash through it while you're dragging a component, that means the operation is not allowed.

You can drag components to and from the following locations:

Drag from...	Into...	Result
Component Library or Palette	Form Designer	Copies the component to the new project (in other words, instantiates the component).
Project window	Form Designer in same project	Moves the item to the Form Designer. (Press CTRL and drag an item to copy it instead of move it.)
Project window	Form Designer in different project	Copies the file or component to the new project.
Form Designer	Form Designer in same project	Moves the component to a new location in the Form Designer. (Press CTRL and drag to copy a component instead of move it.)
Form Designer	Form Designer in different project	Copies the component to the second Form Designer

Form Designer shortcuts

The following table shows a list of keyboard shortcuts for working in the Form Designer:

Shortcut	Result
CTRL-click	Selects each component, in addition to previously selected components.

Shortcut	Result
Arrow keys	Moves selected components one pixel.
CTRL-Arrow keys	Enlarges selected components one pixel in the indicated direction.
TAB/SHIFT-TAB keys	Selects the next or previous component, respectively.

Displaying graphics in the Form Designer

When you have applets that have an image (.`gif` file) drawn by the `paint()` method, the image appears in browsers and in the AppletViewer. However, it may not appear in the Form Designer.

To have your images display at design time in the Form Designer, use the AWT-based `ImageViewer`, `ImageButton`, or `ImagePanel` component, or the Swing-based `ImageIcon` component.

Displaying non-visual components in the Form Designer

A **non-visual component** is a component that's not visible at run time, such as a timer, or displays in a different way in the Form Designer and at run time, such as a menu bar. It does not extend from the Java Component class. In the Form Designer, a non-visual component is represented by an icon. The icon is simply a visual indicator that the component is included, and does not affect the form layout.

Non-visual components give you access to prewritten functionality such as File Open dialog boxes—functionality that is platform-specific and shares a set of basic capabilities.

Note: When you add a non-visual component to a form, all non-visual components in the form display automatically.

It's useful to see non-visual components in the Form Designer, but displaying them can clutter a form and make it difficult to see visual components. Turning off the display of invisible components can make form development easier.

To toggle the display of invisible components:

◆ While the Form Designer is the active window, choose Invisibles from the Layout menu.

- ❖ If Invisibles is checked, non-visual components are displayed in your form.
- ❖ If Invisibles is not checked, non-visual components are not displayed in your form.

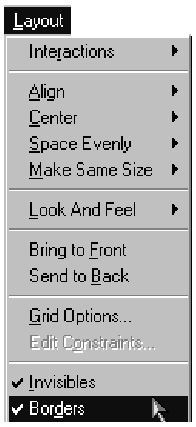
Enabling and disabling borders around components

In the Form Designer, you can choose to display dotted-line borders around components so that you can see boundaries of components. This is helpful in cases where you use non-visual components. If you wish, you can disable the borders.

To enable or disable borders around components:

1. With the Form Designer active, choose Borders from the Layout menu.

   This displays borders around components in the Form Designer.
2. Select Borders again to disable borders around components.



Using virtual fonts

Visual Cafe supports the Java virtual font classes, which you can use to ensure cross-platform compatibility. To use virtual fonts, specify `Serif, Sans-serif, Monospaced, Dialog,` or `DialogInput` in your programs, rather than a particular typeface such as Times or Helvetica. Note that virtual font classes are no longer mapped to non-Latin1 classes of fonts (non-Latin1 characters are for languages such as Korean, Japanese, and so forth).

To select a virtual font class:

1 Select a component that uses text.

2 In the Property List, select the Font Name property.

A list of font classes displays.

3 Choose a font class from the list of classes.

Your component's text changes to the appropriate font.

For information on default font properties, see the `font.properties` file in your Visual Cafe `\Java\Lib` folder.

Overview of designing a GUI

With Visual Cafe, you can add **graphical user interface** (**GUI**) elements to your applets or applications and define how those elements should interact with your applet or application.

Every time you drag and drop a visual component onto the form in the Form Designer or edit properties for a component using the Property List, Visual Cafe updates the source file for you.

This section lists the basic procedure for designing a GUI for your project. The concepts outlined here are described in detail later in the chapter.

To design your graphical user interface:

1 Add forms to your project.

You may already have some forms in your project if the project template added forms automatically when you created your project.

For details on this step, see "Adding a form to a project" on page 7-26.

2 Add components to your forms.

For details on this step, see "Adding components to a form" on page 7-27.

3 Arrange the components on the form.

For details on this step, see “Arranging components” on page 7-37.

4 Modify the components’ properties.

For details on using the Property List, see “Working with component properties” on page 7-34.

For details on using customizers, see “Using a customizer to configure a component on a form” on page 10-26.

5 Create component interactions.

For more information, see Chapter 9, “Working with Events and Interactions.”

Working with forms and components

In this section you’ll learn how to use the Form Designer to create forms and add them to your projects. Step-by-step instuctions for adding a form to a project and adding components to a form are provided.

Accessing the Form Designer

To work on a form, you need to access the Form Designer. When you open a new Visual Cafe project, a blank Form Designer is displayed. If you’ve already created a form and want to work on it, you can access the Form Designer in either of two ways.

To access the Form Designer:

◆ Double-click the form in the Objects view of the Project window.

or

◆ Right-click the form in the Objects view of the Project window and choose Edit form from the pop-up menu.

Adding a form to a project

You can add a form to a project by using the Insert menu or by using drag-and-drop.

To add a form using the Insert menu:

1. While the project is selected, choose Form from the Insert menu.
2. Select a form template from the Insert Form dialog box.
3. Click OK.

   The new form is added to the project and the Form Designer opens.

To add a form using drag-and-drop:

◆ Drag a form from any of the following areas into the Project window:

- ❖ The Component Palette
- ❖ The Component Library
- ❖ The same Project window (press CTRL and drag to copy the form)
- ❖ A different Project window

The new form is added to the project and the Form Designer opens.

In addition, any source file that contains a subclass of `Applet`, `Dialog`, `Frame`, or `Window` will add a top-level container to the Objects view of the Project window, as well as a form to the project. Unless the Enable RAD for new files option (in the Project Options dialog box) is unchecked, Visual Cafe parses the file to extract information about the component classes defined in the file.

Note: If the parse fails, you will see a file in the Packages and Files view, but not an object in the Objects view of the Project window. You probably need to correct the Java code to get the file to parse. See “Adding custom code to a source file” on page 4-48 for more information.

Adding components to a form

After you add a form to a project, you can add components to it and arrange the components in the Form Designer. For more information about adding a component to a project, see “Adding a component to a project” on page 3-49.

Not all components are appropriate for all types of forms. The following table describes the relationship between form types and their valid components.

Form type	Valid components
Frame	Menus and all components
Applet	All components except menus and forms

When a component is selected and you move the cursor over a handle (the small black box at the corner of the rectangle that represents a component), the cursor changes to a two-way arrow, which means you can hold down the mouse button and drag to resize the component.

When a component is selected and you move the cursor over it, the cursor changes to a four-way arrow, which indicates that you can click and drag the component to another location.

To select multiple components, SHIFT-click them or drag the cursor over them.

Note: The placement and size of a component might be restricted if you are using a layout manager.

There are several considerations to keep in mind if you want to overlap components. See "Overlapping components in applets" on page 7-32 for more information.

To add a component to a form:

1. Add the component by using one of these methods:
   - ❖ Use the Insert menu
   - ❖ Drag the component from the Component Palette to the Project window
   - ❖ Drag the component from the Component Palette to the Form Designer
   - ❖ Drag the component from the Component Library to the Form Designer
   - ❖ Drag the component from the Component Library to the Project window
2. Size the component if no layout manager is in use.

3. While the component is selected, type a name for the component in the Property List. The name should not contain spaces.
4. Change component properties in the Property List as needed.
5. From the File menu, choose Save to save your changes.

Copying components

You can copy a component by dragging it from the following areas into the Project window, Form Designer, or Menu Designer:

- The Component Palette or Component Library
- A Project window or Form Designer in the same project (press CTRL and drag to copy the component)
- A Project window or Form Designer in a different project

Note: Only menu components can be placed into the Menu Designer. See “Creating AWT-based menus” on page 7-48 for information about the Menu Designer.

You can copy form components and menu bars to another form, within the same form, or to another project by:

- dragging and dropping the component to a different form
- copying and pasting the component with menu commands
- copying and pasting the component with Toolbar icons
- right-clicking the component and using the Copy and Paste pop-up menu commands
- selecting the component and using the copy and paste keyboard commands

When you copy a component, keep the following concerns in mind:

- The object’s bound events are not moved to the new location.
- You must manually move any of your own code to the new form.
- Forms and components must have unique names. If necessary, an object is renamed when pasted.

- If you copy and paste a top-level container, the corresponding Java file is duplicated and placed in the target project directory.
- If you copy and paste a component in a container, Java code that is needed to create and initialize that component is generated in the Java file of the top-level container. Only component code that is automatically generated by Visual Cafe is placed in the Java file. This does not include custom code and interactions.
- Visual Cafe allows only appropriate copies; for example, a component that is not a container cannot be copied to the top level. While you're dragging, a circle with a slash through it indicates that the operation is not allowed.

You can copy components between the Project window (Objects view), Form Designer, and Menu Designer as needed.

To copy and paste a component:

1. Open the project(s) you want to use.
2. Click the Objects tab in the Project window, or open the Form Designer or Menu Designer, then select the component you want to copy.
3. Choose Copy from the Edit menu, or right-click and choose Copy, or click the Toolbar's Copy button.
4. If you want to paste the component within another container, select that container.
5. While the target Project window (Objects view), Form Designer, or Menu Designer is active, choose Paste from the Edit menu. Alternatively, you can right-click and choose Paste, or click the Toolbar's Paste button.

   The component appears in the Project window. If necessary, the component is renamed to prevent name conflicts.

To copy a component by dragging it:

- Do either of the following:
  - To copy and drag a component within the same project, press CTRL and drag the component to the location where you want it.
  - To copy and drag a component to a different project, drag the component to the location where you want it.

Deleting components

You can easily delete a component from the Objects view of the Project window or from the Form Designer.

If you delete a top-level container in the Objects view, the corresponding `.java` file is deleted from the project and the other views. The file is not deleted from the folder on your hard disk; you must manually delete it.

Note: If you want to delete a file, you should first delete it from the project and then delete it using the Windows operating system commands. Do not delete a file using the Windows operating system commands before first deleting it from the project.

If you delete a component from a container, any code that was automatically generated by Visual Cafe is deleted. You'll be prompted to remove interactions and bound events associated with the component. If you delete a component's code from a Java file, the component is removed from the Project window after you click outside the Source window or save the Java file.

If you delete a component that's associated with an event or interaction, Visual Cafe asks you if you want to remove event bindings, keep them, or cancel the deletion. If the event bindings and interactions are removed, your code will compile. For more information about events and interactions, see Chapter 9, "Working with Events and Interactions."

Note: To avoid deleting the wrong code or not deleting enough code, we recommend that you delete components from the Project window or the Form Designer instead of from the source code. Then delete any custom code or interactions from the source code. For information on deleting interactions, see "Deleting an interaction" on page 9-17.

To delete a component:

◆ Do either of the following:
  - ❖ Select the component in either the Form Designer or the Project window and press DELETE.
  - ❖ Select the component and choose Cut from the Edit menu, right-click on the component and choose Cut from the pop-up menu.

Renaming a component

You can rename the components listed in the Objects view of the Project window. If you rename a top-level container, the corresponding Java file name changes. If you rename a component in a container, the Java code of the top-level container changes for that component. Visual Cafe changes the component name in the entire Java source file, even in custom code.

You can rename components from either the Project window or the Property List.

To rename a component from the Project window:

Do any of the following:

1. Click the Objects tab in the Project window, and select the component name.
2. Press TAB and retype the name in the field.
3. Press ENTER or click somewhere else when you're finished.

   or

1. Click the Objects tab in the Project window.
2. Slowly double-click the component name and then type the new name in the field.
3. Press ENTER or click somewhere else when you're finished.

   or

1. Select the component in the Objects view of the Project window.
2. Press F2.
3. Type the new name in the field.

To rename a component from the Property List:

◆ Make your project active, then change the `Name` property of the component in the Property List. See "About the Property List" on page 7-34 for more information.

Overlapping components in applets

Different Web browsers use a different "z-order" (see "Determining component z-order (display order)" on page 8-21 for information) when determining how components overlap in an applet. The Project window displays the order, and the browser can read it from top to bottom or from bottom to top. For example, if an `InvisibleHTMLLink` is on top of an image, you need to make sure it ends up on top for users to be able to click it. To ensure compatibility with different browsers, it's a good idea to "sandwich" the `InvisibleHTMLLink`s on top and beneath lightweight components they overlap. Use Send to Back or Send to Front from the Layout menu to do so.

You need to pay attention to whether heavyweight and lightweight components overlap, because heavyweight components always display over lightweight components. See "About lightweight and heavyweight components" on page 7-6 for more information.

Tabbing between fields on a form

You can allow users to tab between fields on a form by making use of the default tab order provided with JDK 1.1. The default order is the order in which the components are added to the form.

You can also establish a tab order by placing the fields in a `KeyPressManagerPanel` container.

In both cases, the tab order of the components (those contained by the panel if you're using one) is listed in the Objects view of the Project window. To change the tab order, change the order of the components in the Project window list. You can also reorder the components visually in the Form Designer.

In your application, the user presses the TAB key to move to the next component you've specified in your tab order, such as a text-entry field in an online order form. Pressing SHIFT-TAB returns to the previous component in the tab order.

Adding a dialog box to a form

In the Component Library, dialog boxes are located in the Forms group. Some of the dialog boxes are top-level components and some must be contained by a top-level component. See the *Components Reference* in the Online Help for more information.

To add a dialog box:

1. Drag a `Dialog` component from the Component Palette or Component Library onto the Form Designer.

   For details on this step, see “Adding components to a form” on page 7-27 or “Adding a component to a project” on page 3-49.

2. Connect the dialog to a trigger component by using the Interaction Wizard.

   For details on this step, see “Starting an interaction” on page 9-6.

Working with component properties

Each component has a set of properties that define its appearance and behavior. Visual Cafe provides a Property List from which you can directly modify these properties. When you change a property, the source code and Form Designer are immediately updated. If you change a property for an object in the Component Library, you’ll see the changes the next time you create a new object.

About the Property List

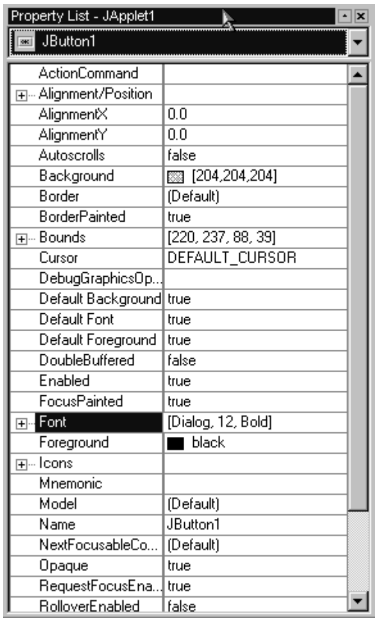
The **Property List** displays properties and values for the currently selected component or components. You can select a component from the Form Designer, Project window, Component Library, or the Property List’s pull-down menu.

Note: When multiple components are selected, only their common properties are shown and editable.

To display the Property List:

◆ Choose Property List from the View menu.

Properties with values that can be expanded are marked with a plus sign (+); for example, the `Font` property. Properties that are defined using a custom editor are marked with an ellipsis button (…).



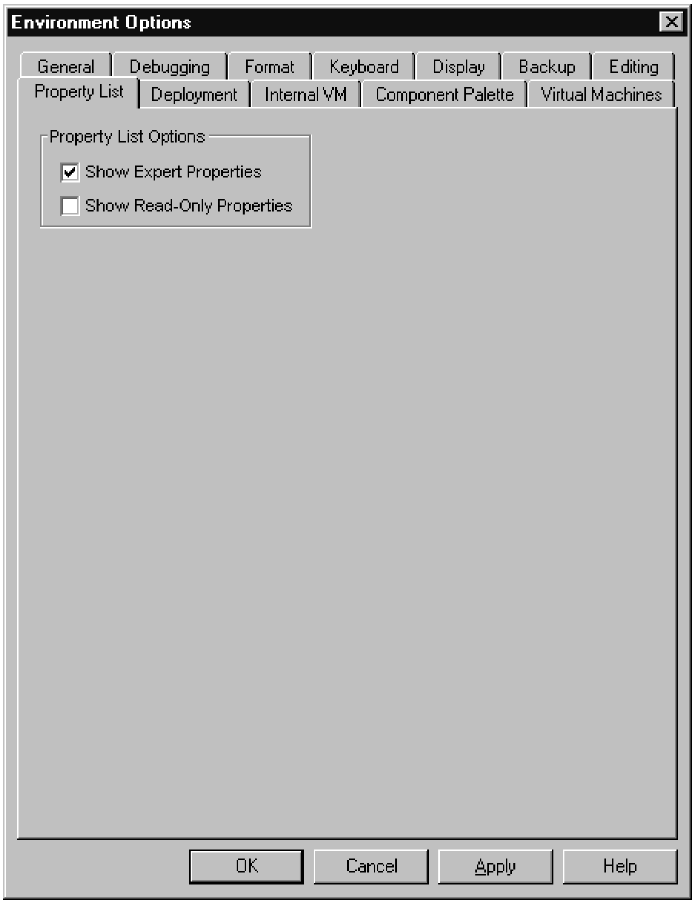
ActionCommand	
⊞ Alignment/Position	
AlignmentX	0.0
AlignmentY	0.0
Autoscrolls	false
Background	▨ [204,204,204]
Border	(Default)
BorderPainted	true
⊞ Bounds	[220, 237, 88, 39]
Cursor	DEFAULT_CURSOR
DebugGraphicsOp...	
Default Background	true
Default Font	true
Default Foreground	true
DoubleBuffered	false
Enabled	true
FocusPainted	true
⊞ Font	[Dialog, 12, Bold]
Foreground	■ black
⊞ Icons	
Mnemonic	
Model	(Default)
Name	JButton1
NextFocusableCo...	(Default)
Opaque	true
RequestFocusEna...	true
RolloverEnabled	false

Components may also have a **customizer,** a tool that helps guide Bean developers through the process of changing a component. A customizer can configure more than one property at a time. You can access a customizer from the Property List. See "Using a customizer to configure a component on a form" on page 10-26 for more information.

Some components can have what are called expert properties. The programmer who coded a particular component may have decided that its properties are **expert properties,** specialized properties that most users won't normally need. (For example, in the Visual Cafe environment, `AlignmentX` and `AlignmentY` are expert properties.) Expert properties are displayed in the Property List only when you tell Visual Cafe to display them.

To display expert properties:

- ◆ Click the Property List tab in the Environment Options dialog box, as shown here:



Property List Options

- [✓] Show Expert Properties
- [ ] Show Read-Only Properties

Modifying component properties

You can modify properties for one or more components at a time by using the Property List.

Tip: Remember that if you change the properties of a component in the Component Library, the change now appears every time you add that component to a project. Projects that already contained the component before the change are not affected.

To modify component properties:

1. To display the Property List, choose Property List from the View menu.
2. Select a component in the Form Designer, Project window, Component Library, or the Property List's pull-down menu. To select multiple components, CTRL-click in the Form Designer, Project window, or Component Library, or SHIFT-click or drag in the Form Designer.

   When multiple components are selected, only their common properties are shown and editable. In the Property List, you see the heading Multiple Selection.
3. To edit a property, click the right column, double-click the left column, or use TAB in the Property List.

   The right column displays a list of valid values or makes the text string editable.

   Properties with multiple values (for example, the `Font` property) are marked with a plus sign (+). Click the plus sign (+) to expand the list.

   Properties that are defined using a custom editor are marked with the ellipsis button (…). Click the … button to display the custom editor.
4. Press ENTER or click somewhere else to make the change.

   The change is applied to all selected components.

   Note: Press ESC to cancel an edit and return the property to its previous value.

Arranging components

The Java language provides several **layout managers** (also called **layouts**) that help you arrange components inside a container so they'll maintain a uniform appearance across multiple platforms and Web browsers, and at varied screen sizes and resolutions.

The available layouts for AWT components are:

- `BorderLayout`
- `CardLayout`
- `FlowLayout`
- `GridLayout`
- `GridBagLayout`

These layouts are described in the following sections.

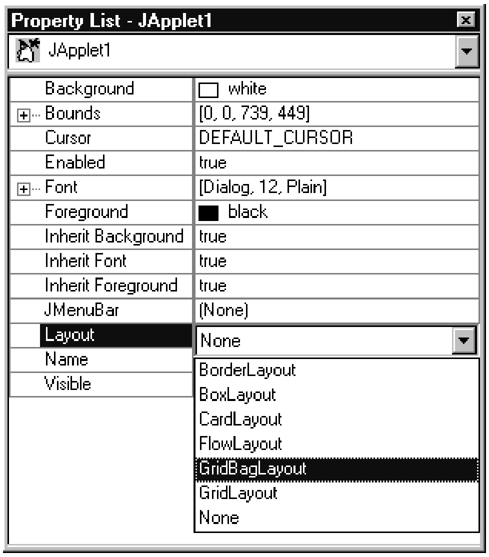
You can also use a layout called `BoxLayout` to arrange your AWT components, even though this layout is part of the JFC Swing layout manager set. For more information on `BoxLayout` and other Swing layout managers, see "Choosing a layout manager for a container" on page 8-14.

Different AWT containers have different default layout managers. For example, the default layout manager for `Panel` is `FlowLayout`, while the default for `Frame` is `BorderLayout`. Visual Cafe, however, defaults the layout property of all other containers to None. When there is no layout manager, components in a container are positioned and sized exactly as you specify.

The way AWT components appear on the screen is determined by the order in which components are added to the panel that contains them, and the layout manager the panel is using to display them on the screen. The layout manager determines which components within that panel will be displayed.

Visual Cafe supports all standard Java layout managers. You can add a layout to a form (or panel) by setting the form's layout property. When you change the property, the components inside the layout are immediately arranged based on their creation order and the specified layout. You can rearrange components in the layout by dragging and dropping to the desired location.

You can group components in panels on your form and use different layout managers to suit the components contained by the panel.



Property List - JApplet1

JApplet1

Background	☐ white
Bounds	[0, 0, 739, 449]
Cursor	DEFAULT_CURSOR
Enabled	true
Font	[Dialog, 12, Plain]
Foreground	■ black
Inherit Background	true
Inherit Font	true
Inherit Foreground	true
JMenuBar	(None)
Layout	None
Name	
Visible	

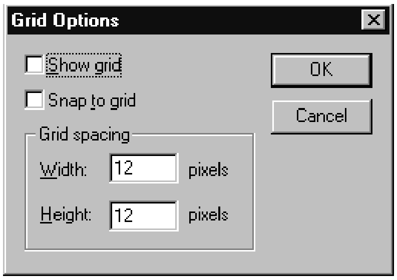
Manipulating the Form Designer grid

While you're designing a form, the Form Designer grid is useful in laying out the components in the form.

To open the Grid Options dialog box:

◆ While a Form Designer is the active window, choose Grid Options from the Layout menu.

The Grid Options dialog box appears:



Grid Options

☐ Show grid

☐ Snap to grid

Grid spacing

Width: 12 pixels

Height: 12 pixels

To enable or disable grid display:

◆ In the Grid Options dialog box, select or deselect Show Grid.

To enable or disable the snap-to-grid capability:

◆ In the Grid Options dialog box, select or deselect Snap to Grid.

This option automatically aligns components to the grid when you're moving, sizing, and creating them.

To set grid spacing:

◆ In the Grid Options dialog box, type the amount of space between grid points, both horizontally and vertically.

Choosing a layout manager

You specify a layout for a container by setting the `Layout` property in the Property List. You can arrange components in a layout by dragging and dropping to a new location in the Form Designer, changing the component order in the Project window, or changing properties for a component or the container — depending on the layout manager you're using.

You can also specify that no layout manager be used by a container by selecting None as the value of the `Layout` property (see "Arranging components without a layout manager" on page 7-41 for details). The advantage of not using a layout manager is that you have precise control over component placement. The disadvantage is that some components display differently on different platforms. For example, any component that displays text (except when text is a part of a saved graphic) can be expected to appear differently on different platforms.

To choose a layout manager:

1. Open the Property List for a form or panel.
2. In the Property List, select a layout for the `Layout` property.

   The Form Designer immediately reflects the layout by rearranging the components based on the order in which they appear in the Project window.
3. Rearrange the components in the Form Designer, if needed.

You can arrange components by:

- Arranging components without a layout manager
- Arranging components in BorderLayout
- Arranging components in CardLayout
- Arranging components in FlowLayout
- Arranging components in GridLayout
- Arranging components in GridBagLayout
- Manipulating the Form Designer grid

These options are discussed in the following sections.

Arranging components without a layout manager

If you wish, you can specify that no layout manager be used by a container by using None as the value of the `Layout` property (None is the default layout). That way, you can place components exactly where you want them in a form, positioning them at precise pixel positions.

To arrange components in a layout manager of None:

- Use any of the following techniques:
  - Drag components on the Form Designer to position them.
  - Explicitly set *x* and *y* coordinates and the `Bounds` property in the Property List.
  - Arrange objects with the Align, Center, Space Evenly, Make Same Size, Bring to Front, and Send to Back commands in the Layout menu.
  - Use the Form Designer grid, which you can set by choosing Grid Options from the Layout menu. See “Manipulating the Form Designer grid” on page 7-39 for more information.
  - Move a component pixel by pixel by selecting the component on the Form Designer and pressing the Right, Left, Up, and Down Arrow keys, as needed.

You should test your layout by running your program on different operating systems and screens of different sizes and resolutions, as applicable.

Tip: For maximum portability, it's sometimes best not to overlap components in your layout.

Arranging components in BorderLayout

Use `BorderLayout` to arrange components in a center, north, south, east, and west orientation. It positions components based on their preferred sizes and the constraints of the container size.

To arrange components in BorderLayout:

1. Choose BorderLayout for the `Layout` property of a form or panel.
2. If components are already on the form or panel, rearrange the components as needed. Set the `Placement` property for each component you want to position.

   Note: In `BorderLayout`, no components should have the same Placement value.

3. In the Property List, choose the form or panel component, then set the `Horizontal Gap` and `Vertical Gap` properties to adjust the layout.
4. For each component you want to add, add the new component then set its `Placement` property to place it on the form or panel.

   When you first add a component, its `Position` property is blank (which is the same as `Center`).

5. Test your layout by running it at different form sizes and screen resolutions. To help test your layout, you can resize the form when you run it from Visual Cafe.

Arranging components in CardLayout

Use `CardLayout` to arrange components on several cards. You might want to simulate a stack of index cards, for example. Only one card is visible at a time, which allows you to flip through the cards.

Note: To set the flipping of cards at run time, you can either create an interaction, or enter code directly in the source. To use an interaction, create an interaction that as its action chooses a new card. For information about writing the code, see "Programming the flipping of cards in CardLayout" on page 7-43.

A component will be sized to take up an entire card; if you want multiple components on a card, place components on panels. That way, the panel will take up the entire card.

To arrange components in CardLayout:

1. Choose CardLayout for the `Layout` property of a form or panel.

   If components are already on the form or panel, each component directly subordinate to the form or panel becomes a separate card. The cards are placed in the order in which they appear in the Project window.
2. In the Property List, choose the form or panel component, then set the `Horizontal Gap` and `Vertical Gap` properties to adjust the layout.
3. If components are already on the form or panel, rearrange the components in the Form Designer or Project window, as needed:
   - ❖ To change the card order, change the component order in the Project window.
   - ❖ To flip between cards in the Form Designer, right-click, then choose Previous Card or Next Card.
4. Add components as needed and rearrange them.
5. Test your layout by running it at different form sizes and screen resolutions. To help test your layout, you can resize the form when you run it from Visual Cafe.

Note: Changing the current card by selecting it in the Property List or by changing the `Selected Card` property of the container only works if the Card Name of each card is unique.

Programming the flipping of cards in CardLayout

You can enter code to implement the flipping of cards. Here are examples of the code you need to write:

To go to a certain card:

```
((CardLayout)container.getLayout()).show(container,
 "cardname");
```

To go to the next card:

```
((CardLayout)container.getLayout()).next(container);
```

To go to the previous card:

```
((CardLayout)container.getLayout()).previous(container);
```

Where `container` is the name of the container that you're using, and `cardname` is the name of the card that you want to show.

Arranging components in FlowLayout

Use `FlowLayout` to arrange components in rows from left to right. You can specify center, left, or right alignment, as well as the horizontal and center gaps between components.

To arrange components in FlowLayout:

1. Choose FlowLayout for the `Layout` property of a form or panel.

   The Form Designer immediately reflects the layout by rearranging the components based on the order in which they appear in the Project window.
2. In the Property List, choose the form or panel component, then set the `Alignment`, `Horizontal Gap`, and `Vertical Gap` properties to adjust the layout.
3. If components are already on the form or panel, rearrange the components in the Form Designer or Project window, as needed.
4. Add components as needed and rearrange them.
5. Test your layout by running it at different form sizes and screen resolutions. To help test your layout, you can resize the form when you run it from Visual Cafe.

Arranging components in GridLayout

Use GridLayout to arrange components in definable rows and columns. This layout is similar to `FlowLayout`, except that each component is in an area of equal size. You can specify the number of rows and columns, as well as the horizontal and center gaps between components.

To arrange components in GridLayout:

1. Choose GridLayout for the `Layout` property of a form or panel.

   The Form Designer immediately reflects the layout by rearranging the components based on the order in which they appear in the Project window.
2. In the Property List, choose the form or panel component, then set the `Rows`, `Columns`, `Horizontal Gap`, and `Vertical Gap` properties to adjust the layout.

   If you set either `Rows` or `Columns` to zero, `GridLayout` computes the other value for you. If both `Rows` and `Columns` are non-zero, the `Rows` value is used.
3. If components are already on the form or panel, rearrange the components in the Form Designer or Project window, as needed.
4. Add components as needed and rearrange them.
5. Test your layout by running it at different form sizes and screen resolutions. To help test your layout, you can resize the form when you run it from Visual Cafe.

Arranging components in GridBagLayout

Use `GridBagLayout` to arrange components by size and position. Like `GridLayout`, `GridBagLayout` treats the form or panel as a grid of cells. Unlike `GridLayout`, however, a component can occupy more than one cell in a grid bag layout.

The `GridBagLayout` is the most powerful way of managing an AWT layout, but it can be very complicated. New to Visual Cafe is the GridBag Constraints Editor, which you use to arrange components in a `GridBagLayout`.

When you use a grid bag layout, you're actually using two classes: `GridBagLayout`, which provides the overall layout manager, and `GridBagConstraints`, which defines the properties of each component in the grid: its dimensions, placement, alignment, and so on. The grid bag, the constraints, and each component come together to create the overall layout.

Grid Bag Contraints are a set of properties that determine how a visual component will grow, shrink, or reposition itself, based on how its container is resized. All grid-bag-constrained components have a separate `GridBagConstraints` property value. This means that unforeseen size and boundary conflicts between components can occur when the container is set to certain sizes and dimensions. You must manually test your component layout design to determine that all components within a container behave appropriately. See the *Components Reference* in the Online Help for more information.

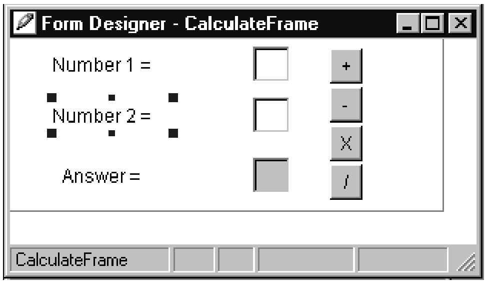
Note: Visual Cafe 3.0 generates different code for `GridBagConstraints`, as compared to previous versions. Code that would take up around eight lines in earlier versions now takes up one.

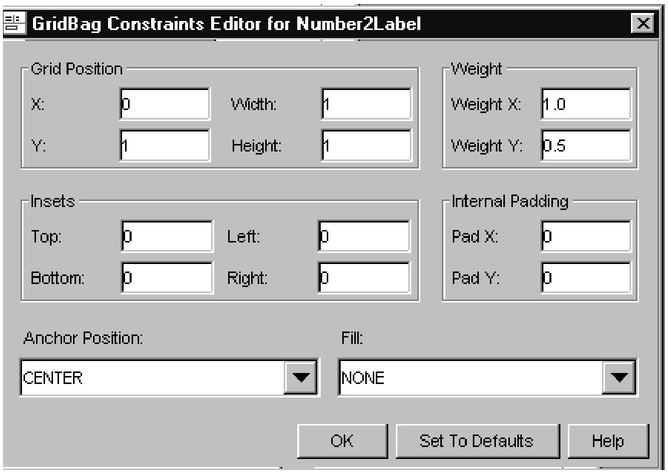
To arrange components in GridBagLayout by using the GridBag Constraints Editor:

1. Open the Property List for a form or panel.
2. Choose GridBagLayout for the `Layout` property of a form or panel.

   The Form Designer immediately reflects the layout by rearranging the components based on the order in which they appear in the Project window.
3. In the Objects view of the Project window or in the Form Designer, right-click on a component contained by the form or panel, then choose Edit Constraints. Or, while the Form Designer is active, choose Edit Constraints from the Layout menu.

The GridBag Constraints Editor for that component appears, as shown here:




GridBag Constraints Editor for Number2Label

Grid Position

X: 0
Width: 1
Y: 1
Height: 1

Weight

Weight X: 1.0
Weight Y: 0.5

Insets

Top: 0
Left: 0
Bottom: 0
Right: 0

Internal Padding

Pad X: 0
Pad Y: 0

Anchor Position: CENTER

Fill: NONE

Only one GridBag Constraints Editor appears at a time.

4 Change `GridBagConstraints` properties as needed. You can think of these properties as suggestions that `GridBagLayout` uses.

The Form Designer display adjusts to each new setting you enter.

5 Use the GridBag Constraints Editor or the Property List to set the `GridBagConstraints` properties to adjust each component's place in the layout. You can rearrange and add components as needed.

Tip: In the Property List, you can press F1 on a `GridBagConstraints` property to get a description of it. Insets specify how much space to leave between the borders of a component and its display area.

6. Test your layout by running it at different form sizes and screen resolutions. To help test your layout, you can resize the form when you run it from Visual Cafe.
7. To close the GridBag Constraints Editor, click OK.

Creating AWT-based menus

You can add menu bars to frames and dialog boxes, which inherit from `MenuContainer`; AWT-based applets do not support menu bars.

Note: The `Frame` component in the AWT Application project template already has a menu bar, which you can modify and enhance.

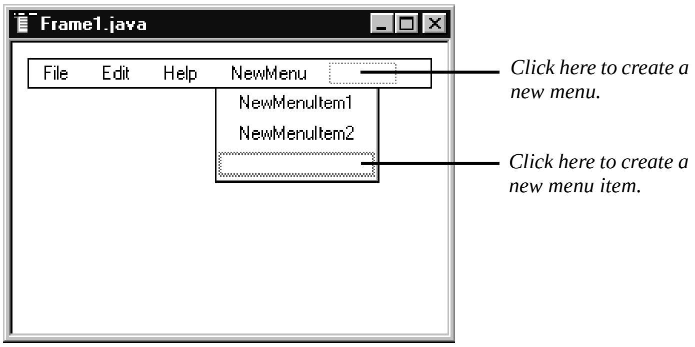
The **Menu Designer** makes it easy to create a menu bar by letting you edit a visual representation of the menu bar. You can move items in the Menu Designer to visually change the menu structure.

To open the Menu Designer:

◆ Double-click a `MenuBar` component in the Project window or Form Designer.

or

◆ In the Project window, select the `MenuBar` component, then choose Edit MenuBar from the Object menu, or right-click and choose Edit MenuBar.


Frame1.java
File
Edit
Help
NewMenu
NewMenuItem1
NewMenuItem2
Click here to create a new menu.
Click here to create a new menu item.

Overview of the menu-design process

Here's an overview of the steps you'll follow when designing a menu. These steps are described in detail in the following sections.

To design your menu bar and menus:

1. Add a menu bar to a frame or dialog.

   (If you used the AWT Application project template, you already have a menu bar in the frame that it created for you.)

   For details on this step, see "Adding a menu bar to an AWT-based frame or dialog box" on page 7-50.
2. Add menus to your menu bar.

   For details on this step, see "Adding menus to an AWT-based menu bar" on page 7-51.
3. Add menu items to your menus.

   For details on this step, see "Adding menu items to AWT-based menus" on page 7-51.

   You can also add menu items that can have submenus.

   For information on submenus, see "Adding submenus to menu items" on page 7-52.
4. Edit the menu structure as needed.

   For details on this step, see "Editing a menu structure" on page 7-53.

5 Associate command keys with menu items, as needed.

For details on this step, see “Associating command keys and menu items” on page 7-54.

6 Add interactions between menu items and other components in your project, as needed. For example, specifying that a menu item should open a dialog within the same project.

To quickly open the Interaction Wizard, right-click and choose Interaction Wizard from the pop-up menu.

For details on this step, see “Starting an interaction” on page 9-6.

7 Bind custom code to menu items, as needed.

For details on this step, see “Binding code to a menu item” on page 7-55.

Tip: You can save time by copying menu bars and menus from other projects. See “Copying components” on page 7-29 for more information. You can also add commonly used menu bars and menus to the Component Library and Palette. See “Customizing the Component Palette” on page 7-13 for more information.

Adding a menu bar to an AWT-based frame or dialog box

You add menus to dialogs and frames in the Form Designer, then edit them in the Menu Designer. (Remember that AWT-based applets do not accept menu bars.)

You can add the `MenuBar` component to a frame or dialog box in a variety of ways.

To add a menu bar to a frame or dialog box:

◆ Do one of the following:
- ❖ Drag a `MenuBar` component from the Component Library or Palette into the Project window or Form Designer.
- ❖ While the Project window or Form Designer is active, choose Component from the Insert menu.
- ❖ Right-click the frame or dialog box in the Project window and choose Insert Component.

- ❖ Copy and paste a menu bar within the same project or between projects. See "Copying components" on page 7-29 for details.

Adding menus to an AWT-based menu bar

You can add a `Menu` component to a `MenuBar` component.

Note: If you're using Swing components you have more options for menu items. For more information, see "Working with Swing menus" on page 8-22.

To add a menu to a menu bar:

1. Do one of the following:
   - ❖ Drag a `Menu` component from the Component Library or Palette into its position in the Project window or Menu Designer.
   - ❖ While the menu bar is selected in the Menu Designer, right-click and choose Insert menu.
   - ❖ Right-click the menu bar in the Project window and choose Insert Component.
   - ❖ While the menu bar is selected in the Project window or Menu Designer, choose Component from the Insert menu.
   - ❖ Copy and paste the menu component within the same project or between projects. See "Copying components" on page 7-29 for details.
2. In the Property List, set the menu properties, including the menu name, in the `Label` property.

   The `HelpMenu` property lets you integrate the menu item into an existing Help menu that's part of the operating system, for example.

Adding menu items to AWT-based menus

You can add a `MenuItem` or `CheckboxMenuItem` component to a `Menu` component.

Note: If you're using Swing components, you have more options for menu items. See "Working with Swing menus" on page 8-22.

To add a menu item to a menu:

1. Do one of the following:
   - ❖ Drag a component from the Component Library or Palette into its position in the Project window or Menu Designer.
   - ❖ While a menu item is selected in the Menu Designer, right-click and choose Insert Menu Item or Insert Checkbox Menu Item.
   - ❖ Right-click the menu in the Project window and choose Insert Component.
   - ❖ While a menu is selected in the Project window or Menu Designer, choose Component from the Insert menu.
   - ❖ Copy and paste it within the same project or between projects. See "Copying components" on page 7-29 for details.
2. In the Property List, set the menu item properties, including the menu item name in the `Label` property.

   Tip: To add more menu items while in the Menu Designer, you can now select the bottom menu item in the list and press ENTER.

Adding submenus to menu items

You can add a `MenuItem` or `CheckboxMenuItem` component to a `Menu` component that is within another `Menu` component.

To add a submenu to a menu item:

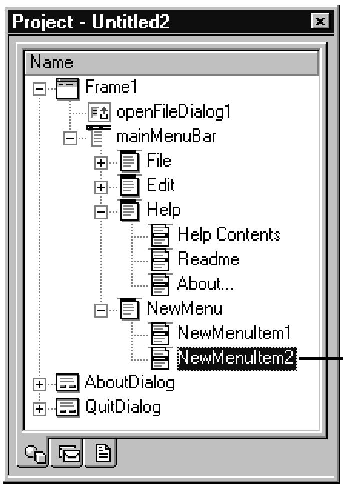
1. In the Menu Designer, right-click a menu item and choose Insert Menu.

   A menu that can have submenus appears.
2. While the new menu is selected, set the menu properties in the Property List, including the name of the menu in the Label field.
3. Click the submenu box, then add a menu item. For more information, see "Adding menu items to AWT-based menus" on page 7-51.

Tip: To add more submenu items, you can now select the bottom submenu item in the list and press ENTER.

Editing a menu structure

You can change a menu's structure as needed.



Click and drag a menu item in the Project window (Objects view) to change the order of the items in a menu.

To move items in the menu structure:

◆ Move items in the Menu Designer or Project window to visually change the order of the items in the menu.

To delete a menu or menu item:

◆ Select the component in the Menu Designer or Project window, then press DELETE or choose Delete from the Edit menu.

Note: If you delete a component from a container, you must manually delete any custom code, interactions, or event bindings that involve that component.

Editing menu bars and menus

You can edit menus by using the Menu Designer and Property List. Each menu is a subcomponent of the menu bar.

To edit a menu or menu bar:

1. Add a menu bar to the form, or if the menu bar already exists, do either of the following:
   - ❖ In the Form Designer, double-click on the menu object.
   - ❖ In the Project window, double-click on the menu bar icon.

   Note the menu placeholder in the menu bar window.
2. Open the Property List by choosing Property List from the View menu.
3. Do either of the following:
   - ❖ Select the `Label` property and enter the menu caption.
   - ❖ Highlight the menu placeholder and start typing. The text is added to the `Label` property.
4. To add more menu items, do either of the following:
   - ❖ Press ENTER to move down to the next menu item.
   - ❖ Choose Insert Menu Item from the pop-up menu to insert a menu item before the selected menu item.
5. Right-click on the menu item and choose Create Submenu from the pop-up menu.
6. Bind code to the appropriate menu items.
7. Define any interactions by right-clicking the menu item and choosing Add Interaction from the pop-up menu. For more information, see Chapter 9, “Working with Events and Interactions.”

Associating command keys and menu items

You can quickly add command keys to menu items. A **command key** allows the user to access a program feature by pressing a combination of keys instead of clicking onscreen items with a mouse. For example, the user might press CTRL-P instead of choosing Print from a menu.

To associate a command key with a menu item:

1. Select a menu item in the Project window or Menu Designer.
2. In the Property List, expand the `Menu Shortcut` property and specify the command key(s) in the Key Code and Use Shift Key fields.

   The Key Code field lets you specify what keys you want to use; for example, `VK_P` selects `CTRL-P`. In the Menu Designer and Project window, this key sequence displays as "`CTRL-Kanji`."

   If you want the SHIFT key to be part of the command key sequence, choose true; otherwise, choose false.
3. Verify your command keys by running your Java program:
   - ❖ Choose Execute from the Project menu to run the project with no debugging.
   - ❖ Choose Run in Debugger from the Project menu to run the project and have access to all debugging functionality.

Note: You can compile and run a project at any time during its development cycle. Visual Cafe automatically saves the files in the project before running.

Binding code to a menu item

You can bind code to a menu item just as you can bind it to a component. Menu items respond to one event: `ActionEvent`, which occurs when the user selects a menu item.

To bind code to a menu item:

1. Open the menu bar in the Menu Designer or Project window.
2. Select the menu item.
3. Right-click and choose Edit Source from the pop-up menu.
4. In the Source window, select the Action event from the Event/ Method drop-down list.
5. Add the appropriate Java code to the event handler.

CHAPTER 8

Working with JFC/Swing Components

Visual Cafe includes a group of components called the Swing components. Swing components, which are part of the Java Foundation Classes (JFC) have the following advantages over the older AWT components:

- They are "lightweight" components that require fewer system resources.
- You can control the look and feel (appearance and behavior) of Swing components.
- A number of the components allow you to place an image icon on them.
- The Swing set of components includes types of components that are not included in AWT, such as a scrolling pane, a table, and a tree.

This chapter discusses how to use the Swing components.

About Swing

Swing is the name given to the new set of Java visual components, which you can use to create user interface elements such as buttons, tables, lists, text fields, windows, and so on.

The Swing components plus some accessibility features make up the **Java Foundation Classes** (**JFC**). Since JFC currently contains little aside from Swing, JFC is often used as a synonym for Swing. JFC will contain other items in the future.

Earlier versions of Java introduced the AWT (Abstract Windowing Toolkit) components. The AWT components use native-code components to draw on local systems. These native-code components are called **peer components**. AWT uses peer components so that components in a Java program look the same as components in a native application. The idea was that you could write one program that would look the same when run on different operating systems; when you ran it on a Windows machine it would look like a Windows application, when you ran it on a Macintosh it would look like a MacOS application, and when you ran it on a UNIX platform it would look like a Motif/CDE application. Although there were advantages to that approach, it presented some problems:

- Since the peer components were not written in Java, different platforms had different bugs in their components.
- You had limited control over the final appearance of your application. It can be difficult to design an application that looks right and works correctly with components that have appearance and behavior (look and feel) that you can't control.

Swing is designed to fix these problems. Swing components are 100% pure Java, and have minimal dependence on native C code. That means you can easily subclass Swing components to create your own components. In addition, the fact that Swing components are entirely written in Java means that the Java code determines what they look like, so that you can control their final appearance.

To extend that control, Swing components have what is called "pluggable look and feel." You can determine the appearance (look) and behavior (feel) of the components in your program at design time or at run time. If you want your application to behave like a Macintosh application on a Macintosh computer and a Windows application on a Windows computer, you can make that happen. On the other hand, if you want your application to behave in the same way on every platform, you can do that, too. Swing comes with several built-in look-and-feels, and you can create your own so that components have a unique style that you determine.

The need for native-code peer components leads to AWT components being called *heavyweight components.* Swing components, on the other hand, are called *lightweight components* because they don't need their own peer components. Swing top-level containers (applets, windows, frames, and dialog boxes) are heavyweight, however, so that they can draw components they contain on the local display; lightweight components that you place inside top-level containers use the drawing facilities of the top-level containers.

There is a large set of Swing components that includes one component for each AWT component plus many variations. A Swing component type starts with a *J* to distinguish it from an AWT component; thus, while `Button` is an AWT component, `JButton` is a Swing component.

Swing components conform to the JavaBeans component specification.

Inside Swing components

In order to facilitate pluggable look and feel and otherwise provide increased flexibility, the Swing components use a *model-view-controller* paradigm. That means that the data storage (model) and the look and feel (view), are embodied in separate objects that are managed by a third object called a *controller*. Thus, when you create a component such as a `JButton` object, that (controller) component actually creates two other components, a `ButtonUI` object and a `ButtonModel` object. Note that the Swing components create the auxiliary objects for you—you never have to create them yourself. You may, however, sometimes want to create the model explicitly, so that you can change the way the component handles data. That approach probably wouldn't make sense in the case of a button, but often makes sense in the case of more complex components such as tables.

The *UI object* handles UI functions such as appearance, installing, deinstalling, painting, updating, sizing, and posting events. It extends from the `ComponentUI` abstract class of the Swing package, so the UI object always has some basic methods you can use. The UI object decides on the look and feel by asking the `UIManager` object that resides in the Java run-time environment what the look and feel should be and acting on the response. (Note that the look and feel is thus determined for the entire UI at once, and cannot be changed only for an individual component.)

The *model object* stores the data and state of the component. For example, a `ButtonModel` object stores the state of a button. For more complex objects, such as `JTable`, you may need to manipulate the model object to describe the data. See "Specifying a model for a Swing component" on page 8-18 for more information.

The structure of a Swing UI

You can divide the Swing components into three groups: top-level containers, other containers, and all other components. The top-level containers are:

- `JFrame`
- `JApplet`
- `JWindow`
- `JDialog`

These containers provide the context in which lightweight components can draw; thus, all the other Swing components must go inside one of these containers.

When you create a new Swing project in Visual Cafe, you choose one of the following project templates:

- JFC Application for an application. This creates a `JFrame` object.
- JFC Applet for an applet. This creates a `JApplet` object.

(If you have the Database Edition of Visual Cafe, you can also choose the DataBound Project Wizard, which lets you create an applet or application that is prepared to handle a database.)

You can then choose components from any of the Swing tabs of the Component Palette or Swing sections of the Component Library and drop them into the `JFrame` or `JApplet` object. (Swing menu components are in the Menu & Menu Items section of the Component Library.) You can also add non-Swing components, but you need to take care when doing so. See "Using non-Swing components in a Swing project" on page 8-26 for information.

If you add components programmatically, you should be aware that you do not add components directly to a top-level container. Each Swing top-level container has a content pane, and you add your components to that content pane. For example, when you add a text field to the main container in the Form Designer, the code generator adds a line like this to the main container's code:

```
getContentPane().add(jTextField1);
```

To use JWindow or JDialog as a top-level container:

1. Choose Form from the Insert menu.
2. Choose `JWindow` or `JDialog` from the list of choices in the dialog box that appears.

   Depending on which you chose, `JWindow` or `JDialog` is established as the top-level container.

About JComponent features

All Swing components are subclasses of the Swing `JComponent` class, which is a subclass of `java.awt.Container`.

Here are some standard features of `JComponents`:

- Borders – A border is a visual container for a component, such as a raised bevel line surrounding a button. See “Specifying a border for a Swing component” on page 8-12 for information.
- Icons – You can display a graphic on many Swing components; for example, you can have a static or animated image on a button. See “Specifying an icon for a Swing component” on page 8-15 for information.
- Tool tips – You can supply a tool tip string in the `ToolTipText` property in the Property List. The appearance of the tool tip depends on the look and feel. See “Specifying tool tips for Swing components” on page 8-12 for information.
- Keystroke handling – You can associate a keystroke mnemonic with a component such as a button or menu item so that when the specified key is pressed along with a command key (such as ALT in Windows), an action occurs. (In the case of menu items, if the menu is already open, the user doesn’t need to press the command key.) In addition, you can associate command-key sequences with a component so that when the keys are pressed an action occurs. The keystrokes are defined by `KeyStroke` objects, which are registered with the `registerKeyboardAction` method. Some components have defined key assignments to be used with a standard look-and-feel. For example, for the metal look-and-feel, `JButton` has the following key assignments: press TAB to navigate forward, press SHIFT-TAB to navigate backward, and press ENTER, the space bar, or ALT-*character* (if defined) to activate the button.

- Auto-scrolling – You can enable auto-scrolling with the `Autoscrolls` property in the Property List. For example, if you have a text field within a scrollable pane, you could drag your cursor over text and scroll (similar to a word processor window). You can also use the `scrollRectToVisible` method (if the parent component is a scrolling component); the `VisibleRect` property in the Property List is read-only.
- Actions – You can use `Action` objects to implement program functions so that one piece of code can handle user requests coming from menu items or toolbar buttons. Visual Cafe includes two special components intended for use with Action objects: `JActionMenuItem` and `JActionButton`.
- Double buffering – You can use an off-screen buffer to draw an image more smoothly (with less flicker); components first draw to a buffer that is not visible to the user, then draw this buffer on the screen. If a parent component has a buffer, the child components share that buffer. In the Property List, you can set the `DoubleBuffered` property for a component that extends from `JComponent`. The standard Swing forms, such as `JFrame`, have a buffer that child components will share when `DoubleBuffered` is enabled.
- Slow-motion rendering – You can use this feature to debug a component and optimize how it redraws itself. You can set this in the Property List by setting the value of the `DebugGraphicsOptions` property or you can use the `setDebugGraphicsOptions` method to set it programmatically. The debug graphics options allow you to initiate slow-motion rendering and to print messages to the console, make the graphics flash, and display a window that shows operations performed in the off-screen buffer.
- Sizes – The minimum, maximum, and preferred size values for a component are used when the component is resized. Usually, the UI object determines these sizes based on the component state, such as whether the component has a label or border. You can programmatically change these values.
- Accessibility support – The `Accessible` interface lets you provide UI enhancements for assistive technologies, such as magnified viewers for people with impaired vision.

Mixing Swing and AWT components

Because all Swing components, including Swing containers, are based on the AWT `Container` class, you can mix AWT and Swing components in your programs. However, it is best to avoid doing so. The biggest problem has to do with z-order (also called display order). See “Determining component z-order (display order)” on page 8-21 for more information. Heavyweight components such as AWT components always display over lightweight components, so the AWT components may hide the Swing components unless you’re careful. See “Using non-Swing components in a Swing project” on page 8-26 for more information.

About Swing windows and applets

Swing has `JFrame`, `JDialog`, `JApplet`, and `JWindow` classes, which are similar to the corresponding AWT classes. One difference is that windows and applets in Swing have a Root pane, which is an instance of the `RootPane` class and has these characteristics:

- Supports menu bars (`JFrame` and `JApplet` only)
- Provides a Content pane to hold other components; for example, you could use `getContentPane.add(myjbutton)` to add a button
- Provides layers for internal frames, which are windows inside other windows; for example, Visual Cafe MDI mode has windows inside of the main window (this feature is not currently supported by the Visual Cafe Form Designer)

Windows have a `DefaultCloseOperation` property, which can cause the window to hide, be disposed of, or do nothing for a close event. There are also several `WindowListener` methods that notify components of `WindowEvents`, such as the window being activated or closed.

Customizing Swing components

Swing gives you more control over components than does AWT. If you wish, you can modify various aspects of Swing components; these modifications are not available for non-Swing components. With Swing components, you have control over the following features:

- You can control the look and feel of the components in your application.
- You can specify tool tips.
- You can select a border.
- Many of the Swing components can display an icon.
- You can specify the data model for most components. The data model stores the component's data. Thus, if you want, you can control how and where data is stored.
- You can determine the components' z-order (display order).

These features are discussed in the following sections.

Note: You don't have to control any of these features; they all have default values.

Creating a Swing-based project

You can create Swing-based applications and applets in Visual Cafe. In general, you should use Swing components only in Swing-based projects. The following sections apply just to Swing components; for more information on using components, see Chapter 7, "Working with Components."

Overview of creating a Swing-based project

This section describes the basic steps involved in setting up a Swing-based project.

To create a Swing-based project:

1. Choose New Project from the File menu.
2. Choose the type of program you want to create:
   - ❖ If you want to create an application, choose the JFC Application project template.

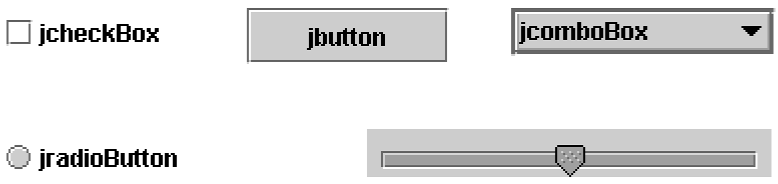
- ❖ If you want to create an applet, choose the JFC Applet project template.
- ❖ If you have the Database Edition of Visual Cafe and you want to create a databound applet or application, choose the DataBound Project Wizard.

3 Insert components from the Swing Containers and Swing tabs in the Component Palette.

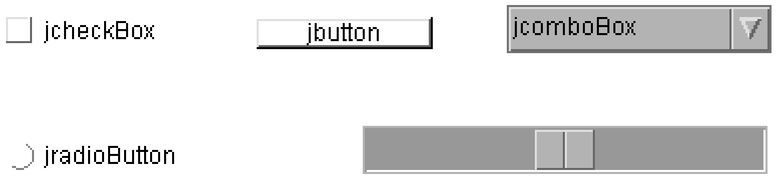
Choosing a look and feel

With AWT components, the look and feel (appearance and behavior) of the components is generally determined by the local system. With Swing components, you control the look and feel.

Here's a selection of components with the Windows look and feel:



Here's the same set with the Motif look and feel:



Not all look-and-feels are available on all systems. In particular, the Windows look and feel may not be available on non-Windows systems and the MacOS look and feel may not be available on non-MacOS systems. Here is the current list of look-and-feels:

- ◆ The Macintosh look and feel uses MacOS-style components.
- ◆ The Windows look and feel uses Windows-style components.

- The Motif look and feel uses Motif/CDE-style components.
- The Metal look and feel, also called the Java look and feel, is the cross-platform Java style. (You may occasionally see references to the Organic look and feel; Organic is an older Java style that has been replaced by Metal, and is no longer supported by Sun.)

Note: The Macintosh look and feel isn't included with Visual Cafe. For more information, point your Web browser to `http://developer.javasoft.com`.

Changing the look and feel of Swing components

You can change the way components look in the Form Designer by using a Visual Cafe menu command. In order to see the changes in a running program, you need to change the look and feel programmatically.

To change the look and feel in the Form Designer:

1. Select the Form Designer.
2. Choose Look and Feel from the Layout menu.

   The submenu lists the available look-and-feels.
3. Select a look and feel from the submenu.

   All Swing components in the project now reflect the new look and feel.

To change the look and feel programmatically:

1. Make sure that the `UIManager` and the `SwingUtilities` classes are imported so that they are available to your program. You can do that in two different ways. One is with two statements like this:

   ```
   import com.sun.java.swing.UIManager;
   import com.sun.java.swing.SwingUtilities;
   ```

   The second way is to import all Swing classes with one statement like this, as is automatically done for you when you create a Swing applet project in Visual Cafe:

   ```
   import com.sun.java.swing.*;
   ```
2. Insert code like this where you want to change the look and feel:

   ```
   try {
       UIManager.setLookAndFeel
   ```

```
        ("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
       SwingUtilities.updateComponentTreeUI(this);
    }
    catch (Exception e)
    {
       System.err.println("Couldn't load look and feel");
    }
```

The `updateComponentTreeUI` call is needed so that the change propagates to all components.

All Swing components in your program reflect the new look and feel.

Of course, the issues involved in setting the look and feel programmatically can't be summed up in such a simple way. For one thing, you can't be sure which look-and-feels are installed on the user's system. For another, you might want the user to control the look and feel, in which case you would need to present a menu with the available choices. The following section provides more information on setting a look and feel.

Finding out which look-and-feels are installed

You may want your program to find out what look-and-feels are installed on the system where the program is running. The `UIManager` has a list of installed look-and-feels. These are stored as an array of `LookAndFeelInfo` objects. You need to import the `UIManager` class, which is in the `UIManager` package, into your program. One way you can do that is by changing the `UIManager` import to:

```
import com.sun.java.swing.UIManager.*;
```

You can get the array with a call like this:

```
LookAndFeelInfo[] lookandfeels =
                   UIManager.getInstalledLookAndFeels();
```

Once you have the array, you can use it to set the look and feel like this:

```
UIManager.setLookAndFeel(lookandfeels[0].getClassName());
```

That call sets the look and feel to be the first look and feel in the list.

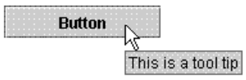
Don't forget that you have to update the component tree with a call like this:

```
SwingUtilities.updateComponentTreeUI(this);
```

Specifying tool tips for Swing components

Every Swing component has a built-in facility for a **tool tip,** which is informational text that appears when the mouse pointer pauses over the component (sometimes called *flyover text*).

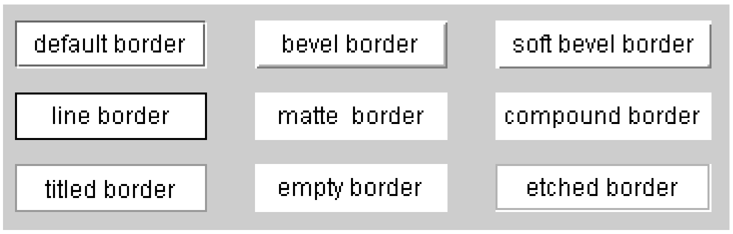
Here is an example of a tool tip:



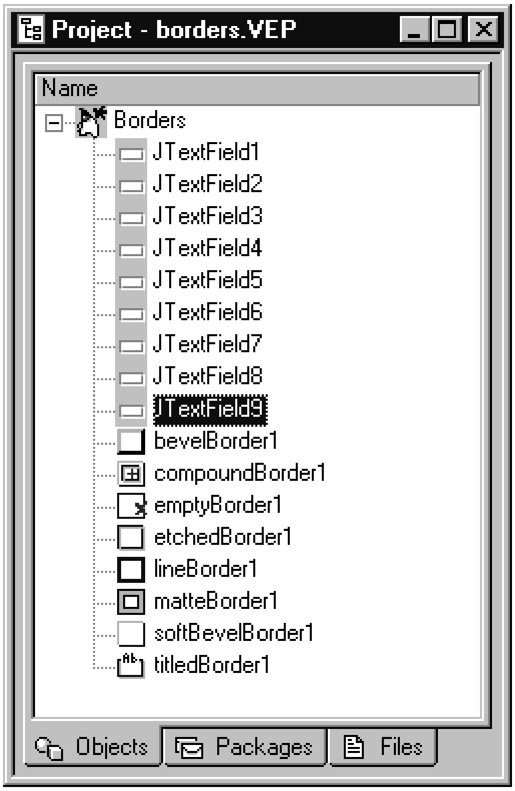
The easiest way to specify the tool tip text is to use the Property List. Put the text that you want in the `ToolTipText` property.

Specifying a border for a Swing component

Every Swing component can have a border. A **border** is a visual container for a component, for example, a raised bevel line surrounding a button. Here are examples of the available borders for a text field:



In Visual Cafe, a *border object* is a non-visual component that appears in the Objects view of the Project window at the same level as the component that created it. For example, here is the Objects view for the project used to create the previous figure:



You can share border objects within the same project across multiple components.

Note: You can manually delete unused border objects from your project. Visual Cafe does not remove them for you.

To specify a border for a Swing component:

1. Choose Property List from the View menu to display the Property List.
2. Select a component in the Form Designer, Project window, or from the drop-down list at the top of the Property List.
3. Find the `Border` property, then select an item from the drop-down list.

The Property List shows items in the `Border` property's drop-down list that are appropriate for that component. Initially, the `Border` property's drop-down menu contains types of borders. When you choose a border type, a new border object is added to your project and the `Border` drop-down list will also show the name of this new object. Every time you choose a border type for the `Border` property in the Property List, a new border object is created and added to the drop-down list.

4 Press ENTER or click somewhere else to make the change.

Note: Press ESC to cancel an edit and return the property to its previous value.

Choosing a layout manager for a container

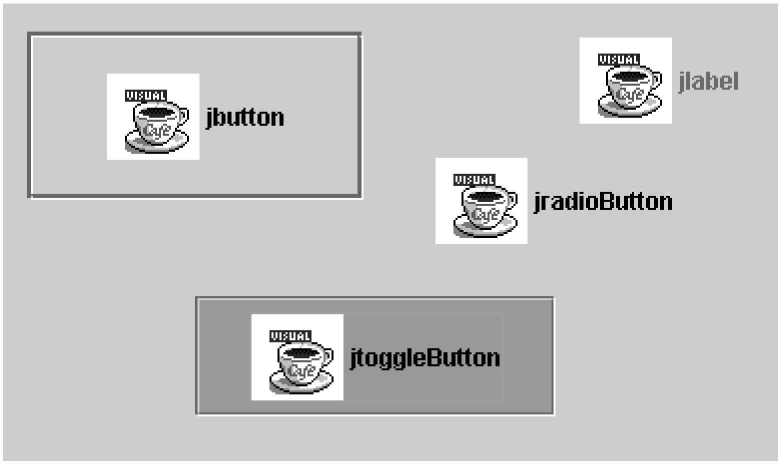
Layout managers in Swing containers work the same way they do in AWT containers, except that Swing provides a few extra choices. (See "Arranging components" on page 7-37 for more information on layout managers.) The new layout managers are:

- `BoxLayout` – This layout manager is similar to the `FlowLayout` manager, but it can lay out components from left to right or from top to bottom, while `FlowLayout` can lay them out only from left to right.
- `OverlayLayout` – This is a small-footprint layout manager that centers an object in its area. It is used by `JButton` and its subclasses. Unlike `BorderLayout`'s centering, `OverlayLayout` does not expand the component to fill the available space, but instead leaves it at its preferred size. Because this layout manager is for button objects, it doesn't appear in the layout manager list for other components.
- `ViewPortLayout` – This is a layout manager for the `JViewport` component. Because this layout manager is for view port objects, it doesn't appear in the layout manager list for other components.
- `ScrollPaneLayout` – This is a layout manager for the `JScrollPane` component. Because this layout manager is for scroll pane objects, it doesn't appear in the layout manager list for other components.

You can set a container's layout manager by opening the Property List for the container and changing the value of the `Layout` property.

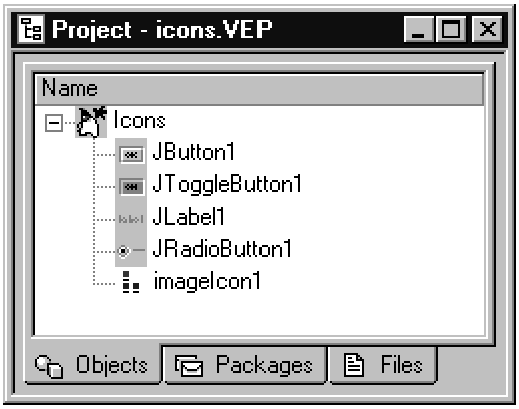
Specifying an icon for a Swing component

The Swing components `JButton`, `JToggleButton`, `JCheckbox`, `JRadioButton`, `JActionButton`, `JLabel`, and `JOptionPane` can display images on their surfaces. Here are some examples:



These images are referred to as icons, because they are usually small, but you can use any GIF or JPEG file, including animated images.

In Visual Cafe, an *icon object* is a non-visual component that appears in the Objects view of the Project window at the same level as the component that created it. Any number of components in a project can use a single icon object. Here is the Objects view of the project used to create the previous figure:



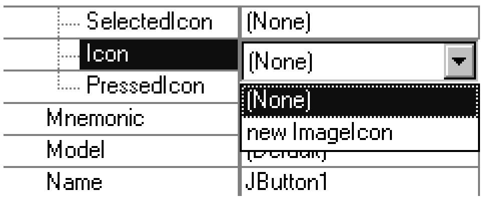
You can set seven icons for your project's buttons. (`JLabel` and `JOptionPane` have some, but not all, of these icons.) All of these properties are available in the Property List in the Icon category.

Name	**Use**
`Icon`	This is the default icon for the component. It displays when none of the other icons show or when another icon would show but hasn't been set.
`RolloverIcon`	This icon displays when the mouse pointer rolls over the component.
`DisabledSelectedIcon`	This icon displays when the component is selected but disabled.
`RolloverSelectedIcon`	This icon displays when the component is selected and the mouse cursor rolls over it.
`DisabledIcon`	This icon displays when the component is disabled.
`SelectedIcon`	This icon displays when the component is selected
`PressedIcon`	This icon displays when a mouse button is pressed on the component.

To set an icon for a Swing component:

1. Choose Property List from the View menu to display the Property List.
2. In the Property List, click the right column next to the name of the icon that you want to set.

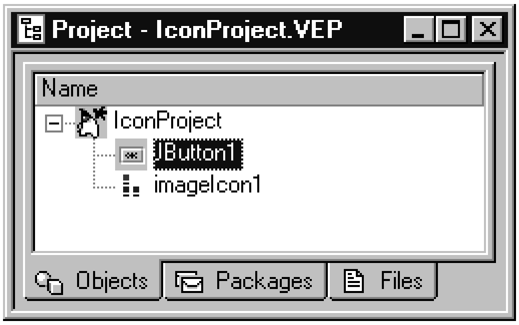
   A drop-down list appears:



SelectedIcon	(None)
Icon	(None)
PressedIcon	
Mnemonic	
Model	
Name	JButton1

3. From the drop-down list, select new ImageIcon.

Visual Cafe creates an `ImageIcon` object and adds it to the list of objects in the project:



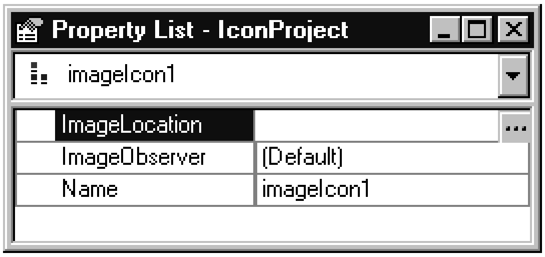
Visual Cafe names the first `ImageIcon` object `imageIcon1`. The next time you pull down any icon menu in the Property List for any object in this project, you see `imageIcon1` listed as well as `ImageIcon`. If you choose `imageIcon1`, then the new icon uses that object. If you choose `ImageIcon`, Visual Cafe creates a new `ImageIcon` object. (Thus, you can have several different icons share the same image or you can have each icon have its own image.)

4 Select the ImageIcon object from the list of objects so that its properties appear in the Property List.

The `ImageIcon` is a non-visual component, so you need to select it from the Objects view of the Projects window or the list of objects in the drop-down list at the top of the Property List.

5 In the Property List, click in the field for ImageLocation.

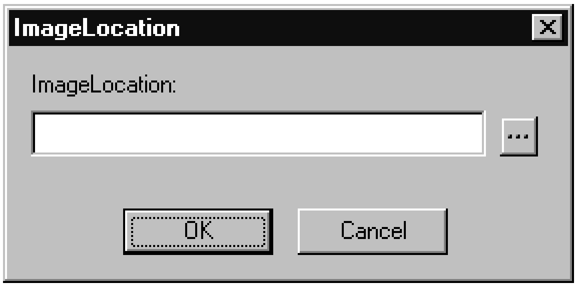
A button with an ellipsis (...) appears in the field:



ImageLocation	
ImageObserver	(Default)
Name	imageIcon1

6 In the Property List, click on the ellipsis (...) button.

A dialog box that lets you specify an image appears:



You can specify a URL or choose a file.

7 To choose a file, click on the button with the ellipsis (...) that appears in the dialog box.

Note: You can manually delete unused icon objects from your project. Visual Cafe does not remove them for you.

Specifying a model for a Swing component

Swing components separate the view of a component's data from the data itself by storing the data in a model object. For example, a `ButtonModel` object stores the state of a button. For more complex objects, such as `JTable`, the data model stores a larger amount of information. In fact, `JTable` uses several models; in addition to the data model, it has a *column model* that handles the appearance of each column and a *selection model* that holds selection information.

You can use this separation of model and view in two ways:

- ◆ You can have more than one view of a single set of data.
- ◆ You can change the way a component stores data without changing the way the component displays the data.

Here's a simple demonstration of sharing data between two components.

To create a table and set its model:

1. Create a new JFC Applet project.
2. Place a `JTable` object in the applet.
3. In the Objects view of the Project window, select JTable1.

4 In the Property List, scroll down to the Model property.

5 Click the right column next to the Model property, where it says (Default).

A list of model types appears.

6 Choose New StringTableModel.

Visual Cafe creates a `StringTableModel` object called `stringTableModel1` and adds it to the project.

To place items in the table:

1 In the Objects view of the Project window, select `stringTableModel1`.

2 In the Property List, click the right column next to the `Items` property, where it says [list].

An empty drop-down list appears.

3 Type `1, 2, 3`.

That action specifies a row with three columns.

4 Press CTRL-ENTER.

The insertion point moves to the next line.

5 Type `4, 5, 6`.

6 Press ENTER.

The table in the Form Designer now shows the two rows with three columns. (You may need to make the table bigger to see all of the columns.)

To associate a second table with the same model:

1 Place another `JTable` object in the Form Designer.

2 In the Property List, scroll down to the Model property.

3 Click the right column next to the Model property.

A drop-down list appears. It looks like the drop-down list for jTable1, but now there is an additional item: stringTableModel1.

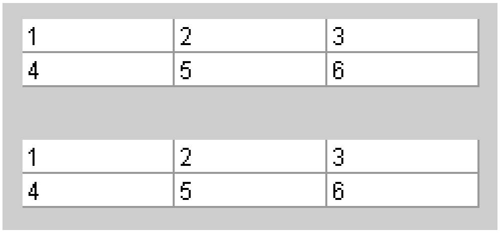
4 Choose stringTableModel1.

Notice that the list of items you entered in the model now appears in both tables.

Note: You can manually delete unused model objects from your project. Visual Cafe does not remove them for you.

You can execute the project, and when you edit either of the tables, both tables show the changed data.

The tables in the executing project look like this:



1	2	3
4	5	6

1	2	3
4	5	6

Here is another simple example.

To make aJComboBox and a JList share a model:

1 Create a new project, which can be a JFC applet or application.

2 Add a `JComboBox` component and a `JList` component.

With `JComboBox` and a `JList` components, you need to define a model in order to add items in Visual Cafe. Both components have default models so that you can add items within a program by calling model methods. However, you can't see the default models in Visual Cafe, so you can't add items using the Property List unless you explicitly set the model. This is also true with `JTable`.

3 In the Property List, choose the Model property of the JComboBox component.

4 Choose New StringComboBoxModel.

Visual Cafe creates a model called `stringComboBoxModel1` and adds it to the project.

5 Choose the Model property of the JList component.

The drop-down list now includes stringComboBoxModel1.

6 Choose stringComboBoxModel1 from the drop-down list.

7 In the Objects view of the Project window, choose stringComboBoxModel1.

8 In the Property List, click the right column next to the Items property, where it says [list].

An empty drop-down list appears.

9 Type `one`.

10 Press CTRL-ENTER.

The insertion point moves to the next line.

11 Type `two`.

12 Press CTRL-ENTER.

13 Type `three`.

14 Press ENTER.

15 Choose Execute from the Project menu to run the program.

You see the same items in the combo box and the list.

Determining component z-order (display order)

You can determine the z-order (also called the *display order*) of Swing components, which is useful when you place components on top of other components. The **z-order** is the order in which components are displayed, and affects components' visibility when they're opaque. (You can make Swing components opaque by setting the `Opaque` property.) The z-order also determines which component receives mouse events.

You determine the z-order in Visual Cafe by using the Objects view of the Project window. Objects at the top of the list are written to the screen after (that is, on top of) objects on the bottom of the list. When an event occurs in a part of the display that contains more than one component, the component that's highest on the list gets the event.

Controlling the display of expert and read-only properties

The programmer who coded the components that you use may have decided that some properties are expert properties, which means they are only displayed in the Property List when you tell Visual Cafe that you want to see expert properties. (The term "expert properties" only means that the programmer defined those properties as "expert.") You can also tell Visual Cafe whether or not you want to see read-only properties.

To control the display of read-only and expert properties:

1 Choose Environment Options from the Tools menu.

2 In the Environment Options dialog box, click the Property List tab.

3 Select or deselect Show Expert Properties and Show Read-Only Properties as needed.

4 Click Apply to make the change, or OK to make the change and close the Environment Options dialog box.

Working with Swing menus

Swing includes the `JMenuBar`, `JMenu`, `JMenuItem`, and `JMenuItem` components for building menu bars. Visual Cafe adds the `JActionMenuItem` component, the Swing Menu Designer, and the Accelerator Editor.

About the Swing Menu Designer

You can create menus by dropping these components in the Form Designer or the Objects view of the Project window, but Visual Cafe includes a Swing Menu Designer that makes it easy to create menus. Visual Cafe also adds the `JActionMenuItem` component for creating a menu item implemented by an `Action` component. (See the following section, “Using Action components in menus and toolbars.”)

To display the Swing Menu Designer:

1 The first step depends on whether your project is an applet or an application.

   ❖ If you used the JFC Applet template, drop a `JMenuBar` object into the Form Designer or the Objects view of the Projects window.

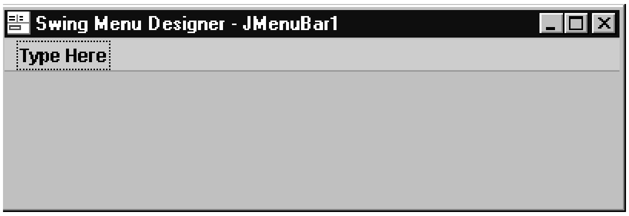
   ❖ If you used the JFC Application template, you already have a menu bar object, so you don’t need to add one.

2 Select the menu bar object.

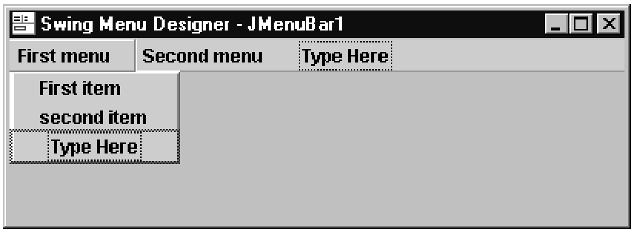
   You can select it in the Form Designer, in the Objects view of the Projects window, or by using the drop-down list at the top of the Property List.

3 Choose Edit JMenuBar from the Objects menu.

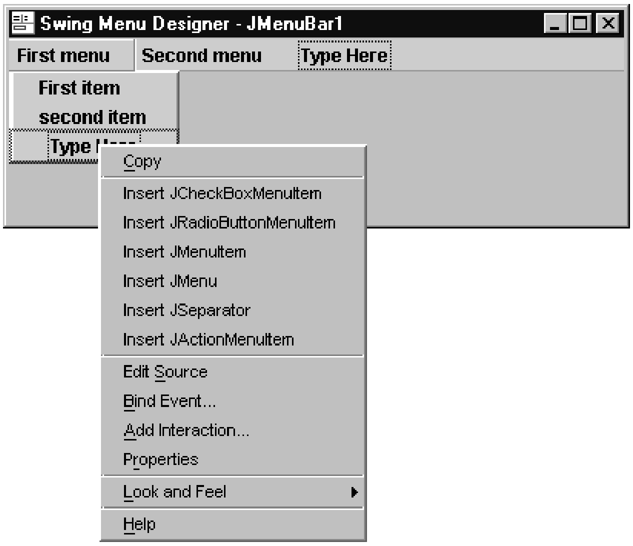
The Swing Menu Designer appears:



You can use the Swing Menu Designer to add menus and menu items by typing menu item names where the Menu Designer says Type Here:

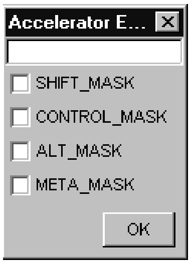


You can also right-click and choose objects and commands from the menu that appears.



About the Accelerator Editor

The `JMenuItem` and `JActionMenuItem` components include an `Accelerator` property. You can define a key combination (SHIFT-L, for example) for a menu item by setting the value of this component. If you click the ellipsis (...) button that appears when you click this property, the Accelerator Editor opens:

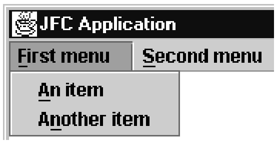


You can specify a character and a modifier key for the accelerator.

About mnemonics

Mnemonics are different than keyboard accelerators. You can define a mnemonic for a menu item by setting the value of the `Mnemonic` property to any character that appears in the menu item’s text. Visual Cafe automatically underlines that character in the text, so the user can see what the mnemonic for that menu item is (Close, for example). When the menu is pulled down and the user presses that key, the menu item executes.

You can also define a mnemonic for a menu. You set it in the same way — by defining the value of the `Mnemonic` property for the menu. When the user presses the ALT key and the underlined character the menu opens, allowing the user to use the mnemonics for the menu items. (Buttons can also have mnemonics, which work in the same way as menu mnemonics: the user presses ALT along with the mnemonic character.) Here is a set of menus and menu items, all of which have mnemonics:



Using Action components in menus and toolbars

Many applications offer the same commands in toolbars and in menus. For example, an application might have a Cut command that appears both in the Edit menu and on a toolbar. Swing includes the abstract `Action` class, which can be used to centralize code for user actions.

To use action objects:

1. Insert `JActionMenuItem` objects instead of `JMenuItem` objects in your menus.
2. Use `JActionButton` objects instead of `JButton` objects. You usually use these in toolbars, but you don't have to.
3. Insert an `Action` component for each action that you want to implement.

   When you insert an `Action` component, Visual Cafe adds an `Action` class to your project. It appears in your project as a top-level object, and Visual Cafe creates a Java source file for it.

   You usually have one `Action` class for each pair of `JActionMenuItem`/`JActionButton` objects.
4. Edit the source code of the `Action` class to implement the action.

   When the user chooses any `JActionButton` or `JActionMenuItem` associated with the `Action` object, Java calls the method `Action.actionPerformed`, so that is the method that you need to implement.
5. In the Objects view of the Project window, drag the icon of the `Action` class into the part of your applet or application that has your `JActionButton` and `JActionMenuItem` objects. (For example, drag the icon into the `JApplet` or `JFrame` class.)

   Visual Cafe creates an object from your `Action` class.
6. Using the Property List, you can now set the `Action` property of the `JActionButton` and `JActionMenuItem` objects to the new `Action` object that you've created.

   When the user chooses the menu item or presses the button, the code that you implemented in your `Action` class executes.

Using non-Swing components in a Swing project

You can freely use lightweight components in Swing projects. The following Visual Cafe components are lightweight:

- All Swing components except for the top-level containers
- All shape components
- `ImageViewer`
- `InvisibleButton`
- `InvisibleHTMLLink`
- `Label3D`
- `RollOverButton`
- `SlideShow`

Other components, including all AWT components, are heavyweight. You can use heavyweight components in Swing projects, but mixing lightweight and heavyweight components causes the following problems:

- Heavyweight components always draw over lightweight components, so they hide any lightweight components that are in the same location. This is the primary problem, and the following section discusses this issue.
- Heavyweight components do not use the look-and-feel mechanism that Swing uses, so you can't control their appearance in the way you can with lightweight components. This restriction also applies to the non-Swing Visual Cafe components listed above.

Mixing lightweight and heavyweight components

As mentioned earlier, the primary problem you may encounter in mixing lightweight and heavyweight components has to do with the fact that heavyweight components always display over lightweight components that are in the same container. This is because lightweight components use the drawing context of their nearest heavyweight container, while heavyweight components create their own drawing context.

You cannot, therefore, place lightweight components and heavyweight components in the same container if you want the lightweight components to overlap the heavyweight components.

There are three less-obvious places where this problem can occur:

- The Swing container `JScrollPane` is a lightweight component, so heavyweight components are not clipped correctly when they are inside this container. (Specifically, a heavyweight component can obscure the scroll bar.) If you want to put heavyweight components in a scroll pane, use the AWT `ScrollPane` component instead of `JScrollPane`.
- `JInternalFrame` is also a lightweight component. The problem here is that heavyweight components in different `JInternalFrame` objects within the same container will always appear on top of the lightweight `JInternalFrame` object. There is no AWT equivalent for this container, so you need to avoid putting heavyweight components in `JInternalFrame` objects.
- The Swing pop-up components `JPopupMenu`, `JComboBox`, and `JMenuBar` can display as lightweight or heavyweight components. Java always makes these heavyweight components under certain conditions (for example, when a pop-up menu extends beyond the edge of a window), but in other cases they will be lightweight if the `lightWeightPopupEnabled` property is `true`. If you have heavyweight components in the window where the pop-up may appear, you should set `lightWeightPopupEnabled` to `false`.

CHAPTER 9

Working with Events and Interactions

This chapter discusses events and interactions and shows you how to work with them in the Visual Cafe environment. You'll learn how to create, edit, and delete interactions using the Interaction Wizard and the Form Designer.

About events and interactions

One of the most powerful features of Visual Cafe is its ability to quickly build a cause-and-effect relationship between two components. This relationship is called an **interaction**. For example, clicking a button can add a line of text to a text field or perform a mathematical calculation.

Any time a button is clicked, a menu item is selected, a box is checked, and so on, a program action called an event is generated. An **event** is a signal that a source-program element sends to a target-program element to notify the target that a particular behavior has occurred. To link those events to a corresponding action in your program, you need an event handler that responds to that event. An **event handler** is a method that's called when an event is triggered.

Visual Cafe lets you create these high-level interactions between components and events. As you create interactions by visually connecting components, Visual Cafe automatically generates the code that's needed to bind the occurrence of an event to an event handler. Therefore, you can assemble interactive applets and applications without writing any code.

The key elements of an interaction are the trigger event and the action component. The **trigger event** is the originator of the interaction and is associated with a component. The trigger event determines when the interaction happens. The **action component** is the component affected by a defined action. The action specifies what to do when a condition is met.

For example, you can connect a button (the trigger component) to a text box (the action component) so that when the user clicks the button (the trigger event), the associated text box is enabled for data input (the action).

Note: An interaction doesn’t have to include two components. You can create an interaction where the trigger and action component are the same. For example, you might create an interaction on an animation component where a mouse click anywhere within the boundaries of the component starts an animation in that component; another click ends the animation.

About interactions in Visual Cafe

You can create, view, and modify interactions visually by using Visual Cafe’s Interaction Wizard. The Interaction Wizard takes you step by step to help you easily create your interactions.

Interactions are represented in the Form Designer by arrows between components. Each interaction arrow links the trigger component to the action component. If they are the same component, the arrow doubles back to the trigger component. Depending on what kind of interactions you create, arrows can appear between components, menu bars, dialog boxes, contextual menus, and other interface elements. For more information, see “Working with interactions” on page 9-5.

If you select an interaction arrow while in the Form Designer, you can see the name of the interaction method in the source code in the Form Designer’s status bar. The name is constructed so that it tells you what object and event trigger this interaction.

You can also selectively filter what interaction arrows are displayed. However, a newly created interaction will always appear in the Form Designer so that you can see the result of your operations in the Interaction Wizard. For more information, see “Choosing which interactions are shown” on page 9-17.

Deleting an interaction is as easy as deleting the corresponding interaction arrow in the Form Designer. For more information, see “Deleting an interaction” on page 9-17.

You don’t have to modify the source code in any way to work with interactions, although you can do so if you wish. Visual Cafe will represent your manually coded interactions in the Form Designer so that you can manipulate them visually as well. For more information, see “About interaction source code” on page 9-18.

Note: If an interaction is incomplete, an alert will display when you try to modify it in the Interaction Wizard. To fix the interaction, you may need to make changes directly in the source code.

Overview of creating interactions

You can create interactions in any of the following ways:

- ◆ By connecting components visually in the Form Designer.
- ◆ By using the Interaction Wizard, which provides help at each step of the interaction creation or modification process (see “Creating an interaction with the Interaction Wizard” on page 9-8 for details).
- ◆ By editing source code in the Source window.

To modify an interaction, you can simply open it in the Interaction Wizard to make your changes. Or, if you prefer, you can edit the interaction’s source code directly.

In general, when you create an interaction you first need to choose the trigger component and event, and then you need to specify what happens as a result of that event: the action. There are two parts to the action: the action component and what happens to it. In the Interaction Wizard, there are three different ways you can specify what should happen to the action component; you can choose to:

- ◆ Perform an action
- ◆ Call a method
- ◆ Set a property

Perform an action lists things you can do with a JavaBeans component, described in English (or your local language). Actions are not automatically derived from the Bean but are explicitly specified by the Bean developer as action descriptions in the `BeanInfo` file. Actions usually call one of the component's methods, possibly with arguments that involve other methods. For example, you could choose the `Toggle` action, and this would be the same as choosing the `getState` and `setState` methods, with a NOT (`!`) operator to change the state. A Bean writer can use actions that could not otherwise be specifed in the wizard.

Note: You can add items to the "perform an action" list in the Interaction Wizard by editing the `ActionDescriptors` in the BeanInfo file. You can add custom actions here. This is helpful in case you want to highlight commonly used methods or allow the user to perform actions that can't be done as simple method calls or property settings. For more information, see "ActionDescriptor" on page 10-28

Call a method lists all accessible methods for the component.

Set a property lists all accessible properties for the component.

If you choose a property, method, or action with arguments, you will be asked to specify a value for the peropety or each of the arguments. Values can either come from an object or an expression you type in. If you choose an object, you will be allowed to select an action, method with no arguments or variable (field) of the object.

If you specify a value that doesn't have the same type as the property or argument but the value can be converted to that type by one of the standard conversions in Java, Visual Cafe will automatically insert the appropriate conversion. The lists that you see of actions, methods, and variables already take this into account and display only selections that have a valid or convertible type.

When you specify a value by typing in an expression, Visual Cafe will scan the expression to ensure that it has the same or a convertible type. If not, you will be warned that a type mismatch has been detected. However, you can always override the warning and generate code for the expression anyway, as the expression might not be valid at the time but will become valid after you add or change some components.

Note: You should not use the Interaction Wizard to create interactions for Swing components placed in an AWT container. These interactions will not work.

Working with interactions

You can create an interaction by connecting:

- two components on a form in the Form Designer
- two components in the Project window's Objects view
- two components across the Project window and Form Designer (for example, a button in the Form Designer and a dialog component in the Project window)
- a component to a contextual menu
- a component to a menu bar menu
- a component to itself
- a form and a component that's contained by the form

Note: In order to create interactions between components, they must be within the same project.

The following tasks about interactions appear in this section:

- Starting an interaction
- Creating an interaction with the Interaction Wizard
- Editing an existing interaction
- Deleting an interaction
- Choosing which interactions are shown

Starting an interaction

When you start an interaction you can visually connect components with the Interaction Tool or go directly to the Interaction Wizard. This section shows you how to set up the initial part of an interaction.

Starting an interaction with the Interaction Tool

You can use the Interaction Tool to begin the process of creating an interaction.

To visually connect components with the Interaction Tool:

1 Open the Form Designer or the Project window's Objects view to display your project's components.

2 Click the Interaction Tool icon in the Toolbar, then drag a line from the trigger component to the action component within the same project.

The action component is highlighted to help you identify which component you selected. (If a component will not highlight, the interaction is probably inappropriate.)

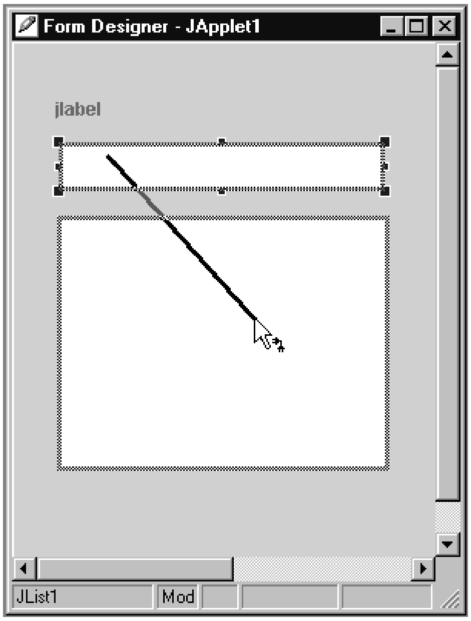
To connect a component to itself, simply double-click the component.

3 Release the mouse button.

The Interaction Wizard appears. (See the following section for information on using the Interaction Wizard.)

Tip: Before you release the mouse button, you can press ESC to cancel the interaction, or move the interaction line to another component.

This is what it looks like when you use the Interaction Tool to create an interaction:



Starting an interaction with the Interaction Wizard

If you prefer, you can go directly to the Interaction Wizard and connect your components from there.

To start connecting components with the Interaction Wizard:

◆ Do one of the following:

- ❖ Select a trigger component in the Form Designer or the Project window's Objects view, then choose Add Interaction from the Object menu.
- ❖ Right-click an item in the Form Designer or the Project window's Objects view, then choose Add Interaction from the pop-up menu.

The Interaction Wizard appears.

Creating an interaction with the Interaction Wizard

You can use the Interaction Wizard to build interactions between components, or between a component and itself. Visual Cafe automatically generates the necessary code for the specified interaction.

To create an interaction with the Interaction Wizard:

1 Open the Interaction Wizard in any of the ways described in the previous section, “Starting an interaction.”

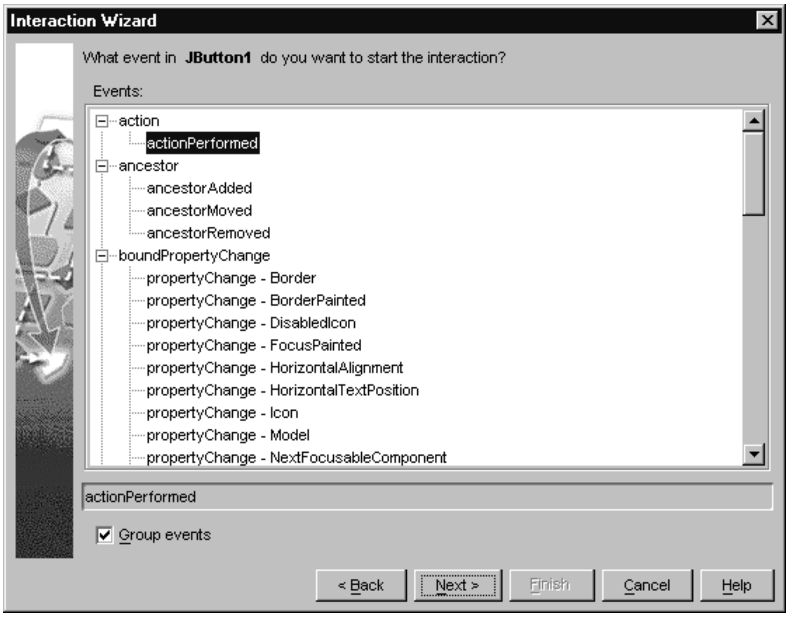
The first page of the Interaction Wizard appears, which lists the tasks you’ll perform. You can choose to not have this first page display the next time you open the Interaction Wizard by selecting the checkbox labeled Don’t show this page in the future.

Click Next to advance to the next page of the wizard.

2 Select the event you want to use to start the interaction.

The available events are listed for the trigger component you’ve already chosen (the component you dragged the Interaction Tool from). The events are grouped by default into similar categories; you can ungroup them and list them in alphabetical order by deselecting the Group Events checkbox.

The default trigger event is highlighted for you, but you're free to choose which trigger event you want to use.



Interaction Wizard

What event in **JButton1** do you want to start the interaction?

Events:

- action
  - actionPerformed
- ancestor
  - ancestorAdded
  - ancestorMoved
  - ancestorRemoved
- boundPropertyChange
  - propertyChange - Border
  - propertyChange - BorderPainted
  - propertyChange - DisabledIcon
  - propertyChange - FocusPainted
  - propertyChange - HorizontalAlignment
  - propertyChange - HorizontalTextPosition
  - propertyChange - Icon
  - propertyChange - Model
  - propertyChange - NextFocusableComponent

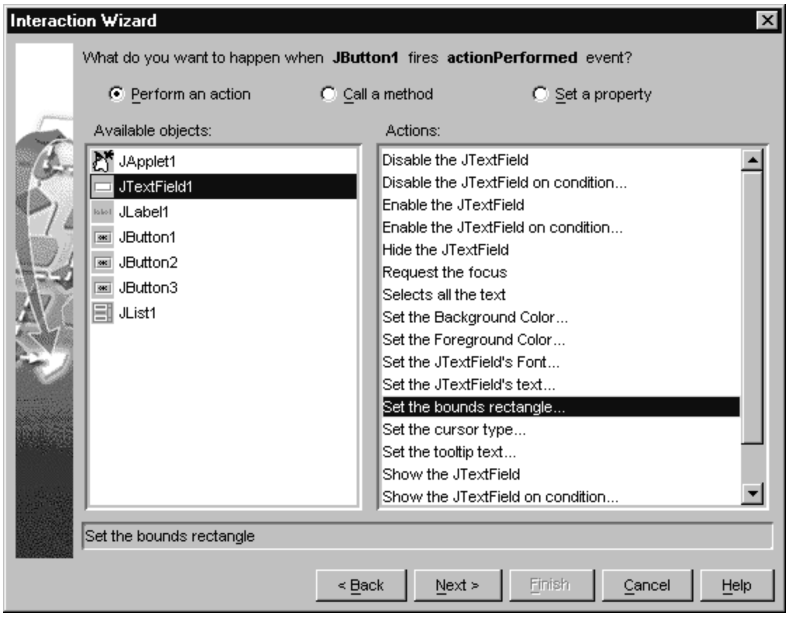
actionPerformed

[✓] Group events

When you're done, click Next to advance to the next page of the wizard.

3 Choose what you want to happen when the trigger component makes the selected event occur. You can choose to perform an action, call a method, or set a property. Notice that available actions are listed in English text, not in Java code format, in order to simplify things for you.

You can choose an action:



Interaction Wizard

What do you want to happen when **JButton1** fires **actionPerformed** event?

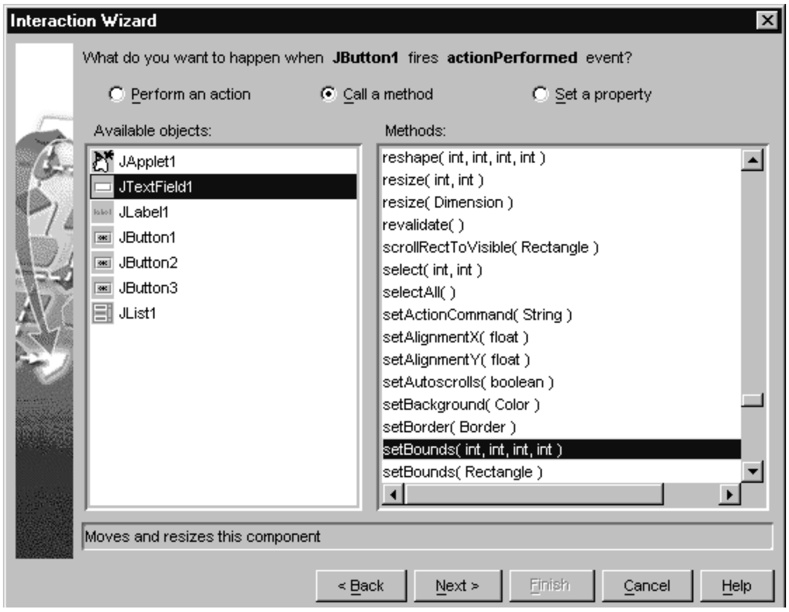
[✓ Perform an action] Call a method Set a property

Available objects:
- JApplet1
- JTextField1
- JLabel1
- JButton1
- JButton2
- JButton3
- JList1

Actions:
- Disable the JTextField
- Disable the JTextField on condition...
- Enable the JTextField
- Enable the JTextField on condition...
- Hide the JTextField
- Request the focus
- Selects all the text
- Set the Background Color...
- Set the Foreground Color...
- Set the JTextField's Font...
- Set the JTextField's text...
- Set the bounds rectangle...
- Set the cursor type...
- Set the tooltip text...
- Show the JTextField
- Show the JTextField on condition...

Set the bounds rectangle

Or choose a method:



Interaction Wizard

What do you want to happen when **JButton1** fires **actionPerformed** event?

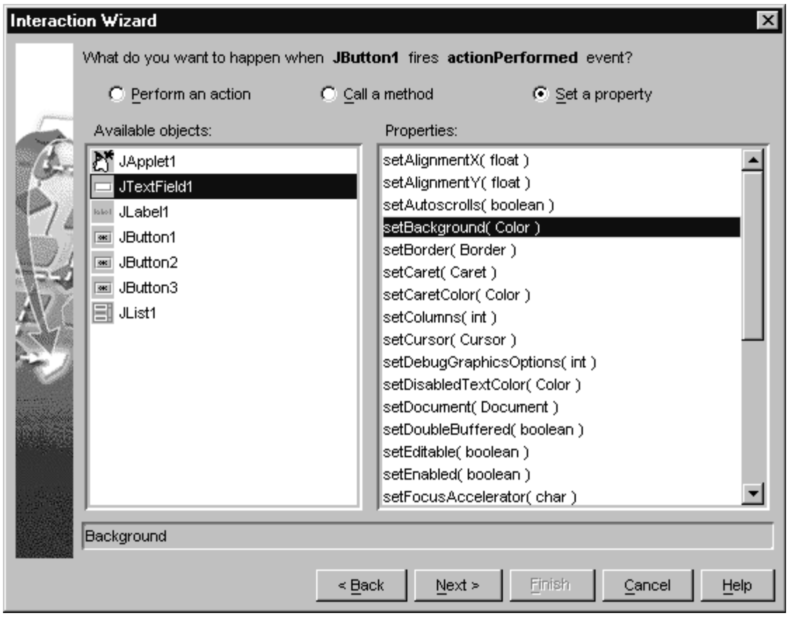
Perform an action [✓ Call a method] Set a property

Available objects:
- JApplet1
- JTextField1
- JLabel1
- JButton1
- JButton2
- JButton3
- JList1

Methods:
- reshape(int, int, int, int)
- resize(int, int)
- resize(Dimension)
- revalidate( )
- scrollRectToVisible(Rectangle)
- select(int, int)
- selectAll( )
- setActionCommand(String)
- setAlignmentX(float)
- setAlignmentY(float)
- setAutoscrolls(boolean)
- setBackground(Color)
- setBorder(Border)
- setBounds(int, int, int, int)
- setBounds(Rectangle)

Moves and resizes this component

Or choose a property:

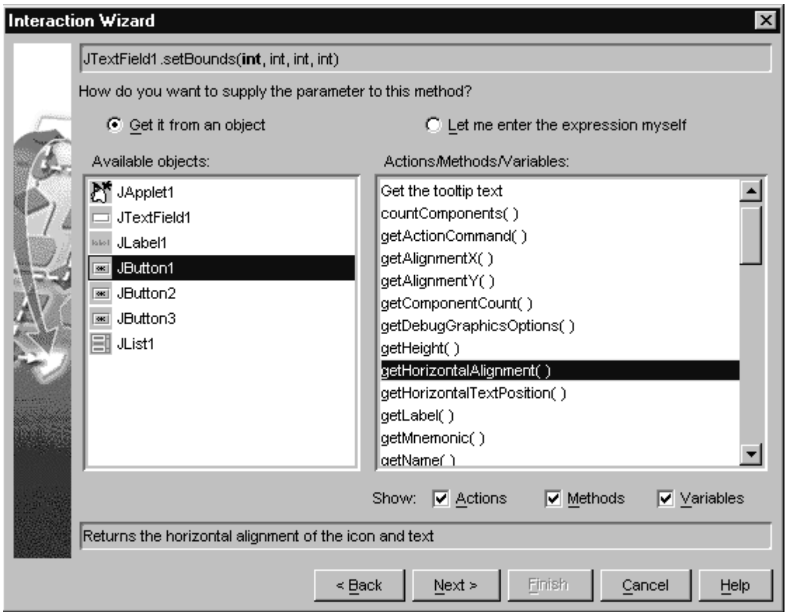


When you select one of the available objects and that object has no actions, methods, or properties, the corresponding radio button – Perform an action, Call a method, or Set a property – is dimmed and cannot be selected.

When you're done, click Next to advance to the next page of the wizard.

4 If you've chosen a property to set or an action or method that has arguments, then one or more additional pages – one for each

argument – appear, prompting you for more information. Select the appropriate responses.



JTextField1.setBounds(**int**, int, int, int)

How do you want to supply the parameter to this method?

[✓ Get it from an object] Let me enter the expression myself

Available objects:

- JApplet1
- JTextField1
- JLabel1
- JButton1
- JButton2
- JButton3
- JList1

Actions/Methods/Variables:

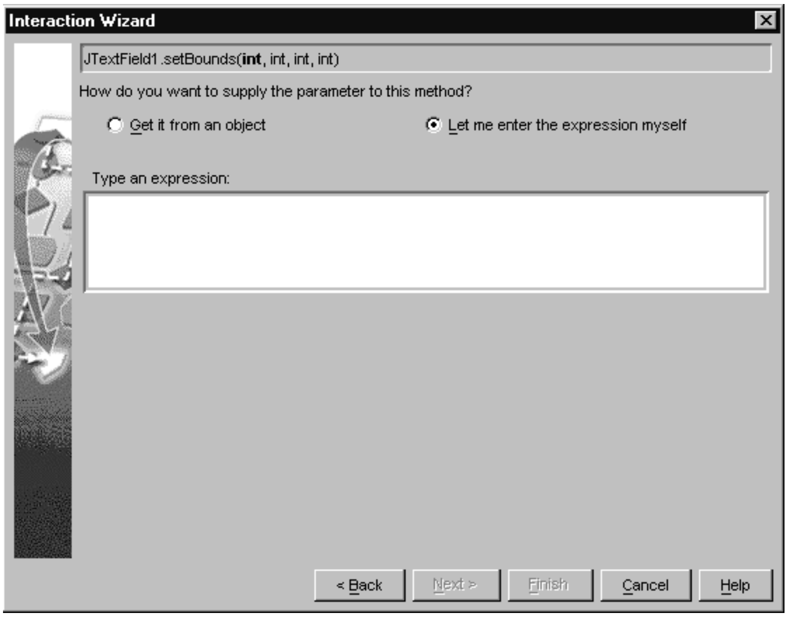
- Get the tooltip text
- countComponents( )
- getActionCommand( )
- getAlignmentX( )
- getAlignmentY( )
- getComponentCount( )
- getDebugGraphicsOptions( )
- getHeight( )
- getHorizontalAlignment( )
- getHorizontalTextPosition( )
- getLabel( )
- getMnemonic( )
- getName( )

Show: ✓ Actions ✓ Methods ✓ Variables

Returns the horizontal alignment of the icon and text

This page may occur several times, depending on the number of arguments needed.

You may also enter an expression if you like. To do so, select the Let me enter the expression myself option, and the appropriate wizard page appears:

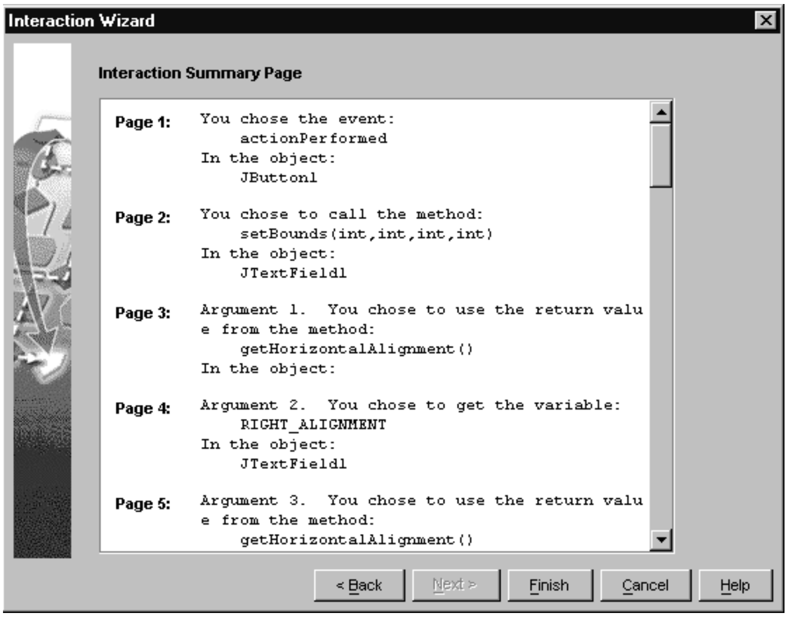


Interaction Wizard
JTextField1.setBounds(**int**, int, int, int)
How do you want to supply the parameter to this method?
Get it from an object
Let me enter the expression myself
Type an expression:
< Back
Next >
Finish
Cancel
Help

You can use quotes if you're typing in a string constant and you don't want to type in escape characters. If you're typing an expression to get a string from another object, you won't want to use escape characters.

When you're done, click Next to advance to the next page of the wizard.

5 On the final page of the Interaction Wizard, you'll see a textual description of your interaction, in outline form. Review this to make sure it's set up the way you want it. If necessary, you can click the Back button to go back to earlier pages in the wizard and make changes.

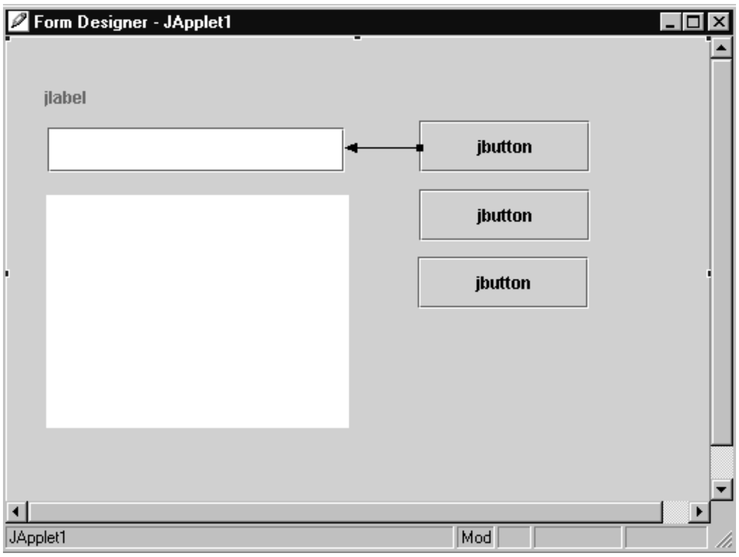


Interaction Summary Page

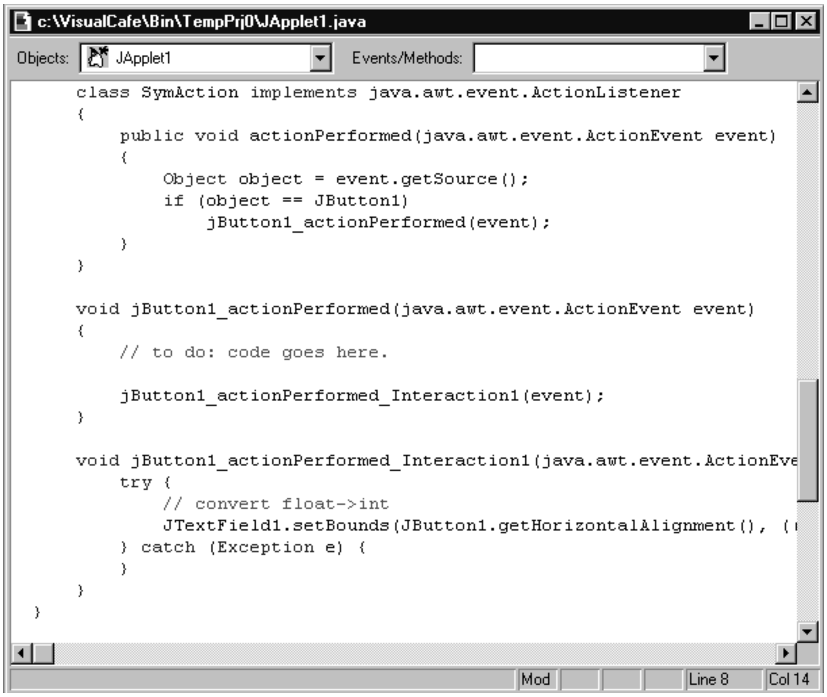
```
Page 1:  You chose the event:
             actionPerformed
         In the object:
             JButton1

Page 2:  You chose to call the method:
             setBounds(int,int,int,int)
         In the object:
             JTextField1

Page 3:  Argument 1.  You chose to use the return valu
         e from the method:
             getHorizontalAlignment()
         In the object:

Page 4:  Argument 2.  You chose to get the variable:
             RIGHT_ALIGNMENT
         In the object:
             JTextField1

Page 5:  Argument 3.  You chose to use the return valu
         e from the method:
             getHorizontalAlignment()
```

Click Finish to close the wizard.

Code is generated in your source file as appropriate, and an arrow appears in the Form Designer between the trigger and action components, or to and from the same component if you've created an interaction that starts and ends with the same component.



The appropriate code has now been generated and is part of your source code. You can view it in the Source window if you wish:



```
class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == JButton1)
            jButton1_actionPerformed(event);
    }
}

void jButton1_actionPerformed(java.awt.event.ActionEvent event)
{
    // to do: code goes here.

    jButton1_actionPerformed_Interaction1(event);
}

void jButton1_actionPerformed_Interaction1(java.awt.event.ActionEve
    try {
        // convert float->int
        JTextField1.setBounds(JButton1.getHorizontalAlignment(), (
    } catch (Exception e) {
    }
}
}
```

If you later want to modify an interaction, you can use the Interaction Wizard (see the following section for details).

If you modify the source code directly, it's possible that the interaction will only be viewable in the Source window, although Visual Cafe makes an effort to parse the source code and make it visually editable. For more information, see "About interaction source code" on page 9-18.

Editing an existing interaction

You can easily modify existing interactions using the Interaction Wizard, which you use just as you did when you created the interaction. If you prefer, you can edit interaction code manually by opening the associated file in the Source window.

To edit an existing interaction in the Interaction Wizard:

◆ Do one of the following:

- ❖ Double-click the interaction arrow in the Form Designer. This opens the Interaction Wizard.
- ❖ Click the interaction arrow in the Form Designer, then right-click and choose Edit Interaction from the pop-up menu.

The Interaction Wizard opens and the current settings for the selected interaction display. You can make changes in the Interaction Wizard by using the same wizard pages you did when you created the interaction. Changes are made to your source code once you finish the Interaction Wizard, thereby overwriting the code that was there before.

Deleting an interaction

You can delete an interaction by deleting its interaction arrow in the Form Designer, or you can manually delete it in the source code.

To delete an interaction:

1. Select one of the following in the Form Designer:
   - ❖ An interaction arrow
   - ❖ An object involved in the interaction (the trigger component, the action component, or the source of an argument)
2. Press DELETE or choose Delete from the Edit menu.

   The interaction is deleted.

It's recommended that you delete interactions visually, according to the previous instructions. However, if you're comfortable with modifying interaction source code, you can delete interactions manually.

Choosing which interactions are shown

You can specify what types of arrows associated with interactions that you'd like to view in the Form Designer.

Note: A newly created interaction will always appear in the Form Designer so that you can see the result of your operations in the Interaction Wizard.

To view selected interactions in the Form Designer:

- ◆ Activate the Form Designer, then choose Interactions from the Layout menu, then one of the following:
  - ❖ Show Interactions From
  - ❖ Show Interactions To
  - ❖ Show Interactions To and From
  - ❖ Show All Interactions
  - ❖ Hide Interactions

Visual Cafe displays only the interactions you've chosen.

About interaction source code

When you create an interaction with the Interaction Wizard, Visual Cafe automatically generates source code for the interaction. This greatly simplifies the development process.

If you wish, however, you can add an interaction by writing source code; it will display in the Form Designer, with the appropriate arrows to and from components. If you add interactions by hand, you must follow the conventions for interaction methods exactly. An interaction method name is composed like this:

*objectname_eventname*_`Interaction`*X*.

The interaction method name is the object name, followed by the event name, followed by the word "Interaction" (all separated by underscores), followed by a digit that uniquely identifies the interaction (for example, `button1_ActionPerformed_Interaction3`).

Visual Cafe is particular about the interaction method names in order to keep from confusing your own methods with generated interaction code. For this reason, the ability to add interactions to the source code and have them recognized as such is mainly useful for adding interactions previously generated by the Interaction Wizard — for example, when migrating a file from one project to another or when checking out a new version of a file that others have modified. Of course, you can always add your own method calls or other code to event handlers, but these won't be recognized as interactions unless they follow the above conventions.

You can also go into the source code and change existing interactions. Visual Cafe acknowledges your changes and displays them in the Form Designer.

If you delete the code for an interaction, that interaction will no longer display in the Form Designer.

If for some reason you change the code so that there is incomplete information for the interaction, when you try to edit the interaction in the Interaction Wizard an alert will appear, warning you of an incomplete interaction.

In the Java source file, code is generated for the call handler, listener registration, and adapter/listener class. Code is also generated for any additional operations you specified.

When you create an interaction in the Form Designer or Interaction Wizard, Visual Cafe performs the following steps when generating source code:

1. Visual Cafe generates an adapter or listener implementation for the event. If one was already generated, it's used with the new interaction.
2. In the adapter/listener class, Visual Cafe generates a check for the object requesting the handling of the event and a call to the event handler.
3. Visual Cafe instantiates this adapter/listener class and generates the registration (specifically, `object.add`*type*`Listener`) after the `REGISTER_LISTENERS` tag. If the adapter/listener class has already been instantiated, it is used.
4. An event handler is generated. If the event handler already exists, it is used.
5. A call to the interaction method is generated in the event handler. If the call already exists, it is used.
6. The interaction method for the interaction specified in the wizard is generated. If the method already exists, its contents are replaced.

If you're determined to create, edit, delete, or otherwise modify interactions in the source code, you need to be aware of some issues related to doing so. Being aware of information that Visual Cafe needs in order to render the interactions will save you from having to regenerate or recreate code.

- Names of interaction methods are specially constructed to signal that a method is an interaction and to uniquely identify which interaction it is. You can change the contents of an interaction method but not its name, or it will no longer be recognized as an interaction method.
- If you delete either the call in a handler to an interaction method or the method itself, the interaction arrow will disappear in the Form Designer. If you undo these changes, the interaction arrow will reappear. If you intend to delete the interaction from the source code, you should delete both the call and the method.
- When you change the contents of an interaction method, Visual Cafe scans the method and attempts to reconstruct an interaction that can be edited in the Interaction Wizard. It can do so only if there are no errors in the source code and the interaction can be represented in the Interaction Wizard. Generally, interactions can be represented in the Interaction Wizard if they could have been generated by the wizard in the first place, but in some cases this reverse engineering won't work. For example, you can force the wizard to generate bad code, but you can't force the parser to accept it. Even if reconstruction fails, the method is still recognized as an interaction, the interaction arrow will still show in the Form Designer, you will still be able to visually delete it, and so on. But if you try to edit such an interaction in the wizard, you will be warned that the wizard only knows about the earlier version of the interaction and that finishing such an edit will overwrite your changes in the source code.

About the Java 1.1 event delegation model

In the Java 1.1 event delegation model, events are sent from an event source to an event listener. An **event source** is an object that generates events, such as an AWT component. An **event listener** is an object that implements the appropriate listener interface so that it can receive events. Event sources implement standard methods so that a listener can request that it be notified of events. After a listener registers with a source, it gets called whenever an event of the requested type occurs.

There are two general types of events: *low-level input* (or *window*) *events,* and *semantic events.* Listeners may handle low-level events before the component that originated them does. This allows a listener to consume the event so that it never gets handled by the originating component. All low-level events are handled by all components, with the exception of window events, which are handled only by `Frame` and `Dialog` objects. Semantic events are higher-level events that usually indicate that a component has changed its value.

All Java events extend the `java.util.EventObject` class; low-level input events extend the `java.awt.event.InputEvent` class. All listener interfaces extend the `java.util.EventListener` interface. All event sources implement standard methods of the form `set` *EventType*`Listener` (where a single listener is allowed), and/or `add` *EventType*`Listener` (where multiple listeners are allowed). These methods tell a component to notify the listener of events.

In Visual Cafe, you can usually route an event to a target component and specify the action to be taken by using the Interaction Wizard, which automatically writes event-related code. The Interaction Wiard creates new listeners as needed, adds them to the appropriate components, and performs routine event-processing tasks.

Java defines the 1.1 model events it supports in the `java.awt.Event` class, which extends the `Object` class.

When an interaction is created in Visual Cafe, code for the interaction method is automatically generated. If necessary, code for the listener registration, an adapter/listener class, and an event handler is also generated. All code conforms to the JDK 1.1 event model.

About the Java 1.0 event inheritance model

The old Java 1.0 event inheritance model shouldn't be used for new code. It has been replaced with the new 1.1 delegation model. Code that uses the old event model will still run, but should be phased out.

In the old event inheritance model, all components handled events by overriding the component's `handleEvent` method. Programs that wanted to handle events generated by a component typically either subclassed the component or handled the event in a parent of that component. Unhandled events eventually pass to the `Component` class, which discards events it doesn't recognize.

Working with event handlers

When you're working with events and interactions, you may need to modify event handlers. This section covers the following tasks:

- ◆ Adding an event handler to a component
- ◆ Editing an event handler
- ◆ Deleting an event handler

There is also an example of event handler source code that you can look at. See page 9-24.

Adding an event handler to a component

Event-handler code is automatically added to a component's source code when you add an event to a component.

Visual Cafe changes the source code in four ways when you add an event handler from the Source window:

- ◆ First, Visual Cafe generates an adapter or listener implementation for the event. If one was already generated, it is used with the new interaction.
- ◆ Second, in the adapter/listener class, Visual Cafe generates a check for the object requesting the handling of the event and a call to the event handler.
- ◆ Third, Visual Cafe instantiates this adapter/listener class and generates the registration (specifically, `object.add`*type*`Listener` or `object.add`*type*`Adapter`) after the comment tag `REGISTER_LISTENERS`. If the adapter/listener class has already been instantiated, it is used.
- ◆ Fourth, an event handler is generated. If the event handler already exists, it is used.

If instead you use the Interaction Wizard to create an interaction, Visual Cafe makes the four modifications mentioned previously; in addition, a call to the interaction method specified in the Interaction Wizard is generated in the event handler and the interaction method itself is generated. For more information, see "About interaction source code" on page 9-18.

To add an event handler in the Source window:

1. In the Objects drop-down list of the Source window, choose a component.
2. In the Events/Methods drop-down list, choose the event or method.

   Existing events and methods are shown in bold. If you choose an event or method that is not bold, it is created for you.
3. In the event handler, replace the placeholder text `//to do: code goes here` with appropriate Java code.

Editing an event handler

You can conveniently edit event handlers in the Class Browser and the Source window.

To edit an event handler in the Class Browser:

◆ Select a class in the Class Browser's Classes pane, then a member in the Members pane, and edit the member in the Source pane. For more information, see "Finding a class or class definition" on page 4-25 and "Finding a member" on page 4-28.

To edit an event handler in the Source window:

1. Choose an object in the Source window's Objects drop-down list.
2. In the Events/Methods drop-down list, choose the event or method.

   Existing events and methods are shown in bold. If you choose an event or method that is not bold, it is created for you.
3. Edit the Java code.

Deleting an event handler

Deleting an event handler is very similar to deleting an interaction. The only additional consideration is to make sure that you delete all the interactions that the event handler calls.

For more information, see "Deleting an interaction" on page 9-17.

An example of event handler source code

If you have a button called `nextButton` that displays the next image of a slide show called `slideShow1`, the interaction method could look like this and appear toward the end of the Java source file:

```
void
 nextButton_ActionPerformed_Interaction1(java.awt.event.Act
 ionEvent event)
{
   try {
         //slideShow1 Go to Slideshow’s next image
         slideShow1.nextImage();
   } catch (Exception e) {
   }
}
```

Note: The default name of an event handler is the object name, followed by an underscore and then the name of the action that triggers the event.

Toward the middle of the Java source file would be the listener registration:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
nextButton.addActionListener(lSymAction);
//}}
```

Toward the end of the Java source file would be the adapter/listener class with a call to the event handler:

```
class SymAction implements java.awt.event.ActionListener
{
   public void actionPerformed(java.awt.event.ActionEvent
 event)
   {
      Object object = event.getSource();
      if (object == NextButton)
         nextButton_ActionPerformed(event);
   }
}
```

And the event handler method would call the interaction method:

```
void nextButton_ActionPerformed(java.awt.event.ActionEvent
 event)
   {
      // to do: code goes here.

      nextButton_ActionPerformed_Interaction1(event);
   }
```

If you then add an interaction between `slideShow1` and a `Label` component (in the Form Designer, click the `Slideshow` component with the Interaction tool, and drag an interaction line to the `Label` component), Visual Cafe would generate a new event handler for `slideShow1`, and the listener registration code would change:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
nextButton.addActionListener(lSymAction);
slideShow1.addActionListener(lSymAction);
//}}
```

and the adapter/listener class code would also change:

```
class SymAction implements java.awt.event.ActionListener
{
   public void actionPerformed(java.awt.event.ActionEvent
 event)
   {
         Object object = event.getSource();
         if (object == NextButton)
            NextButton_ActionPerformed(event);
         else if (object == VacationSlides)
            slideShow1_ActionPerformed(event);
   }
}
```

And the new interaction would have its own event handler method calling the interaction method:

```
  void
slideShow1_ActionPerformed(java.awt.event.ActionEvent
event)
  {
     // to do: code goes here.

     slideShow1_ActionPerformed_Interaction1(event);
  }
```

This example shows what happens when you add two interactions that both use the `ActionPerformed` event, and thus the same listener type. If they were to use other events, then different listeners would be used, and the code would look a little different.

CHAPTER 10

Working with JavaBeans Components

In this chapter you'll learn how to use JavaBeans components in Visual Cafe. You'll find out how to create and quickly update JavaBeans components, as well as how to test your components and package them for distribution. This chapter covers:

- JavaBeans services
- Creating a Bean
- Using the JavaBean Wizard
- Changing Bean properties

About JavaBeans and Java

JavaBeans is a portable, platform-independent Java component model. A **JavaBeans component,** or **Bean,** is a reusable component that can be manipulated visually in a builder tool such as Visual Cafe. Beans can be combined to create applications or applets.

Here are some of the features of JavaBeans components that make them attractive to software developers:

- Beans are discrete. Beans are typically small and have very specific functionality. Beans can be used with other Beans to make larger and more complex Java programs. Note that although many Beans are small, a Bean is not restricted in size. For example, a Bean could be a spreadsheet or a fully functional spelling checker.

- Beans are reusable. Beans can be used over and over in unlimited numbers of Java programs. Examples of reusable Beans are Beans that are user-interface elements such as buttons or menus, or Beans that perform various mathematical calculations.
- Beans can be configured visually in some kind of visual tool, such as Visual Cafe. Bean properties can be easily configured to allow your Bean to interact with other Beans that have specific properties.

JavaBeans terminology

Throughout this manual (and in other trade publications) the terms *JavaBeans component* and *Bean* both refer to the same thing. Also keep in mind that the term *JavaBeans* typically refers to the component technology itself and not to multiple Beans.

Basic Bean structure

In the simplest sense, a JavaBeans component consists of three fundamental parts: properties, methods, and events. This is no different from other objects in object-oriented environments. The property portion of the Bean describes the Bean's state; the methods provide the interface to manipulate the Bean's state; and the Bean is capable of responding to events.

A Bean's methods can be public or private. It is through the public methods that the Bean communicates to the outside world.

Support of features in the JavaBeans specification

Visual Cafe supports the following features in the JavaBeans specification:

- Invisible Beans — They're represented at design time by icons on the form.
- Internationalization — Visual Cafe supports internationalized Beans without any issues. In addition, Visual Cafe code generation has the ability to automatically generate code that references resource bundles instead of constant strings.

- Persistence — Visual Cafe supports serialization of Beans, including proper handling of hidden state Beans.
- Event binding — Support for JavaBeans event binding is provided, with the exception of item 6.4.1 in the JavaBeans specification (event methods with arbitrary argument lists) and item 6.5.2 (unicast event methods).
- Properties — Full support for JavaBeans properties is provided, including indexed properties, exceptions on accessor methods, and bound and constrained properties.
- Customization — Both property editors and customizers are fully supported.
- Packaging — Visual Cafe supports and allows the user to specify all of the standard manifest properties for Beans, including Java-Bean, Depends-On, and Design-Time-Only.

About JavaBeans services

For all the fantastic things that JavaBeans components can accomplish, you might think that there's some highly complex system at work. Actually, Beans have been designed to be fairly simple to use. This lack of complexity is one strength of JavaBeans.

The component model that's described by the JavaBeans specification contains five major services:

- Property management
- Introspection
- Event handling
- Persistence
- Application builder support

These services are described in the following sections.

Property management

Properties are virtual variables. They are defined by the methods that access them. Properties can be changed in a number of ways: at run time through setter and getter methods (these methods are described in the following section, “Accessor and manipulator methods”), or by scripting.

The following are some examples of how Bean properties can be accessed:

- Programmatically via public accessor methods
- Visually through Visual Cafe’s property sheets (See “Application builder support” on page 10-8 for information on property sheets)
- Through persistent storage and retrieval of a Bean
- As object fields in scripting environments (VBScript or JavaScript)
- Through a customizer
- Through a custom property editor

Accessor and manipulator methods

Accessor methods are the primary means by which a Bean’s properties are exposed. An *accessor method* is a public method defined in the Bean that reads the property value in the Bean; a *manipulator method* writes the property value of the Bean. A Bean property typically has a pair of accessor and manipulator methods called getter and setter methods. A *getter method* gets (reads) the property value; a *setter method* sets (writes) the property value. Each property must have a getter method; the setter method is optional (although typically included).

Indexed properties

In addition to single-value properties, Beans support *indexed properties*, properties that represent an array of values. Indexed properties in Beans are similar to indexed properties in Java; you access a specific value using an integer index into the array.

Indexed properties are useful when a Bean needs to maintain a group of properties. For example, a property may be a list of names.

Bound and constrained properties

JavaBeans supports two advanced-level mechanisms for working with properties: bound and constrained properties. A *bound property* is a property that notifies any interested party when the property value changes. A *constrained property* enables an interested party to perform a validation on a new property value before accepting the change. An *interested party* is an application, applet, or another Bean that needs to know about changes in the property.

Bound properties are defined at the component level, which means that the Bean is responsible for specifying that a property is bound. For example, the visibility property of a Bean could be a bound property

A constrained property enables an interested party to perform a validation on a changed property value before the Bean accepts the change. A constrained property can be used to enable an interested party to control how the Bean is modified. For example, a date may be designated as constrained where the application containing the Bean needs to limit the valid range of dates.

Bound and constrained properties are not mutually exclusive. A property may be designated as bound or constrained — or neither or both.

Introspection

Introspection is a Bean's ability to make public (or "publish") the operations, methods, and properties it supports, as well as be able to discover operations, methods, and properties of other Beans.

Note: Although the introspection services in JavaBeans are designed primarily for use by application-builder tools such as Visual Cafe, they are a separate service because they can be used independently of Visual Cafe.

Introspection calls on two API processes: the Java Reflection API and the Java Serialization API. The *Java Reflection API* (or simply *reflection*) is a set of classes that looks into a class file and examines the properties, methods, and events of Beans. The *Java Serialization API* (or simply *serialization*) is used to store the class, including its state. Visual Cafe uses these two Java API functions to allow easy creation and modification of Beans.

If there is a `BeanInfo` file associated with a Bean, then Visual Cafe uses introspection with that Bean. If there isn't a `BeanInfo` file, then Visual Cafe uses reflection (see the following section).

Reflection and design patterns

Beans can determine information about another Bean's properties, methods, and events by analyzing the Bean using a set of low-level reflection services. **Reflection** is the process of querying a Bean to determine information about its public facilities and functionality.

These services gather the Bean's information by applying simple design patterns. *Design patterns* are rules that are used to determine information about a Bean from its reflected method names and signatures. The JavaBeans introspection facilities match them based on a specified design pattern of the method names and automatically determine the property they access.

Design patterns rely on method names and signatures that conform to a standard convention as defined in the JavaBeans specification (which can be found on Sun Microsystems' web site at `http://java.sun.com`). This approach to introspection encourages Bean developers to use consistent naming conventions.

Explicit Bean information

Design patterns are not required or strictly enforced; you're free to use whatever naming conventions you want. If you choose to do so, Beans must use the *explicit introspection* facility. You must provide specific information about the Bean, including a property descriptors, method descriptors, and event descriptors. This information goes into a `BeanInfo` class that must be included with the Bean in its JAR (Java Archive) file. Although this feature is not automatic, there may be some situations where this approach is advantageous. For example, explicit introspection is helpful in situations when you want to hide some information (properties, methods, or events) and show only certain information.

The Introspector

The JavaBeans Introspector service can consolidate the reflection and explicit introspection approaches. First, it tries to use information explicitly provided by the Bean's developer. If that information is not provided, the introspector relies on design patterns to try to get the Bean information.

Event handling

The **event handling** facilities determine how Beans respond to changes in state and how these changes propagate to applications (and other Beans). The event handling in JavaBeans works with the concepts of event sources and event listeners.

An *event source* is a Bean that's capable of generating events. An *event listener* is an application (or Bean) that's capable of responding to events. An *event state object* is used to store information associated with an event.

Event sources and event listeners are connected using an event registration mechanism that is part of JavaBeans. This registration links the source with one or more listeners. When the source generates an event, a designated method is called on the listener with an event state object sent as an argument. Event state objects carry information related to the event with them.

Unicast and multicast event sources

In practice, most event sources are multicast event sources. That is, more than one listener can be registered with the event source.

A *unicast event source* is an event source that can generate events for a single listener. A *multicast event source* is an event source that can generate events for more than one listener.

The main difference between the two is that a unicast event source throws an exception when an attempt is made to register more than one listener.

Note: You should avoid using unicast event sources unless there is a specific reason to do so.

Event adapters

Event adapters are classes provided to simplify the implementation of event listeners. They implement the listener interfaces with empty methods. When a listener is needed, using an event adapter allows you to write only the method of interest.

In a Bean's interface are a number of empty methods that act as placeholders. If you want to implement this interface, you would normally have to create signatures for all the methods that are defined in the interface, whether you want to use all the methods or not. A *method signature* is something like the following method declaration: `int Foo (Boolean)`.

By using an event adapter you have to create a signature for only the method or methods you want to use.

Persistence

Persistence is the ability of a component to remember and store its state long after you're finished working with it. When you change the state of your Bean, you might want your Bean to store, or *persist,* the changed state. A Bean's state can change because of some action that affects the Bean during development or at run time.

Bean storage

Beans are stored in JAR (Java Archive) format. These JAR files are ZIP files that contain another component called a manifest file. This **manifest file** contains additional information about what else is in the JAR file. For more information, see “About JAR files” on page 5-54.

Application builder support

The Application Builder Support service that's a part of JavaBeans is what enables your Bean to be smoothly integrated into the Visual Cafe environment

An intended side effect of Application Builder Support is the separation of design-time and run-time code. It would be wasteful to bundle the design-time-specific code with your Bean for distribution (think of the Web bandwidth!). Placing the design-time code into a separate file solves this problem.

Property editors and property sheets

Visual Cafe supports the editing and manipulation of a Bean's properties through property sheets. A **property sheet** is a Visual Cafe user interface element that provides access to property editors for each of the properties of the Bean. A **property editor** is a user interface that the Bean's developer implements to set the default state of one property type. That means, for example, that you use one kind of property editor for strings and another kind for colors.

Property editors can be different, even though they might edit the same property type. For example, two property editors might both edit strings, but one may edit names, and the other addresses. As you can see, a Bean can have many property editors.

Customizers

JavaBeans enables you to visually edit your Beans in Visual Cafe through customizers. *Customizers* are user interfaces that the Bean's developer implements that extend the Bean's standard properties. Customizers can be used to edit many properties at a time, and a Bean can have only one customizer. You can also use customizers to edit hidden values that are otherwise unavailable through the Property List.

About building Beans

You want to design your Bean to be reused — that's one of the main reasons for creating Beans. It's important for you to fully evaluate the functional specifications of a Bean before you begin writing any code. This has always been good programming practice; however, it's especially crucial because of the nature of JavaBeans components and how they're used. For example, you want your Bean to be backward compatible as it evolves, so you need to make sure that the interfaces to the Bean are extensible enough to provide room for added functionality.

Bean design fundamentals

The Bean-design process doesn't need to be complex. At a minimum you should consider questions such as the following:

What does the Bean do?

The answer to this question helps you to clearly identify what the Bean is to accomplish as a reusable piece of software.

How is the Bean used?

The answer to this question depends on the functionality you have in mind. For example, you might be planning a Bean as a specific part of an application. It might be visible or invisible. It might be a customizable component.

No matter what you're planning for the Bean, be sure you have a good idea of how the Bean is to be used by the end user.

What kind of properties, methods, and events does your Bean need?

You need to figure out how the Bean will interact with the outside world. You must consider how the Bean will handle input and output, and events and interactions. How the Bean handles these dictates what properties, methods, and events you need to create.

Creating a Bean

The following sections provide step-by-step instructions for creating a Bean. The following topics are covered:

- Overview of creating a Bean
- Using the JavaBean Wizard
- Testing your Bean
- Packaging your Bean for distribution
- Packaging your Bean for distribution
- Adding an existing Bean to the Component Library
- Deleting Beans from the Component Library

Overview of creating a Bean

Visual Cafe provides a number of tools that make creating JavaBeans components easy, including a JavaBean Wizard.. This section provides an overview of the Bean-creation process; each step is described in detail later in this chapter.

Tip: You can create a component template by dragging a component from the Project window (Objects view) to the Component Library. The source file is copied by Visual Cafe, so you don't have to keep the files in the same location. A component and a component template appear the same in the Component Library, and you add them to projects in the same way.

To create a Bean:

1. Create a new project for developing one or more JavaBeans components.

   You can get started quickly by using the JavaBean Wizard.

   For information about the JavaBean Wizard, see "Using the JavaBean Wizard" on page 10-12.

2. Create the Bean according to the JavaBeans standard.

   This includes adding any necessary support files such as classes, icons, graphics, sounds, HTML documentation, serialization files, and localization files. If you add support files to the project, they're automatically included when you create a JAR file from the project with the Visual Cafe JAR utility. For more information, see "About JAR files" on page 5-54.

3. (Optional) Add to `BeanInfo` some information for better integrating the Bean into the Visual Cafe environment and other environments.

   For details on this step, see "Adding Visual Cafe information to a Bean" on page 10-27.

4. To package your Bean or Beans in a JAR file, choose JAR from the Project menu.

   For details, see "Adding external files to a JAR" on page 5-55.

   Note: When you add a JAR file to the Component Library, all JavaBeans components in the JAR file are added (as specified in the manifest file). You cannot remove one component independently of the others in the JAR file.

5. To use and test the Bean within the Visual Cafe environment, add it to the Component Library.

   For details on this step, see "Adding an existing Bean to the Component Library" on page 10-21.

If you create JavaBeans components and want to use them in a new project, you can update your component project file and have the changes applied to any project that uses the component. See "Automatically updating Beans in the Component Library" on page 10-19 for more information.

Using the JavaBean Wizard

The JavaBean Wizard can help you easily create JavaBeans. The **JavaBean Wizard** steps you through a series of choices and then generates the class and `BeanInfo` source files for you, based on your selections. The wizard creates two files: *name*`BeanInfo.java` and *beanclass*`.java,` where *name* is the name of your Bean.

To learn more about Beans, refer to Sun Microsystems' web site at `http://java.sun.com`.

You can either start the JavaBean Wizard and create a new project, or add a Bean to a new project.

To add a Bean to a new project:

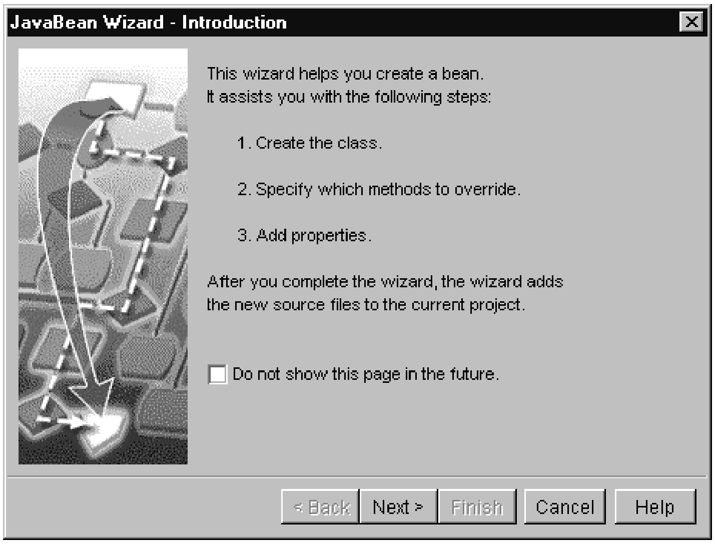
1. Activate the Project window for the project you want to add a Bean to.
2. From the Insert menu, choose JavaBean.

   The JavaBean Wizard opens.

To start the JavaBean Wizard and create a new project:

1. Choose New Project from the File menu.

2 Select JavaBean from the submenu and click OK. The JavaBean Wizard appears:



This wizard helps you create a bean.
It assists you with the following steps:

1. Create the class.
2. Specify which methods to override.
3. Add properties.

After you complete the wizard, the wizard adds the new source files to the current project.

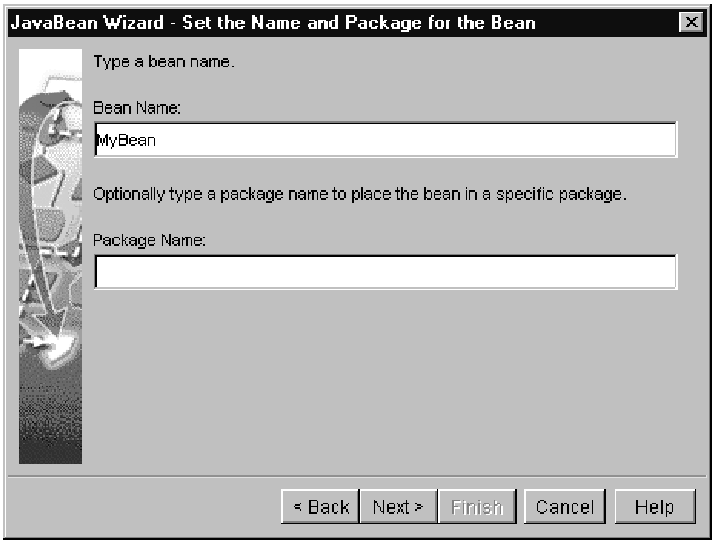
☐ Do not show this page in the future.

If you don't want to access this page again, select Do not show this page in the future.

Tip: If you ever want to view the Introduction page again, simply click Back from the Name and Package page of the wizard.

Click Next to go to the next page.

3 Enter a Bean name and package (choosing a package is optional).


JavaBean Wizard - Set the Name and Package for the Bean

Type a bean name.

Bean Name:

MyBean

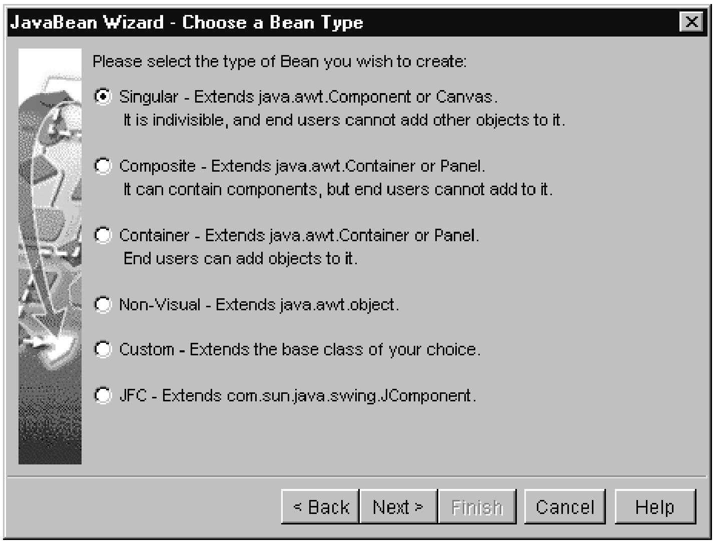
Optionally type a package name to place the bean in a specific package.

Package Name:

The Bean name can contain Unicode letters and digits, starting with a letter, and shouldn't be the same as a Java keyword. There is no length restriction.

Click Next to access the next page.

4 Choose a Bean type.


JavaBean Wizard - Choose a Bean Type

Please select the type of Bean you wish to create:

[✓ Singular - Extends java.awt.Component or Canvas. It is indivisible, and end users cannot add other objects to it.]

Composite - Extends java.awt.Container or Panel. It can contain components, but end users cannot add to it.

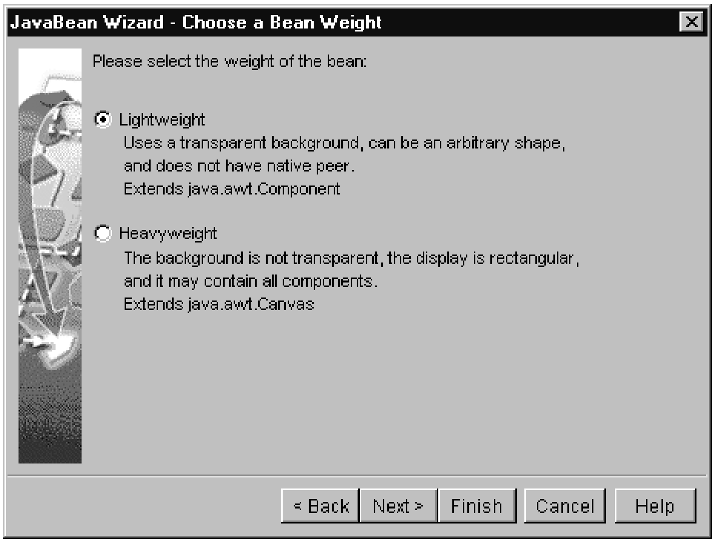
Container - Extends java.awt.Container or Panel. End users can add objects to it.

Non-Visual - Extends java.awt.object.

Custom - Extends the base class of your choice.

JFC - Extends com.sun.java.swing.JComponent.

5 Choose a Bean weight or base class (if creating a custom Bean).

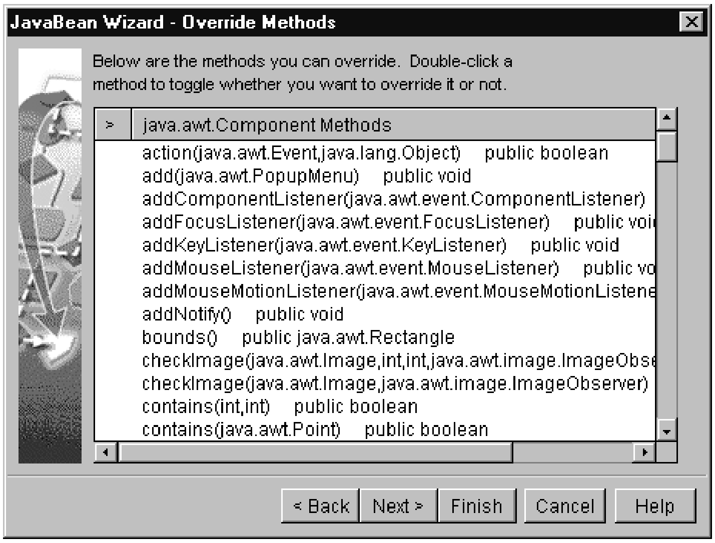


Please select the weight of the bean:

(●) Lightweight
Uses a transparent background, can be an arbitrary shape, and does not have native peer.
Extends java.awt.Component

( ) Heavyweight
The background is not transparent, the display is rectangular, and it may contain all components.
Extends java.awt.Canvas

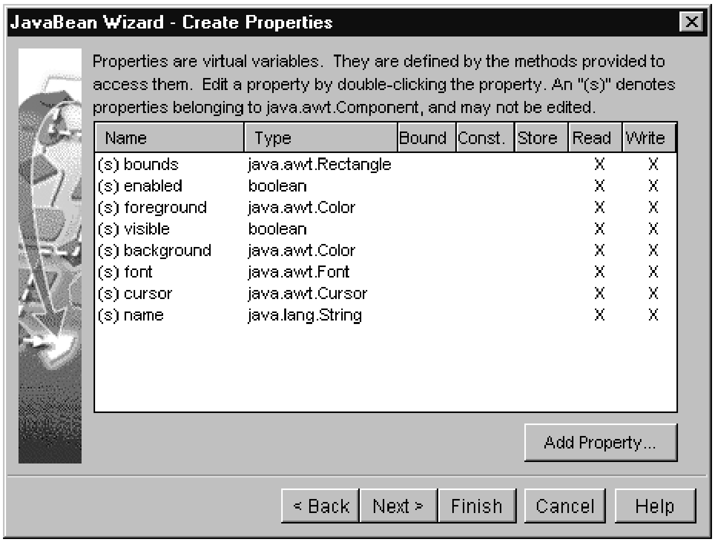
6 Select override methods (optional).



Below are the methods you can override. Double-click a method to toggle whether you want to override it or not.

```
>  java.awt.Component Methods
   action(java.awt.Event,java.lang.Object)   public boolean
   add(java.awt.PopupMenu)   public void
   addComponentListener(java.awt.event.ComponentListener)
   addFocusListener(java.awt.event.FocusListener)   public voi
   addKeyListener(java.awt.event.KeyListener)   public void
   addMouseListener(java.awt.event.MouseListener)   public vo
   addMouseMotionListener(java.awt.event.MouseMotionListene
   addNotify()   public void
   bounds()   public java.awt.Rectangle
   checkImage(java.awt.Image,int,int,java.awt.image.ImageObse
   checkImage(java.awt.Image,java.awt.image.ImageObserver)
   contains(int,int)   public boolean
   contains(java.awt.Point)   public boolean
```

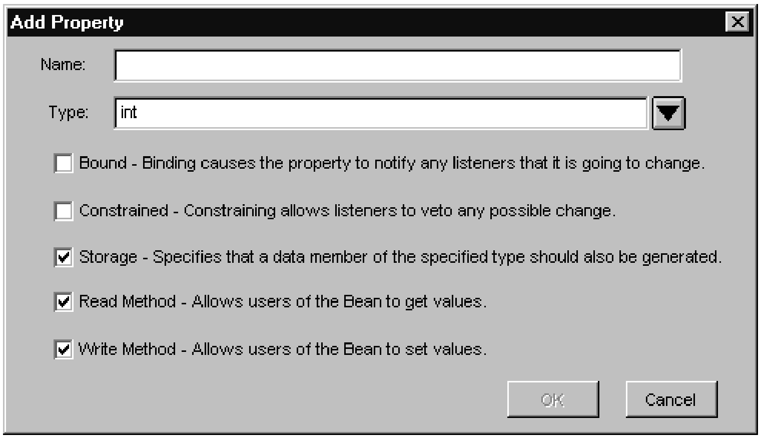
7 Create properties (optional).



Properties are virtual variables. They are defined by the methods provided to access them. Edit a property by double-clicking the property. An "(s)" denotes properties belonging to java.awt.Component, and may not be edited.

Name	Type	Bound	Const.	Store	Read	Write
(s) bounds	java.awt.Rectangle				X	X
(s) enabled	boolean				X	X
(s) foreground	java.awt.Color				X	X
(s) visible	boolean				X	X
(s) background	java.awt.Color				X	X
(s) font	java.awt.Font				X	X
(s) cursor	java.awt.Cursor				X	X
(s) name	java.lang.String				X	X

If you want to add a property, click Add Property. The Add Property dialog box appears.

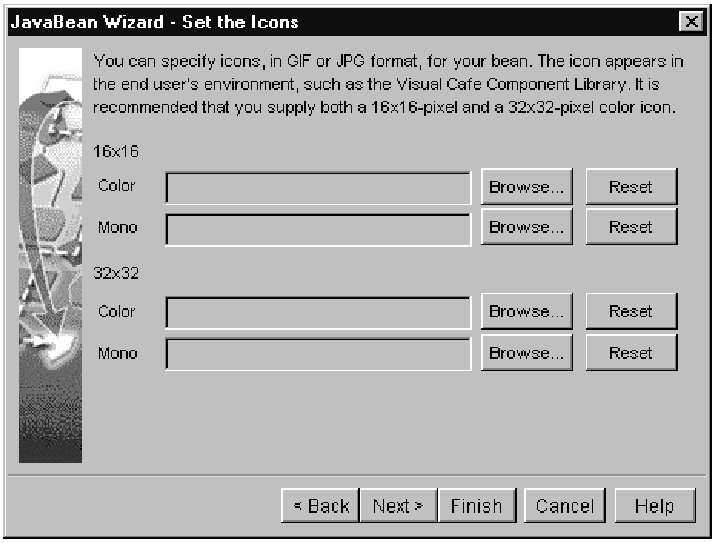


Name:

Type: int

- [ ] Bound - Binding causes the property to notify any listeners that it is going to change.
- [ ] Constrained - Constraining allows listeners to veto any possible change.
- [x] Storage - Specifies that a data member of the specified type should also be generated.
- [x] Read Method - Allows users of the Bean to get values.
- [x] Write Method - Allows users of the Bean to set values.

8 Add icons (optional).



JavaBean Wizard - Set the Icons

You can specify icons, in GIF or JPG format, for your bean. The icon appears in the end user's environment, such as the Visual Cafe Component Library. It is recommended that you supply both a 16x16-pixel and a 32x32-pixel color icon.

16x16

Color

Mono

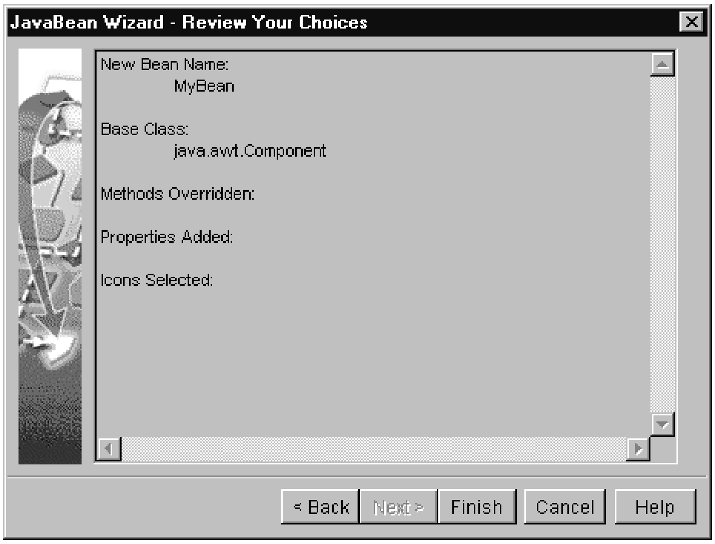
32x32

Color

Mono

If you've chosen to add icons, they are also added to your project, as shown in the Project window's Files tab. This allows the icons to be added to the JAR when you build your project.

9 Review your choices.



JavaBean Wizard - Review Your Choices

New Bean Name:
MyBean

Base Class:
java.awt.Component

Methods Overridden:

Properties Added:

Icons Selected:

Testing your Bean

As you develop your Bean, you can incrementally test and debug the functionality using Visual Cafe. For information on Visual Cafe's debugging environment, see Chapter 6, "Debugging Your Program."

Updating Beans that are local to your project

You can add a Bean to the top level of the Objects view of the Project window, then drag it onto another form in the same project to use the Bean. The Bean becomes part of that form, and yet remains as a top-level object in the Project window. If you modify the Bean, choose Update Project Beans from the Project menu to update the Bean on the form. You don't have to place your bean in a JAR and drag it to the Component Library before you can use it in your projects. In addition, you can view and modify the Bean properties in the Property List.

This process is similar to when you're making changes in a dialog box and then, when you're finished with the changes or want them to be saved, you click OK or Apply. Updating the Beans in your project saves the recent changes and makes the most up-to-date version available for use in other projects.

You can choose to enable or disable Beans that are local to a project upon opening the project.

To enable or disable project Beans upon opening a project:

1. Choose Options from the Project menu.

   The Project Options dialog box appears.
2. If the Project tab isn't already active, select it to make it active.
3. Select Update project beans on project open to update any local project Beans upon opening the associated project.

   Deselect this option if you don't want Visual Cafe to update local project Beans when opening the associated project.
4. Click OK.

   The changes take effect immediately.

Automatically updating Beans in the Component Library

Visual Cafe enhances the development of JavaBeans components by providing a single command that immediately shows updates made to Beans, providing a tool for dynamic development. After modifying the form, source code, or events in your JavaBeans components, use AutoJAR to update the JAR file. You quickly see the results of those modifications in real time, because any open projects using those components are automatically updated.

Choosing AutoJAR automatically:

- saves your source files
- compiles your source files into class files
- creates a JAR file with a manifest file
- adds the JAR to the Component Library

The process of dynamically developing Beans involves preparing a JavaBeans component for dynamic development, modifying code, and updating the Component Library. Each of these steps is explained below.

To prepare a JavaBeans component for dynamic development:

1. Create a Bean using the JavaBean Wizard.

   For details, see "Using the JavaBean Wizard" on page 10-12.

   If you choose to use existing source code by adding a source file to the project, you can include a corresponding *BeanName*`BeanInfo.java` file, where *BeanName* is the name of your Bean.

2. Choose AutoJAR from the Project menu.

   If you want to customize how Visual Cafe creates JAR files, you can do so by setting deployment options for a project or for all projects. For more information, see "Setting deployment options" on page 5-38.

   Your JavaBeans component now has a JAR file and has been added to the Component Library, and all open projects using this component have been updated.

   The manifest file generated by AutoJAR automatically includes the following attributes for class files that follow this pattern:

- If a *name*`.class` file is a public, non-abstract class and has a constructor that takes no arguments, then Visual Cafe will automatically set that class file to be a Bean. It will have the attribute and value of `Java-Bean:True`.
- If a class file is an instance of either `java.beans.BeanInfo`, `java.beans.Customizer`, or `java.beans.PropertyEditor`, Visual Cafe automatically marks the file as `Design Time Only`. It will have the attribute and value of `Design-Time-only:True`.

Note: Make sure you select Add to Library so your JavaBeans component is added to the Component Library. For more information, see “Setting deployment options for a project” on page 5-39.

3 Open a different project and add your new JavaBeans component(s).

Now, if you later modify the code of your JavaBeans component, you can dynamically update it in the Component Library and in all projects where it’s used.

Note: In order to dynamically update your JavaBeans components, before making any modifications to the form, source code, or events of the project that contains the JavaBeans component you should have run AutoJAR on the JavaBeans component.

To dynamically develop with JavaBeans components:

1 Make any necessary modifications to the form, source code, or events of the project that contains the Bean.

2 Choose AutoJAR from the Project menu again to update the JavaBeans component(s) in the Component Library and all instances of the component(s) in any open projects.

Note: If your project is not open at the time you run AutoJAR, all instances of your components are updated the next time you open that project.

Note: If you replace a JAR file added to the Component Library by dragging and dropping from Windows Explorer or by choosing Add Component into Library from the File menu, all instances of components in the JAR that are in any open projects are automatically updated.

Packaging your Bean for distribution

After you've developed and tested your Bean, you need to package it for distribution. Beans are packaged into compressed archive files called JAR files. For more information, see “About JAR files” on page 5-54.

Adding an existing Bean to the Component Library

You might get a Bean from a colleague, from your favorite software source, or from the Internet. Where do you put it?

To use an existing JavaBeans component in Visual Cafe, you must add it to the Component Library. You can add class or JAR files. The Bean must comply with the JavaBeans standard for it to be added. After you add a Bean:

- The component appears in the Component Library.
- If an icon was specified in the `BeanInfo` file, the component uses that icon. If the `BeanInfo getIcon` method returns `NULL`, Visual Cafe uses the icon of a base class already in the Component Library. Visual Cafe examines the classes the JavaBeans component inherits from and picks the class deepest in the inheritance hierarchy. The icon of this class is used with your new JavaBeans component.
- If the `get` and `set` methods conform to the JavaBeans design pattern, the component properties will appear in the Property List window. You can edit properties as a text field, a drop-down list, or in a dialog box, depending on the property. The dialog box appears if you click a field value then click the Browse (…) button when custom property editors are provided.
- Visual Cafe derives the actions, methods, and properties displayed in the Interaction Wizard. It determines the actions from explicit specifications in the `BeanInfo` class. Methods and properties are determined through reflection and introspection.

Note: When you add a JAR file to the Component Library, all JavaBeans components in the JAR file are added (as specified in the manifest file). You cannot remove one component independently of the others in the JAR file.

You can also automatically add the Bean to the Component Library during development. For more information, see "Automatically updating Beans in the Component Library" on page 10-19.

To add a Bean to the Component Library:

1 Make sure the component's class or JAR file is in the location where you want to store it.

   Note: For class files, this location must be in your class path. For example, `c:\VisualCafe\java\lib` is in your class path. Remember that class files are case-sensitive. If you want to add the class file to a package, the class file must be in the package directory.

2 Choose Add Component into Library from the File menu. (This menu item is available only when a project is open.)

   An Open dialog box appears.

3 Select the class or JAR file, then click Open.

   For a class file, an Add to Library dialog box appears.

   For a JAR file, Visual Cafe inserts the Beans into the Component Library. The Beans are put in a group with the same name as the JAR file, unless another group name was specified in `BeanInfo`.

4 For a class file, select a group and then click OK.

   A dialog box appears that lists the number of components that were added in a JAR file to the Component Library.

   The component appears in the Component Library. You should verify that it is there. (You can display the Component Library by choosing Component Library from the View menu.)

5 If you want to add the component to the Component Palette, select the component in the Component Library, right-click the component, and choose Add to Component Palette from the pop-up menu.

   You can also specify the folders in the Component Palette and Component Library from which you want the Bean to be accessed.

Here are some considerations when inserting Beans:

- ◆ If a component is not added to the Component Library, it most likely does not comply with the JavaBeans standard.
- ◆ If you change or move a class or JAR file after it is in the Component Library, the change is not recognized until you restart Visual Cafe or re-add the file.
- ◆ Some components are made of more than one class file (for example, if the Java source file for a component has inner classes). You need to make sure these class files are in the class path. You cannot add inner classes by themselves into the Component Library. In order to add inner classes, they must be part of the JAR file.

Caution: After you insert a JAR file, you should not move it to a new location. If you move it, Visual Cafe will be unable to find it.

Deleting Beans from the Component Library

As you're developing your project, you can delete user-created objects in the Component Library. Deleting a component from the Library also removes the component from the Component Palette.

To delete a Bean from the Component Library:

1. In the Component Library, select the object to be deleted.
2. Choose Cut from the Edit menu or press the DELETE key.

   The other Beans in the JAR are also removed automatically.

Converting component description files to Beans

In Visual Cafe 1.0, you needed to create a separate **description file** (with an extension of `.desc`). This file contained the explicit instructions on how a Bean was supposed to function and what results to expect. In Visual Cafe 3.0, description files are no longer needed for Beans. If you want to reuse a component that you developed in an earlier version of Visual Cafe, or if you want to use third-party components, you need to convert your description file to a Bean. Visual Cafe includes a utility called the Description File Converter to help you with the conversion process.

To convert a Visual Cafe 1.0 component to a Bean:

1 Run the Description File Converter.

To do so, double-click the `\Bin\DescToBeanInfo.bat` batch file, which is the `DescFileConverter` folder of the main Visual Cafe folder.

The Description File Converter dialog box appears. You can view the Java version by going to the Help menu and choosing Environment.

2 Click the Location tab.

3 In the Description File Directory field, enter the full path to the directory that contains one or more description files you want to convert. You can use the Browse button (…) to specify the directory.

4 If the component icons are with the description files, select .ico Files are with .desc files. Otherwise, in the Icon File Directory field, specify the location of the corresponding icon files, if present.

5 In the Output Directory field, specify the full path to a directory where you want your output files (`BeanInfo` and `.gif`) to go.

When the description file is converted, the name of the output file is the first part of the Bean name (without the extension), appended with `BeanInfo.java`; the icon files are converted to `.gif` files.

6 Select relative or absolute to specify the relative or absolute directory.

If you select relative, the fully qualified class name is used to create a directory structure subordinate to the output directory. For example, if the output directory is `c:\temp` and the class name is `symantec.beans.Beans`, then `Beans.java` will be placed in the directory `c:\temp\symantec\beans`.

If you select absolute, all files are placed in the directory you specify. This could potentially cause file name conflicts, because the package directory structure isn't preserved.

7 Click the Selection tab and select the files you want to convert. SHIFT-click to select multiple files. Click Select All Listed Files to select all files.

8 Choose Convert from the File menu.

A dialog box appears when the conversion is complete. The files are displayed in the output directory; if the path was relative, the files are in a directory subordinate to the output directory. For each component entry in a description file, a `BeanInfo.java` file is created. For each icon file, a 16-by-16 and a 32-by-32 `.gif` file is created. For the latter, the 16-by-16 image is expanded to a 32-by-32 size.

Viewing and changing Bean properties

Each Bean has a set of properties that define its look and behavior. Visual Cafe provides a Property List from which you can directly modify these properties. When you change a property, the source code and Form Designer are immediately updated.

To view or change the properties of Beans in a project, select one or more components in the Form Designer or Project window.

To view or change the properties of Beans in the Component Library and Palette, select one or more Beans in the Component Library. Remember that if you change the properties of a component in the Component Library, the change now appears every time you add that component to a project. Projects that already contained the component before you changed it are not affected. (For information on using the Component Library and Palette, see Chapter 7, “Working with Components.”)

Using the Property List to modify Bean properties

You can use the Property List to modify the properties of a Bean.

To modify Bean properties:

1. To display the Property List, choose Property List from the View menu (or press F4).
2. Select a component in the Form Designer, Project window, Component Library, or the Property List’s pull-down menu. To select multiple components, CTRL-click in the Form Designer, Project window, or Component Library, or SHIFT-click or drag in the Form Designer.

   When multiple components are selected, only their common properties are shown and editable. In the Property List, you see the heading Multiple Selection.
3. To edit a property, click the right column, double-click the left column, or use TAB in the Property List.

The right column displays a list of valid values or makes the text string editable.

Properties with multiple values (for example, the `Font` property) are marked with a plus sign (+). Click the + to expand the list.

Properties that are defined with additional information are marked with the ellipsis button (…). Click the … button, and a dialog box appears which is appropriate for that property.

4 Press ENTER or click somewhere else to make the change.

Tip: Press ESC to cancel an edit and return the property to its previous value.

Using a customizer to configure a component on a form

Visual Cafe allows you to call an available customizer to configure the characteristics of a Bean. Use a customizer when you want more guidance for configuring a Bean's behavior than the Property List provides. A customizer can configure more than one property at a time, and many have a wizard-like interface.

Even if Visual Cafe detects a `bd.SetValue("hidden-state", Boolean.True)` statement in a Bean's `BeanInfo` file, you can still access the Bean's customizer. Visual Cafe now supports serialization for hidden-state customizers.

Note: Customizers will vary, because each Bean has a unique customizer created by the Bean's developer. Not all Beans can have a customizer.

To access a Bean's customizer:

1 Do one of the following:

- ❖ Right-click a component in the Project window (Objects view) or the Form Designer, and choose Customize from the pop-up menu.
- ❖ Select a component in the Project window (Objects view) or the Form Designer, and choose Customize from the Object menu.
- ❖ Right-click a component in the Project window (Objects view) or the Form Designer, and double-click the Customize property in the Property List.

If the Customize menu item is not available, the component does not have an available customizer.

2 Make any necessary changes in the customizer.

After you complete the customizer, the appropriate code is generated.

Adding Visual Cafe information to a Bean

Beans have *introspection,* the ability to read JavaBeans classes directly with the Java Reflection API using the Introspector class. This information is stored in a `BeanInfo` object and includes data such as properties, events, and all the accessible methods. In addition, you can add information about how a Bean should integrate into the Visual Cafe environment. The integration information is provided through `BeanDescriptor` attributes specific to Visual Cafe and the com.`symantec.itools.vcafe.beans.ActionDescriptor` class.

A quick description and code samples follow; for more information, see the Java API reference, which is available from the Help menu.

Visual Cafe BeanDescriptor attributes

Visual Cafe supports the standard JavaBeans conventions. In addition, it supports some attributes that allow Beans to work better within the Visual Cafe environment. You can specify the following attributes in the `BeanDescriptor` area of `BeanInfo` to further integrate your Bean into the Visual Cafe environment:

- The name of a Component Library group (folder) to put the Bean in
- The name of a Component Palette tab to put the Bean in
- Whether to forbid users to drop other components into this Bean (if this Bean derives from `java.awt.Container`)
- Visual Cafe flags, such as `INVISIBLE` for specifying non-visual components
- Visual Cafe actions (used by the Interaction Wizard)

The constants for these attributes can be found in `com.symantec.itools.vcafe.beans.BeanDescriptorAttributes`.

ActionDescriptor

The `ActionDescriptor` class extends from `java.beans.FeatureDescriptor`. An `ActionDescriptor` encapsulates a Visual Cafe action, which is used by the Interaction Wizard. A Visual Cafe *action* specifies something done to or by the Bean. The Interaction Wizard allows users to graphically connect these actions together, and Visual Cafe is able to generate the code for the specified action based on the information encapsulated in the `ActionDescriptor`. For more information about the Interaction Wizard, see Chapter 9, "Working with Events and Interactions."

The action is made up of five pieces:

- Form – Determines whether the value of an action is `INPUT` or `OUTPUT`. An input action sets a variable or calls a method; an output action returns a value.
- Type – Sets the Java type of the input or output value of the action.
- Expression – Defines the code that's generated to create the action. Properties and the following replacement variables are allowed in the code string:
  - %*name*% – name of the component
  - %*class*% – short class name of the component (for example, `JButton` rather than `com.sun.java.swing.JButton`)
  - %*arg*% – argument used for output actions
- Description – Supplies an English description of the action. This string appears in the Interaction Wizard. %*class*% is allowed.

A Visual Cafe action looks like a method call. However, an action can be more than that; it can be a "metamethod." Any valid code expression can be used to generate the action; for example, the action "Toggle pause" might have

```
%name%.setPaused(!%name%.isPaused());
```

as the code expression. (In this expression %*name*% is a replacement variable for the name of the component.) An action does not have to be tied to a method; for example, the action "Point the button arrow LEFT" might have the code expression %*name*%`.LEFT`.

The `ActionDescriptor` methods allow you to set the form, type, expression, and description of a particular action. If an action is tied to a specific method, its `ActionDescriptor` is associated with that method's `MethodDescriptor`.

Currently, the association uses the `MethodDescriptor`'s inherited method `setValue`, passing in a Vector of all `ActionDescriptor` objects to be associated with that method.

Code samples

Following are three code samples that show three different ways of implementing the `BeanInfo` information that's specific to Visual Cafe.

The following code sample shows how to use an `ActionDescriptor` in a `getMethodDescriptors` override:

```
import com.symantec.itools.vcafe.beans.ActionDescriptor;
import com.symantec.itools.vcafe.beans.MethodDescriptorAttri
 butes;
```

Before returning a `MethodDescriptor` from a `getMethodDescriptors` override, you'll want to add the action descriptor to it:

```
ActionDescriptor borderNoneActionDescriptor = new
 ActionDescriptor();
borderNoneActionDescriptor.setForm(ActionDescriptor.OUTPUT);
borderNoneActionDescriptor.setType(“int”);
borderNoneActionDescriptor.setExpr(“%name%.BORDER_NONE"” );
borderNoneActionDescriptorr.setShortDescription(“BORDER_NONE
 ”);

//ActionDescriptors are provided in a Vector, you can add as
 many as you want per method
//In this example, we only provide one
java.util.Vector borderActions = new java.util.Vector();

borderActions.addElement(borderNoneActionDescriptor);

//Set the MethodDescriptor attribute for actions
methodDescriptor.setValue(MethodDescriptorAttributes.ACTION_
 ATTRIBUTE,borderActions);
```

III

Professional Features

CHAPTER 11

Creating Native Win32 Java Applications

This chapter provides an overview of the steps you must follow to create native Win32 Java applications and DLLs or convert existing bytecode applications to native code; things to remember while working with native applications; and an example of creating a simple native executable with a DLL.

If you have the Visual Cafe Professional Edition or Visual Cafe Database Edition, then you can make use of these native features.

About native Win32 applications

Visual Cafe lets you create native Win32 Java applications that run without using the Java Virtual Machine. Native Win32 Java applications offer the following advantages over bytecode Java applications:

- Speed – Because you don't need to run native Win32 applications using a Java Virtual Machine, applications will run faster.
- Packaging – Unlike bytecode Java applications, in which all the class files must be available in order to run the application, Native Win32 Java projects can create an executable (`.exe`) file that contains all the class information.
- Compatibility with existing executable and Dynamic Link Library (DLL) files – You can include code already written in C or C++ in your native Win32 Java applications with minor modifications.

  A **Dynamic Link Library** (**DLL**) is a library that is linked to programs when they are loaded or run, rather than as the final phase of compilation.

- Portability – Applications written for native Win32 still generate all the class files necessary to port the Java code to another platform.

Native libraries and DLLs included with Visual Cafe

Visual Cafe includes libraries and DLLs to support native application and DLL development.

Support for native development includes several Sun and Symantec Java API packages that are already converted to DLLs.

To find out what classes are specifically contained in each DLL, see `packlst.txt` in the `VisualCafe\Redist` folder. This file will always have the most current information regarding the classes stored in each DLL.

Creating native executables and DLLs

The process of creating a native executable or DLL file is very similar to the process used to create bytecode Java applications. In general, the development cycle consists of these basic steps:

1. Create a new project.

   To get started quickly, you can use the Win32 AWT Application, Win32 Console Application, or Win32 Dynamic Link Library project template.

   For more information, see “About files in a project” on page 3-42.
2. Design the user interface.

   See Chapter 7, “Working with Components.”
3. Enhance the project’s Java source code.

   See Chapter 4, “Working with Source Code.”
4. Set the project options.

   See “Setting project options for native programs” on page 11-7.
5. Test run the application or DLL in Visual Cafe.

   See Chapter 5, “Compiling and Deploying Your Project.”
6. Debug the application or DLL, if needed.

See "Debugging native programs" on page 11-6.

7 Test run the application or DLL outside of Visual Cafe.

8 Deploy the application or DLL.

   See Chapter 5, "Compiling and Deploying Your Project."

9 (Optional) Register the DLL using `SNJREG.EXE`, if necessary.

   See "Registering DLLs using SNJREG" on page 11-19.

You can see an example of creating a simple native executable with a DLL can be found in the section "Example: Creating an executable file" on page 11-22.

Considerations when creating native Win32 Java applications

When you're developing native Win32 Java applications, there are several things you can do to minimize build errors. Keep the following considerations in mind when you're creating native Win32 applications:

- Keep all the files in a single project. Build any DLLs as subprojects that you add to the main project.
- Use `SNJREG` to register any Java DLLs you create.
- When you're compiling DLLs in the command line, use the `-export` option.
- Remember that `SNJRT11.LIB` is automatically linked to your project.
- During run time of the executable, make sure that the path includes the correct folder that contains the DLLs.
- Statically linked DLLs have to be in the Windows path. For an example, see the `staticDLL` sample in the `Samples\Symantec\Tutorials\Win32` folder.
- Before you run an application that dynamically loads classes during run time, you must register the DLLs using `SNJREG`. For example, if you use `Class.forName(className)` to access a class in a DLL, you need to register the DLL. The `dynamicDLL` sample in the `Samples\Symantec\Tutorials\Win32` folder illustrates this.
- If you link object files that contain static members to different library or DLL files in a project, be aware that there will then be more than one copy of the static member.

- Do not include any library files in the Files tab of the Project window. Include library files in a project by using the Compiler tab in the Project Options dialog box. For more information, see "Including library files to link into your native program" on page 11-14.

In addition to the considerations listed above, you must keep in mind that native Win32 Java application development uses a link step after compiling and that the main class is treated differently in native Win32 development than in bytecode development.

Linking native Win32 applications

For a native project to link, you may have to perform extra steps that you don't need to perform for a bytecode project. In bytecode programs, Java uses the classpath to dynamically and automatically resolve references to classes. However, in native applications the Java classpath is not used to resolve these references. They must be resolved by *static linking*, which means they are resolved when the application is built.

Resolution by static linking occurs in the traditional compile/link mechanism. The object (`.obj`) files, which are generated by the SJ compiler for each Java class, are linked together by the *linker*, which resolves the references. Visual Cafe supplies the linker with the object files and libraries necessary to build your native Java executable. Linker errors will occur if the linker was unable to resolve a reference made to a class in your project. Typically, this means an object file that corresponds to a class that your project uses didn't get passed to the linker by the project system. Standard Java run-time libraries and Visual Cafe components are automatically passed to the linker by Visual Cafe.

Visual Cafe may report "symbol undefined" errors when you build your project. These errors will be reported in the Messages window. If this happens, you need to get these symbols resolved by making them known to the linker. The simplest way to do this is to add the Java source file for any missing classes to your project. Other possible techniques include adding appropriate object files or import libraries to your project. Any symbols in source files, object files, or import libraries that are in your project will be passed to the linker and used to resolve references when you build an application. Various ways to do this are discussed in the linking examples in the `VisualCafe\Samples` folder.

As with bytecode programs, native Java applications allow programs to dynamically load arbitrary classes at run time, without requiring them to be statically linked. However, classes that are not statically linked into your application at build time must exist in a separately compiled DLL and be registered in the Windows registry using the SNJREG tool. For more information, see "Registering DLLs using SNJREG" on page 11-19.

The main class in bytecode and native applications

The *main class* is the name of the class that contains the `main` method. For both a bytecode Java application and a native Win32 Java application, the main method in the main class is the starting point of execution. However, you should note these differences between bytecode and native applications:

- When you run a bytecode application from the command line, you type the name of the Java file that contains the main method as the first argument to `Java.exe`. For a native Win32 application, you run the application outside the Visual Cafe environment as you would any other executable and use the application name, not the main class name. See "Specifying a program for running and debugging a DLL" on page 11-9 and "Specifying the name of a native application or DLL" on page 11-7 for more information.
- An application must have a main class containing a main method with this signature:
  ```
  static public void main(String args[])
  ```

If a bytecode application does not have a method of this format, the application can compile but will not run. If a native application doesn't have a method of this format, the application cannot link or run.

Deploying native Win32 applications, DLLs and libraries

When you deploy your native Win32 application, you must include the DLLs the program requires. Visual Cafe comes with the standard DLLs you need to run your Java programs. The redistributable Visual Cafe DLLs are in the `VisualCafe\Redist` folder.

In addition to Visual Cafe's standard DLLs, you need any DLLs you created for your program. To create a program that uses your custom DLL, you need to provide the class files, the library file (which is created when the DLL is created), and the DLL. The class files must be on the classpath and the library should be specified in the compiler options for the project. For more information, see “Registering DLLs using SNJREG” on page 11-19 and “Setting compiler options” on page 5-57.

You need to register any dynamically loaded DLLs on the computer where the executable will run or the DLL is used. To do so, you can use the `SNJREG` tool. See “Registering DLLs using SNJREG” on page 11-19 for more information.

In order to launch a Java Win32 application on a machine that doesn’t have Visual Cafe installed, you’ll need to run `snjrt20.exe`, which is in the `VisualCafe\Bin` folder. This will correctly deploy your properties files and any other files from Visual Cafe that your application needs to run. When deploying your program, you can include the `snjrt20.exe` file for redistribution. If you’re debugging on the target machine, you’ll want to use `snjrt20d.exe`. The only difference between `snjrt20d.exe` and `snjrt20.exe` is that `snjrt20d.exe` contains debug information as well.

Note: You only need `snjrmiregistry.exe` if you have a remote method invocation (RMI) executable that is like one of the samples, which does not start the naming server itself (the other RMI sample does not require `snjrmiregistry.exe`).

Debugging native programs

In Visual Cafe, you can debug native Win32 applications and DLLs, just as you would debug bytecode programs (if you have the Visual Cafe Professional Edition or Visual Cafe Database Edition). For DLLs, you need to specify the calling program you want to use to run and debug your DLL. See “Creating native executables and DLLs” on page 11-2 for more information.

Note the following differences between debugging native and bytecode programs:

- The Calls and Threads windows are different for native and bytecode programs.
- When you’re debugging native Win32 code while incremental debugging is enabled, you can add new methods, while with bytecode

you cannot. (For more information, see “Using incremental debugging” on page 6-40.)

- You cannot perform remote debugging with native programs.

Setting project options for native programs

When you create a native Win32 Java executable or DLL, you need to set the project options so Visual Cafe recognizes that you’re developing native Win32 Java applications. You also need to set options to tell Visual Cafe to debug a native Win32 Java application.

This section provides information on setting project options for building native applications. For information on general project options, see Chapter 3, “Working with Projects.”

Specifying the name of a native application or DLL

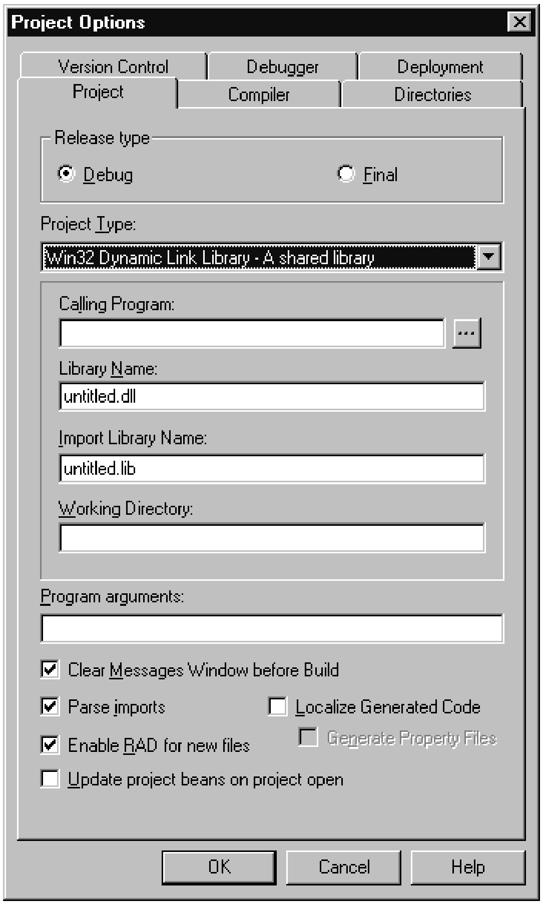
Visual Cafe provides a default file name of `untitled` to executable and DLL files. The default file name is `untitled.exe` or `untitled.dll`. You can change the name from the default by performing the following steps.

To specify the name of a native executable or library:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.

3 In the Project Options dialog box, click the Project tab.


Project Options

Version Control | Debugger | Deployment | Project | Compiler | Directories

Release type: (● Debug) ○ Final

Project Type: Win32 Dynamic Link Library - A shared library

Calling Program:

Library Name: untitled.dll

Import Library Name: untitled.lib

Working Directory:

Program arguments:

☑ Clear Messages Window before Build
☑ Parse imports
☐ Localize Generated Code
☑ Enable RAD for new files
☐ Generate Property Files
☐ Update project beans on project open

4 Set the Project Type field to Win32 Application or Win32 Dynamic Link Library.

5 If you've chosen a DLL project type, enter the library file name in the Library Name field.

If you've chosen a Win32 application project type, enter the executable file name in the Application Name field.

The application name is by default the project name, appended with the `.exe` extension.

The library name is by default the project name, appended with the `.dll` extension.

The import library name is the name of the import library file that gets created along with your DLL. It is by default the project name, appended with the `.lib` extension.

6 Click OK.

The change takes effect the next time you run your project.

Specifying the working directories for a native program

If needed, you can specify the location of a native application or the calling program used to run a native DLL. For example, you need to specify a working directory if the executable is not in the same directory you want to run it from.

To specify the working directory for a native program

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 In the Project Options dialog box, click the Project tab.

4 While Win32 Application or Dynamic Link Library is the Project Type, type the working directory you want to run the executable from.

5 Click OK.

The change takes effect the next time you run your project.

Specifying a program for running and debugging a DLL

Because DLLs are called by an executable file to run, you need to specify the name and path of the executable file so Visual Cafe can run the DLLs for testing or debugging.

To specify an executable file for running or debugging a DLL:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 In the Project Options dialog box, click the Project tab.

4 Set the Project Type field to Win32 Dynamic Link Library.

5 Type the name of the executable in the Calling Program field.

Note: In the Calling Program field, you enter the name and fully qualified path to the executable. If you do not specify a fully qualified path, Visual Cafe looks in the project directory, then through the directories in your Windows `PATH` environment variable.

6 Click OK.

The change takes effect the next time you debug the DLL.

When the calling program tries to load the DLL, it looks for the DLL in the following order:

1. The directory from which the application loaded. This is the project folder or, if you specified a full path, the folder where the program resides.
2. The current working directory, if it's different from the directory from which the application loaded.
3. For Windows 95 and 98, the Windows system directory. For Windows NT, the 32-bit Windows system directory, then the 16-bit Windows system directory.
4. The `Windows` directory.
5. The directories that are listed in the Windows PATH environment variable.

Specifying a class or package to be exported

When you build DLLs you need to tell Visual Cafe which classes or packages can be used by calling programs.

This option specifies the packages or classes that you want a DLL or native application to make available to a native application that uses it. The default is to export all packages and classes. This option is only applicable to native, Win32 programs.

For example, you might not want to export some packages and classes for these reasons:

- ◆ You want to hide functions from use.
- ◆ You can reduce the size of a DLL by not exporting everything. The more exports you have, the larger the exports table, and the larger the image. The smaller the image, the faster it can be loaded.

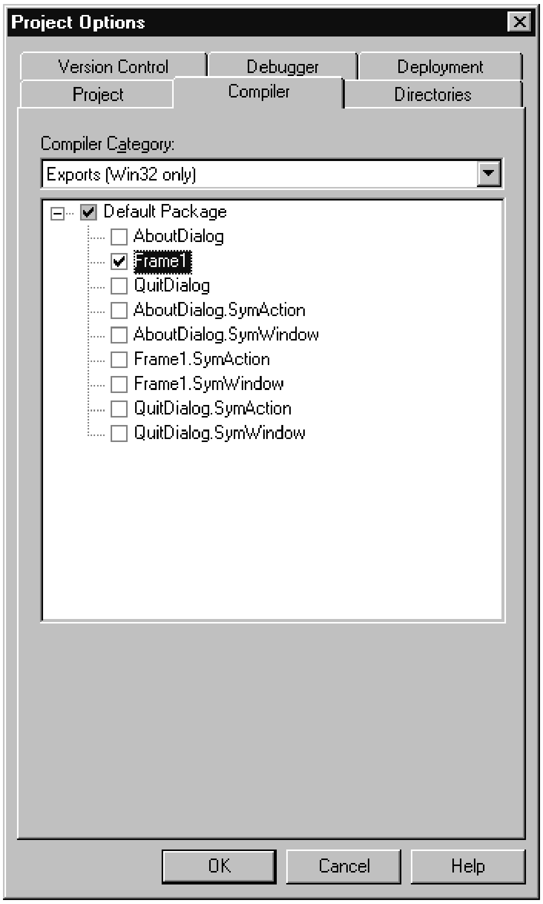
To specify a class or package to be exported:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.



Compiler Category: Exports (Win32 only)

- [✓] Default Package
  - [ ] AboutDialog
  - [✓] Frame1
  - [ ] QuitDialog
  - [ ] AboutDialog.SymAction
  - [ ] AboutDialog.SymWindow
  - [ ] Frame1.SymAction
  - [ ] Frame1.SymWindow
  - [ ] QuitDialog.SymAction
  - [ ] QuitDialog.SymWindow

4 Set the Compiler Category to Exports (Win32 only).

A list of classes and packages appears. This is the same list as the one that appears in the Packages view of the Project window.

5 Click on the classes or packages you want to make available to other projects.

A dimmed, selected box for a package means that not all classes are selected for that package.

6 Click OK.

The changes take effect the next time you build your project.

Specifying advanced Win32 compiler options

Visual Cafe provides you with additional options that allow you control aspects of how Visual Cafe compiles files.

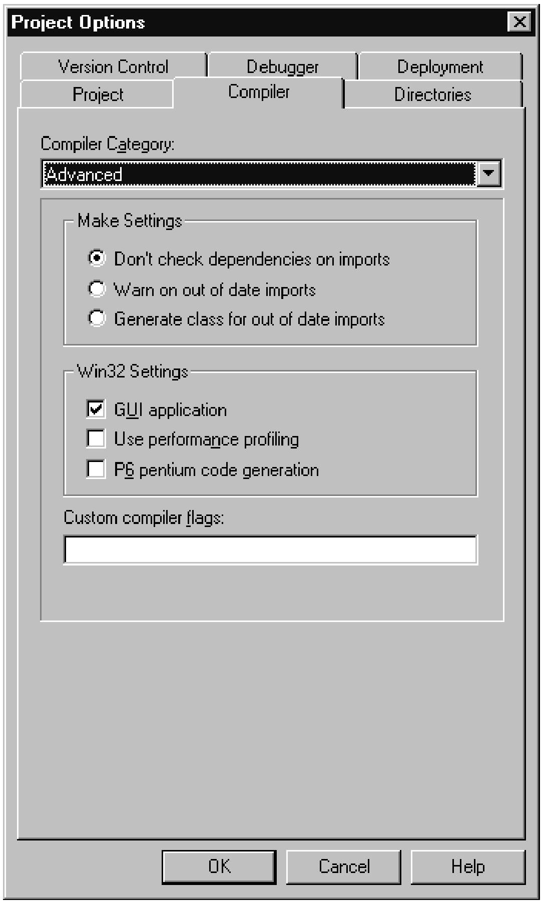
To specify advanced native compiler options:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 In the Project Options dialog box, click the Compiler tab.



Compiler Category: Advanced

Make Settings
- (●) Don't check dependencies on imports
- ( ) Warn on out of date imports
- ( ) Generate class for out of date imports

Win32 Settings
- [✓] GUI application
- [ ] Use performance profiling
- [ ] P6 pentium code generation

Custom compiler flags:

4 Set the Compiler Category to Advanced.

5 Select the checkboxes of the Win32 settings you want to select. There are three options:

Option	**Description**
GUI application	Suppresses the console window. By default this option is disabled.
Use performance profiling	Attributes the code with profiling calls so that the executable can generate profiling information. By default this option is not selected.

Option	Description
P6 Pentium code generation	Generate P6 Pentium code. The default setting is to generate P5 Pentium code.

6 Click OK.

The change takes effect the next time you build your project.

Including library files to link into your native program

In order to build the executable file and the associated DLL files, Visual Cafe needs to know the names of the import library files for the associated DLLs.

You can specify library files (`.lib`) that are linked into a native application or DLL before the default library, which is included automatically by the compiler. The libraries are specified in the order you want them to be linked. The compiler links them into your source code when you compile your native application or DLL. Visual Cafe can automatically locate libraries for you if they are in your library search path. This option is only applicable to native Win32 programs.

To specify libraries to link into your source:

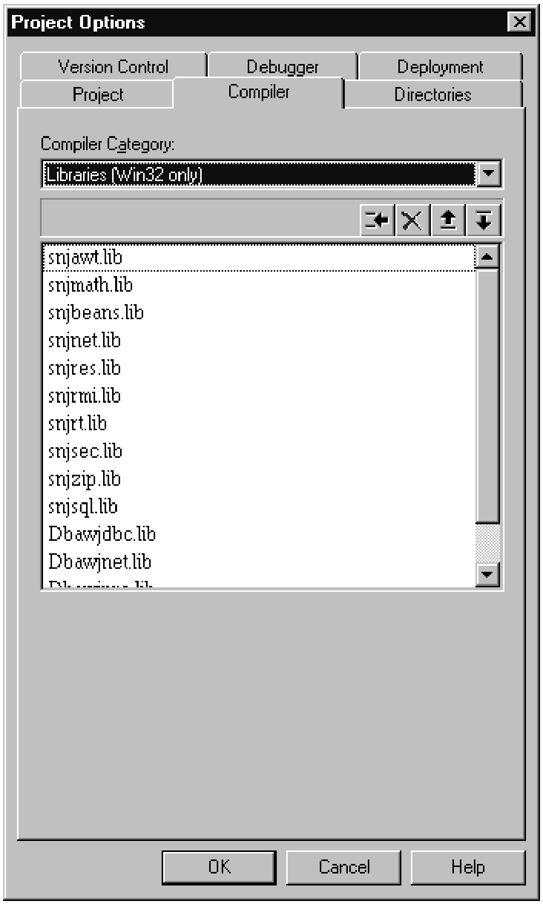
1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 Click the Compiler tab.

4 In the Compiler Category drop-down list, choose Libraries (Win32 only).



Compiler Category: Libraries (Win32 only)

snjawt.lib
snjmath.lib
snjbeans.lib
snjnet.lib
snjres.lib
snjrmi.lib
snjrt.lib
snjsec.lib
snjzip.lib
snjsql.lib
Dbawjdbc.lib
Dbawjnet.lib

5 Add libraries to the list.

- ❖ To add a library to the list, select the blank entry (marked by an empty box) at the bottom of the list and type the library name, including the full path. Or click the New button (located above the text box), then select a library by clicking the Browse (…) button that displays in the field. You can also use the New button to insert a new entry above the selected entry.
- ❖ To change the link order, select a library and move it with the Up Arrow and Down Arrow buttons.
- ❖ To delete a library from the list, select the library and click the Delete button.

6 Click OK.

The change takes effect the next time you run your project.

Making a library file available to a project

Because a library file may be in a different directory than the main project, you can specify the directory in which the library files you want to use are located. If you've already specified all the library files you're including in a project, you don't need to specify a directory.

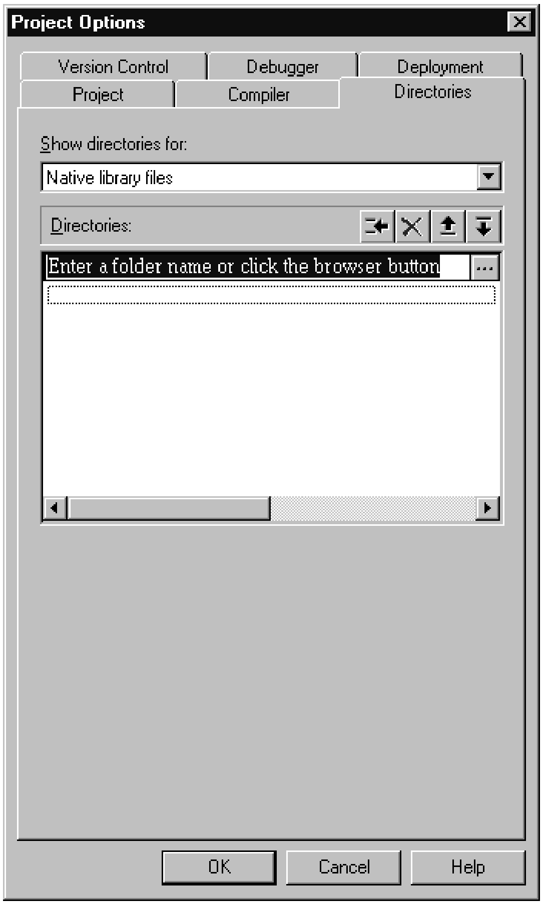
To set the directory:

1 Activate the Project window of the project you want to work with.

2 From the Project menu, choose Options.

The Project Options dialog box appears.

3 In the Project Options dialog box, click the Directories tab.



Show directories for: Native library files

Directories:

Enter a folder name or click the browser button

4 Set the Show directories for field to Library files.

5 Select a directory from the list of available directories or add the directory where the library files are located.

6 Click OK.

The change takes effect the next time you run your project.

Specifying library file search paths

If a Win32 native application or DLL uses libraries, you need to make sure Visual Cafe can find them. To do this, you can specify the search paths for libraries.

To specify library file search paths:

1. Activate the Project window of the project you want to work with.
2. From the Project menu, choose Options.

   The Project Options dialog box appears.
3. Click the Directories tab.
4. In the Show directories for field, choose Library files.

   A list of folders (directories) that can contain library files appears. The order affects the search order: the topmost folder is the first to be searched. The default is to search folders specified for the Windows LIB environment variable; these folders do not appear in the list and are last in the search order.
5. Modify the list as needed:
   - ❖ To change the order in which folders are searched, select a folder and move it with the Up Arrow and Down Arrow buttons.
   - ❖ To delete a folder from the list, select the directory and click the Delete button.
   - ❖ To add a folder to the list, select the blank entry (marked by an empty box) at the bottom of the list and type the folder name, including the full path. Or click the New button (located above the text box), then select a folder by clicking the Browse (…) button that displays in the field. You can also use the New button to insert a new entry above the selected entry.
6. Click OK.

   The changes take effect the next time you compile.

Using native command-line tools

There are a number of command-line tools that you can use with your native programs. These tools allow you to register packages in DLLs, link object files to executables, display debugging information for files, print version information, and more.

Here are the topics covered in this section:

- Registering DLLs using SNJREG
- Using OPTLINK and SMAKE with Java programs
- Displaying the contents of binary files using OPTDUMP
- Displaying the component version using Cafever
- Converting coff object files to omf using coff2omf

Registering DLLs using SNJREG

Unlike bytecode Java applications, native Win32 Java applications can't use the class path to find classes and packages. Instead, the Windows `PATH` is searched to locate classes and packages in DLLs. Classes that are dynamically loaded must have their names stored in the Symantec registry. The native classes and packages Symantec provides with Visual Cafe are already registered when Visual Cafe is installed. Some packages are already stored in the registry.

If you create a DLL whose classes or packages are not stored in the registry, the packages or classes in the DLL will not be found by your program and you'll receive an error message. `SNJREG` allows you to enter the package or class name and which `.dll` or `.exe` files contain the package or class.

`SNJREG` has the following options:

Option	Description
`-class`	DLLs register individual classes, not packages. Using `-class` will erase any previous DLL references assigned to a class that's already registered.
`-noprompt`	Disables prompting the user before making changes to existing entries
`-nowarn`	Turns off all warnings

Option	Description
`-verbose`	Reports all registrations made
`-reg` *file*`.reg`	Creates a registry file, where *file*`.reg` is the name you give the new registry file

To register a package in a DLL file using SNJREG:

1 Open a DOS window.

2 Change to the directory where the DLL files are located.

3 Enter the following:

`SNJREG [`*options*`]` *file1*`.dll` *file2*`.dll` *file3*`.dll`

where *options* can be any of the parameters in the previous table and *file1*`.dll`, *file2*`.dll`, and *file3*`.dll` are the names of the DLL files that contain the classes or packages that you want to register. You may specify as many DLL files as you want, delimited by a space. If the DLL files are not in the current directory, specify the DLL name by using a full path.

Using OPTLINK and SMAKE with Java programs

`OPTLINK` is a tool for linking object files into executables. The Java compiler provided with Visual Cafe compiles and links native applications for you automatically. You need to use `OPTLINK` only if you need more flexibility while linking your programs.

`SMAKE` is a make tool for command-line users. It's recommended that you instead use Visual Cafe projects, which are much easier to use. Only very complicated programs, such as those that include multiple languages, might require the use of makefiles.

Most Java programmers don't need to use `OPTLINK` and `SMAKE` to create native Java programs with Visual Cafe.

Keep in mind that while `OPTLINK` and `SMAKE` run under both MS-DOS and Windows 95, 98, and NT, and while `OPTLINK` can link both 16-bit and 32-bit executables, native Java applications are 32-bit Windows applications that will run only on Windows 95, 98, or NT.

For more information on using `OPTLINK` and `SMAKE`, see the Visual Cafe Online Help.

Displaying the contents of binary files using OPTDUMP

`OPTDUMP` displays the contents of `.obj` (both `.omf` and `.coff`), `.lib`, `.exe`, and `.dll` files. Any debug information in these files is also displayed. Any files of unknown type are binary dumped.

The `OPTDUMP` command has the following format:

```
OPTDUMP [ options ] infile [ outfile ]
```

Here are the options you can set for this command:

Option	Description
`/B [SeekOffset[,length]]`	Force binary dump
`/C`	Disable CodeView display
`/N`	No logo
`/P`	Enable PharLap #RVAs
`/H`	Disable `.exe` header display
`/V`	Verbose PE object display

Displaying the component version using Cafever

`Cafever` is a command-line utility that prints version information for certain components in Visual Cafe. It reads a text file `Cafever.dat` to determine which components to report version information for. This command should be invoked from the `VisualCafe\Bin` folder.

Here's the format of the `Cafever` command:

```
C:\visualcafe\bin > cafever [ component_name ]
```

Converting coff object files to omf using coff2omf

The `coff2omf` utility converts coff object (`.obj`) files to `.omf` format. Visual Cafe uses omf for the format of object files. Some other vendors use `.coff` as their format for `.obj` files.

Here's the format of the `coff2omf` command:

`coff2omf` *filenames*`.obj`

It converts the file(s) in place. The `.obj` extension is optional.

Working with samples of native applications

To illustrate how to create native Win32 Java applications, Symantec has provided several sample applications. These simple applications demonstrate:

- how to create an `.exe` file
- how to create an `.exe` file and DLLs
- registering a DLL
- working with the RMI register
- resource binding
- working with C code and JNI

The samples are located in the `Samples\Symantec\Tutorials\Win32` folder. The following examples use the `*.java` files found in the sample called `Exe`.

Note: If you use any of the samples, copy them to a new folder before making any changes to the code or the project options.

Example: Creating an executable file

Creating a native Win32 Java application is very similar to creating a bytecode application. The example below builds a simple Win32 application that prints the word `Hello`.

To create a Win32 executable file:

1 Create a new project by choosing New project from the File menu and selecting Win32 console application.

2 Create a source file called `Main.java` that contains the following code:

```
public class Main
{
    public static void main(String args[] )
    {
        Hello hi = new Hello();
        hi.printHello();
    }
}
```

3 Add a second source file called `Hello.java` with the following code:

```
public class Hello
{
    public void printHello()
    {
        System.out.println("Hello");
    }
}
```

4 Delete the file `SimplCon.java.`

5 From the Project menu, choose Options. In the Project Options dialog box, type `Main` in the Main Class field, and type `simple.exe` in the Application Name field.

6 Compile and execute the application by choosing Execute from the Project menu.

7 Save the project to new folder called `Simple`.

When you compile a native Win32 executable, the `.java` files are compiled into both class files and object files. Next, the object files are linked together to create an `.exe` file.

Example: Creating an executable that uses a DLL

Let's take this example further by building an executable and a DLL file. The two files `Hello.java` and `Main.java` stay the same.

To create an executable and a DLL file:

1 Save the `Simple` project to a new name and folder, both called `Simple3`, and remove all files with the name `Hello` from the `Simple3` folder.

2 Create a project for a DLL by choosing New Project from the File menu. Select Win32 Dynamic Link Library as the project type.

3 Add `Hello.java from the Simple` folder to the project, and delete `SimplDLL.java`.

Note: If you were creating a DLL project from scratch, you would not delete the `SimplDLL.java` file, but use it as a template.

---

4 Save the project to a new folder called `Simple2`.

5 Set the project options for the DLL by choosing Options from the Project menu and using the Project Options dialog box to select `Simple3.exe` as the calling program, `Hello.dll` as Library Name, and `Hello.lib` as Import Library name. See “Setting project options for native programs” on page 11-7 for more information.

6 Make the `Hello` class exportable. In the Compiler tab of the Project Options dialog box, select Exports, and then add the path and file name of the class you want to export.

See “Specifying a class or package to be exported” on page 11-10 for more information.

7 Build the project to generate `Hello.lib` and `Hello.DLL`.

After creating the DLL, you need to add the `Simple2` project as a subproject to the main project, `Simple3`.

To add the DLL to the Simple3 project:

1 Create a new Win32 console application project.

2 Delete all files from the project and add `Main.java`.

3 From the Project menu, choose Options. In the Project Options dialog box, type `Main` in the main class field, and type `simple.exe` in the Application Name field.

4 From the Insert menu, choose Files into Project. Click the Files of Type drop-down list box, and select Project Files (*.vep).

5 Select `Simple2.vep` and click Add.

6 Click OK to add the `.vep` file to the project.

7 Save the project as `Simple3` in a new folder called `Simple3`.

8 Make `Hello.lib` a recognized library file. In the Compiler tab of the Project Options dialog box, select Libraries, and then add the path and file name of the specified DLL.

   See “Including library files to link into your native program” on page 11-14 for more information.

9 Build the `Simple3` project and execute.

What happens in this example is similar to building an `.exe` with no DLL file. The major differences are:

- More project options need to be set
- The build order of the subproject and project must be considered when a DLL file is linked with an `.exe` file
- `.lib` files are linked to the `.exe` instead of linking multiple object files to form an `.exe`.

CHAPTER 12

Using Version Control with Visual Cafe

You can now easily integrate your version control system with Visual Cafe Professional or Database Editions. You can use third-party version control software to help you manage your programming files by keeping versions safe from unauthorized changes and controlling how many people can work on a file at one time.

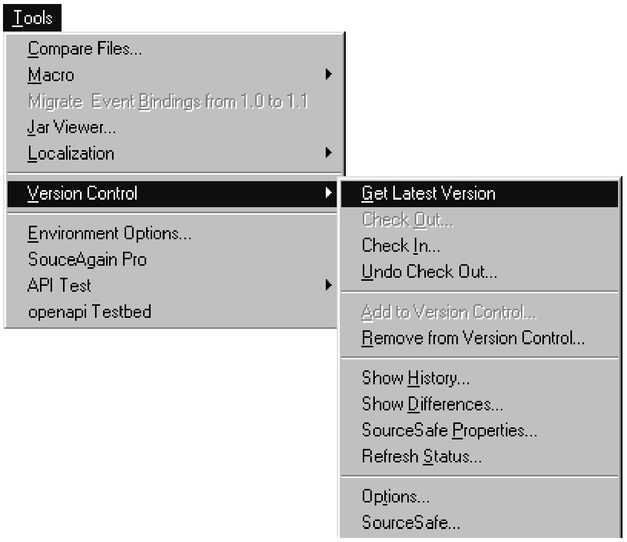
About version control

You can use version control system (VCS) software that integrates into the Visual Cafe environment. Visual Cafe supports two interfaces to version control systems:

- The **Visual Cafe Version Control** interface lets vendors create a plug-in that integrates their version control system into Visual Cafe.
- The **Microsoft Source Code Control (SCC)** interface is supported in Visual Cafe. This allows Visual Cafe to work with most version control software that uses the SCC interface.

When you integrate Visual Cafe with your version control system, you can then access a number of version control tasks directly from within the

Visual Cafe environment. All commands are accessed from the Tools menu's Version Control submenu, as shown here:



The list of items you see in the Version Control submenu depends on what features your version control system supports; you'll see fairly consistent options when you're using version control programs that use the SCC interface; they may vary more when you're using other version control programs.

Installing version control systems

You can install your version control system in the normal manner, making sure you install the software for the interface you want to use, either the Visual Cafe Version Control interface or the SCC interface. See "Configuring version control" on page 12-5 for information.

Important: When you install your version control system, you might have different install options for the SCC interface and the Visual Cafe Version Control interface. If you want to use the SCC interface, you should choose that version control install option (which might be listed as Microsoft Developer Studio support) and install the Visual Cafe SCC Bridge. If you want to use the Visual Cafe Version Control interface, choose that version control install option, which might be listed as a Visual Cafe install option.

Some vendors that have implemented the Visual Cafe Version Control interface are:

- StarBase Versions
- MKS Source Integrity

Visual Cafe should integrate with version control software that uses the SCC interface. The following software has been tested to work with Visual Cafe:

- Intersolv PVCS Version Manager version 5.3
- Microsoft Visual SourceSafe (VSS) versions 4.0 and 5.0
- StarBase Versions version 2.0
- MKS Source Integrity version 7.3
- Rational ClearCase version 3.2

In Visual Cafe, it's possible to open more than one project that uses the same SCC version control provider, or projects that use different SCC version control providers.

Enabling version control for a project

You enable version control software on a per-project basis. If you've properly installed version control software on your computer, you'll be able to select it in Visual Cafe's project options. Version control systems that use the SCC interface are prefaced with the words "SCC Provider" in the Project Options dialog box.

Visual Cafe's integration with version control systems is designed to be flexible. While using version control within Visual Cafe, you can open several projects at once and:

- use the same version control system for different projects
- use different version control systems for different projects
- use version control with some projects and not others

Managing projects using the SCC interface

You can add the Visual Cafe project file (`.vep`, not `.vpj`, `.ve2`, or `.cdb`) and any file in the project, including Java and HTML files, to the version control system.

Before you make changes to your project file (such as adding files to or removing files from the project), you should first check it out of the version control system (see "Checking files in and out" on page 12-13). If the project file on your local computer is different from the project file in the version control system and you attempt to check out the project file or get the latest version of it, Visual Cafe does a project merge to update your local version of the project. You are prompted before changes are made to your local copy. If you do not accept changes, you cannot check out the project file or get the latest version.

Visual Cafe synchronizes with the version control system after any significant version control operation. See "Refreshing file status" on page 12-18 for details.

Note: If you want to move a Visual Cafe project and the files it contains to another location, make sure no files are checked out before moving the files (see "Checking files in and out" on page 12-13).

Using version control

You can access version control commands by choosing Version Control from the Tools menu, then choosing various items from the submenu. What specific items are visible in the submenu depends on what features your version control system supports.

You can perform a variety of tasks with version control while within the Visual Cafe environment. These tasks include:

- Configuring version control
- Setting version control options
- Adding and removing files for version control
- Checking files in and out of version control

- Getting the latest version of a file
- Refreshing file status in version control
- Showing the version control history of files
- Showing the differences between files in version control
- Showing version control properties for files
- Running the version control system

Each of these tasks is discussed below.

Configuring version control

In order to use version control in conjunction with Visual Cafe, you need to configure Visual Cafe for your specific version control system or systems that you have installed.

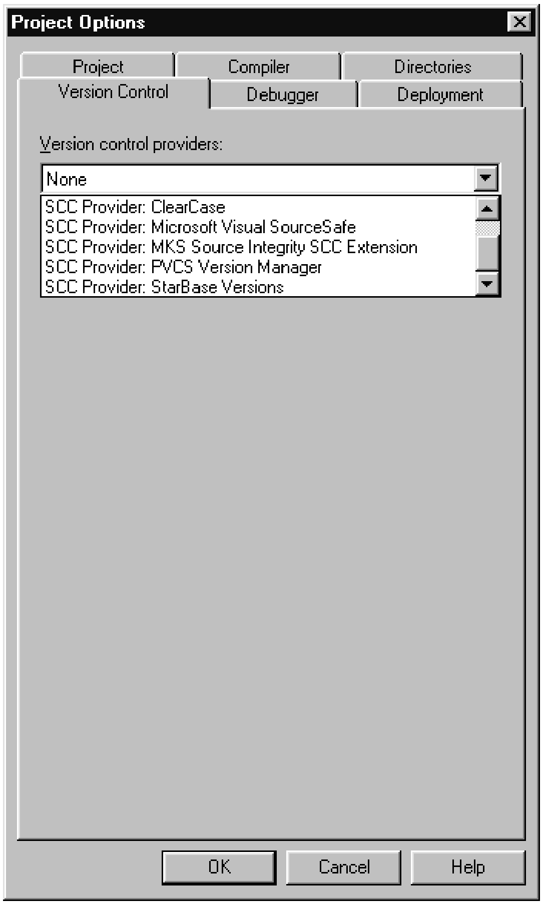
To set up version control for use in the Visual Cafe environment:

1. Activate the Project window of the project you want to work with.

   Note: After creating a new project, you should save the project to the location where you want it before choosing a version control system.

2. Choose Options from the Project menu.

3 In the Project Options dialog box, click the Version Control tab.



Version control providers:

None

SCC Provider: ClearCase
SCC Provider: Microsoft Visual SourceSafe
SCC Provider: MKS Source Integrity SCC Extension
SCC Provider: PVCS Version Manager
SCC Provider: StarBase Versions

4 Choose the version control software you want to use.

Remember that you must properly install the software before you can use it with Visual Cafe. If you want to use the SCC interface, choose a version control system that is prefixed with "SCC Provider." For example, you might see the line SCC Provider: StarBase Versions; whereas, if StarBase does not use the SCC interface, the line would show StarBase Versions.

5 Click OK.

The change takes effect immediately.

If you're using the SCC interface, the Version Control Options dialog box appears. The version control system's provider may prompt you to create a new project or open an existing project, and then the Add to Version Control dialog box appears.

Tip: If you're using the SCC interface, choose Version Control from the Tools menu to view the version control menu items available to you. For example, you can set version control options and check files in and out (see "Checking files in and out" on page 12-13).

Note: If you enable a version control system or switch version control systems for a project, the files are checked into the new version control system as new files.

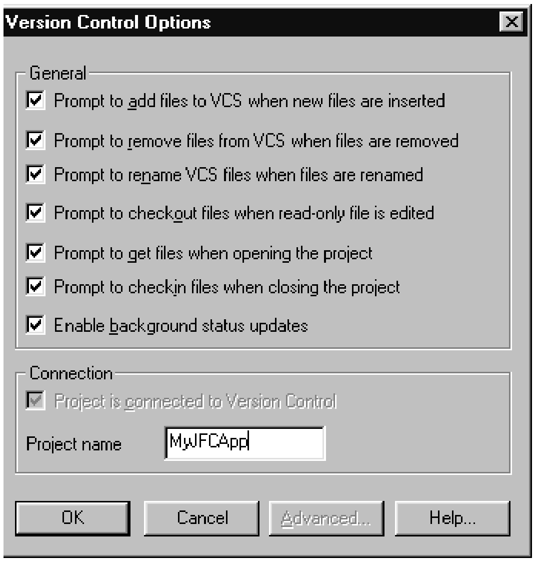
Setting version control options

If you're using the SCC interface, you can set version control options for the project after you enable version control.

To set general version control options:

1. Activate the Project window of the project you want to work with.
2. From the Tools menu, choose Version Control, and then choose Options.

The Version Control Options dialog box appears.



[✓] Prompt to add files to VCS when new files are inserted
[✓] Prompt to remove files from VCS when files are removed
[✓] Prompt to rename VCS files when files are renamed
[✓] Prompt to checkout files when read-only file is edited
[✓] Prompt to get files when opening the project
[✓] Prompt to checkin files when closing the project
[✓] Enable background status updates
[✓] Project is connected to Version Control
Project name: MyJFCApp

Note: When you first establish the connection between Visual Cafe and your version control software, the Version Control Options dialog box will automatically appear.

---

3 In the Version Control Options dialog box, select the options you want:

Select...	**To specify this...**
Prompt to add files to VCS when new files are inserted	If selected, when you add one or more new files to a project you're given the ability to add files (that are new to the project) to the version control system. If deselected, new files are not added to version control. (If you don't select this option, you can later add the files to the version control system by choosing Version Control from the Tools and then Add from the submenu). This option is selected by default.

Select...	To specify this...
Prompt to remove files from VCS when files are removed	If selected, when you remove one or more files from a project you're asked if you want to remove them from the version control system. If deselected, the files are not removed. (If you don't select this option, you can later remove the files from the version control system by choosing Version Control from the Tools menu, and then Remove from the submenu.) This option is selected by default.
Prompt to rename VCS files when files are renamed	If selected, when you rename a file in a project you're asked if you want to rename it in the version control system. Alternatively, if your version control system does not support file renaming, you're asked if you want to add the file to the version control system under the new name, then if you want to remove the file under the old name. In this case, you lose some history information about this file. If deselected, you are not prompted and the file is not renamed or removed from the version control system. Be aware that this can create complications by creating a discrepancy between your local files and the files in the version control system. (If you don't select this option, you'll have to go into the version control system to rename files from there.) This option is selected by default.
Prompt to check out files when read-only file is edited	If selected, when you try to edit a read-only file in version control in a project you're asked if you want to check it out of the version control system. If deselected, you are not prompted and the file is not checked out. (If you don't select this option, you can later choose Version Control from the Tools menu, and then Checkout from the submenu.) This option is selected by default.
Prompt to get files when opening the project	If selected, when you open a project you're asked if you want to get the latest version of files from the version control system. If deselected, you do not get the latest versions. (If you don't select this option, you can later choose Version Control from the Tools menu, and then Get from the submenu.) This option is selected by default.
Prompt to check in files when closing the project	If selected, when you close a project you are asked if you want to first check in files that you had checked out of the version control system. If deselected, the files are not checked in. (If you don't select this option, you can later go into the version control system and check in the files there.) This option is selected by default.

Select...	To specify this...
Enable background updates	If selected, your version control system performs background status updates. If deselected, there are no background updates. How background status updates are implemented depends on your version control system. This option is selected by default.

4 Click OK.

The change takes effect immediately.

Adding and removing files

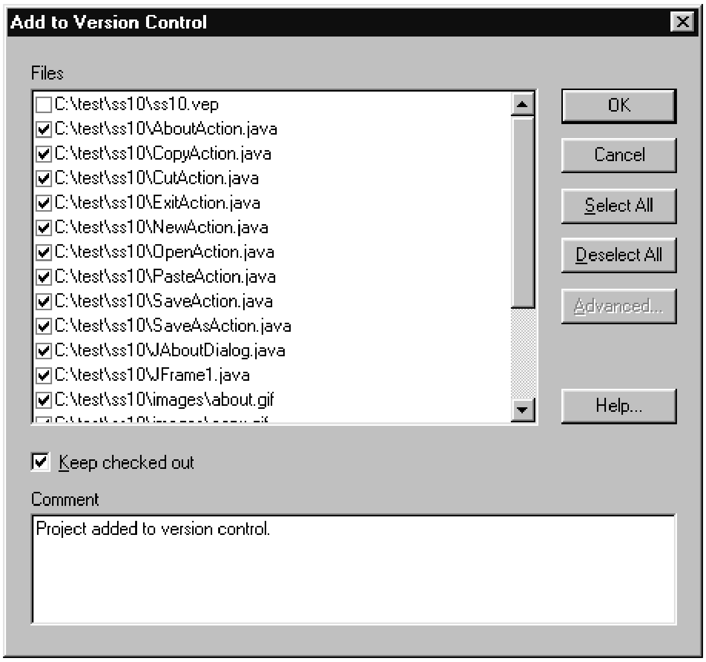
If you're using the SCC interface, within Visual Cafe you can add files to or remove files from the version control system. When you first choose a version control system for a project, you're asked what files you want to add. In addition, depending on your version control options, you may also be prompted to add or remove files at other times. See "Setting version control options" on page 12-7 for more information.

To add one or more files to version control:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Add to Version Control from the submenu.

The Add to Version Control dialog box appears.



Add to Version Control

Files

- [ ] C:\test\ss10\ss10.vep
- [x] C:\test\ss10\AboutAction.java
- [x] C:\test\ss10\CopyAction.java
- [x] C:\test\ss10\CutAction.java
- [x] C:\test\ss10\ExitAction.java
- [x] C:\test\ss10\NewAction.java
- [x] C:\test\ss10\OpenAction.java
- [x] C:\test\ss10\PasteAction.java
- [x] C:\test\ss10\SaveAction.java
- [x] C:\test\ss10\SaveAsAction.java
- [x] C:\test\ss10\JAboutDialog.java
- [x] C:\test\ss10\JFrame1.java
- [x] C:\test\ss10\images\about.gif

- [x] Keep checked out

Comment

Project added to version control.

(This dialog box can also appear at other times, such as when you choose a version control system for a project.)

The dialog box lists the files in your project that have not already been added to the version control system by you or another user; if the project file (`.vep`) has not been added, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files you want to add:

- ❖ To select individual files, click a file so the checkbox is selected.
- ❖ To deselect individual files, click a file so the checkbox is deselected.
- ❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.

- ❖ To add files and check them out, select Keep Checked Out. To add files without checking them out, deselect the option.
- ❖ To add comments to selected files, type in the Comments field. For some version control systems, you can also add or modify comments in the version control system's dialog boxes.
- ❖ To set other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.

4 Click OK.

The selected files are added to version control.

Consult your version control software's documentation for more information about what happens next.

To remove one or more files from version control:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Remove from Version Control from the submenu.

The Remove from Version Control dialog box appears.

(This dialog box can also appear at other times, such as when you rename a file but the version control system does not support a rename operation.)

The dialog box lists the files in your project that have already been added to the version control system by you or another user; if the project file (`.vep`) has been added, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files you want to remove:

- ❖ To select individual files, click a file so the checkbox is selected.
- ❖ To deselect individual files, click a file so the checkbox is deselected.
- ❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.
- ❖ To set other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.

4 Click OK.

The selected files are removed from version control.

Consult your version control software's documentation for more information about what happens next.

Checking files in and out

If you're using the SCC interface, within Visual Cafe you can check files in and out of a version control system. You can also undo a file checkout so the file is not modified and you no longer have it checked out. Depending on your version control options, you may also be prompted to check files in or out at other times. See "Setting version control options" on page 12-7 for more information.

To check out one or more files in a project:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Check Out from the submenu.

The Check Out File(s) dialog box appears. It lists the files in your project that aren't checked out; if the project file (`.vep`) is not checked out, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files you want to check out:

❖ To select individual files, click a file so the checkbox is selected.

❖ To deselect individual files, click a file so the checkbox is deselected.

❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.

❖ To set other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.

4 Click OK.

The selected files are checked out.

Consult your version control software's documentation for more information about what happens next.

To check in one or more files:

1. Activate the project you want to work with.
2. Choose Version Control from the Tools menu, then Check In from the submenu.

   The Check Out File(s) dialog box appears. It lists the files in your project that are checked out; if the project file (`.vep`) is checked out, it is also listed.

   Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

   ---

3. Select the files you want to check in:
   - ❖ To select individual files, click a file so the checkbox is selected. To deselect individual files, click a file so the checkbox is deselected.
   - ❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.
   - ❖ To check in file changes but keep files checked out, select Keep Checked Out. To just check in files, deselect the option.
   - ❖ To add comments to selected files, type in the Comments field. For some version control systems, you can also add or modify comments in the version control system's dialog boxes.
   - ❖ To set other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.
   - ❖ To examine differences between one of your files and a file in the version control system, click Differences. The dialog box is implemented by your version control system, not Visual Cafe.
4. Click OK.

   The selected files are checked in.

   Consult your version control software's documentation for more information about what happens next.

To undo one or more file checkouts:

1. Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Undo Checkout from the submenu.

   The Undo Checkout dialog box appears. It lists the files in your project that are checked out; if you checked in the project file (`.vep`), it is also listed.

   Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files for which you want to undo the checkout:

   - ❖ To select individual files, click a file so the checkbox is selected. To deselect individual files, click a file so the checkbox is deselected.
   - ❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.
   - ❖ To specify other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.

4 Click OK.

   The selected files are no longer checked out, and they behave like checked-in files.

Working with the Visual Cafe project file and version control

You can now merge project file changes directly into your version control system while working in Visual Cafe, rather than having to exit Visual Cafe and check out your files from version control.

The Visual Cafe project file (`*.vep`) is a special case, since it's not a text file like most files under version control, such as `*.java`, `*.html`, and so on, and also because it is locked by Visual Cafe. However, unlike most binary file types, merging two project files can make sense.

After the project file has been checked in, it may be made read-only by the version control system. When it's checked out or when the latest version of the project file is retrieved, the version of the `.vep` in memory and the newly retrieved version are merged. In some cases, the project file may be automatically saved to the disk so that the options are synchronized.

Keep the following precautions in mind when you're working with a project file in a version control system:

- You should be very careful when replacing an existing, local version of the project file. This may cause new files to be added to and/or existing files to be removed from the project. The options of the new project file override the options of the existing project file. This is the expected behavior, but it can be confusing.
- You should also be very careful when checking in a project file; it may be made read-only, and this can be a source of confusing warnings for you.
- In general, it's a good idea to handle the project file independently of the other source files.

You shouldn't check in the `.vpj` and the `.ve2` files, which contain information that's generated by Visual Cafe. These files can be recreated automatically from the source files. If you try to check in these files they could become out of sync with the user's local changes.

It's possible to move a Visual Cafe project to another location after some of its files have been put under an SCC version control provider. However, it's recommended that you check in or un-check-out your files before moving them, because some version control systems will believe that you have files still checked out at the old location.

About renaming files in conjunction with version control

The SCC provider may or may not handle the rename function. If they don't, Visual Cafe will try to add the renamed file as a new file under version control, and to remove the old file from version control. You can choose whether or not to perform each of these operations.

If the SCC provider doesn't support the rename function, then you must be very careful when renaming files in Visual Cafe if you don't want to introduce breaks in the history of a source file. Visual Cafe, unlike most development environments, allows you to rename a file very easily and indirectly when renaming a class in the Objects view of the Project window. You are strongly advised to determine the definitive class or file name early in the development process, and to add the file to version control afterwards. If the SCC provider supports the rename function, then renaming a class shouldn't be a problem.

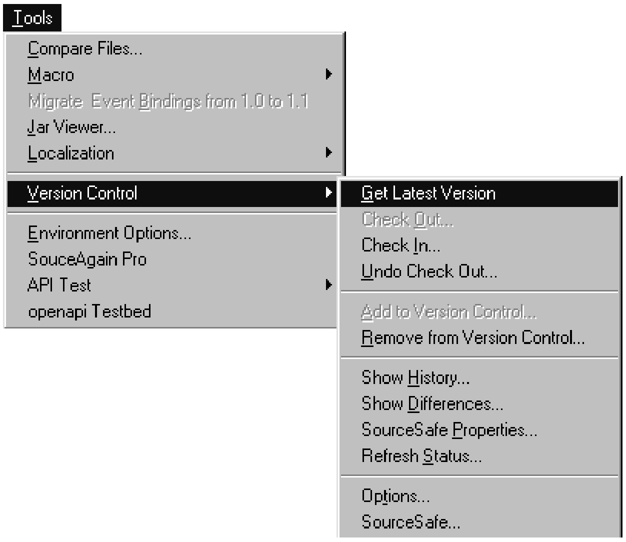
Getting the latest version of a file

If you're using the SCC interface, within Visual Cafe you can get the latest version of a file in a version control system. Depending on your version control options, you may also be prompted to get the latest version of files when opening a project. See "Setting version control options" on page 12-7 for more information.

To retrieve the latest version of a file:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Get Latest Version from the submenu.



The Get Latest Version dialog box appears. It lists the files in your project that have been added to the version control system by you or another user; if the project file (`.vep`) has been added, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files you want to get the latest version of:

- ❖ To select individual files, click a file so the checkbox is selected. To deselect individual files, click a file so the checkbox is deselected.
- ❖ To select all files in the list, click Select All. To deselect all files, click Deselect All.
- ❖ To specify other options in a version control system dialog box, click Advanced (if available). The dialog box is implemented by your version control system, not Visual Cafe.

4 Click OK.

The latest version of the file is retrieved.

Consult your version control software's documentation for more information about what happens next.

Refreshing file status

Visual Cafe synchronizes with the version control system after any significant version control operation (for example, when you use a command in the Tools menu's Version Control submenu). In addition, you can synchronize your Visual Cafe and VCS projects at any time by choosing Version Control from the Tools menu, then Refresh Status from the submenu. Visual Cafe prompts you before performing any version control operation that might affect the Visual Cafe or VCS project. The only operations you do not receive a prompt for are:

- ◆ when a VCS project is opened or closed as a result of opening or closing a Visual Cafe project
- ◆ file status inquiries

Showing the version control history of files

If you're using the SCC interface, within Visual Cafe you can obtain the version control history of files.

To show the version control history of a file:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Show History from the submenu.

The History dialog box appears. It lists the files in your project that have been added to the version control system by you or another user; if the project file (`.vep`) has been added, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3 Select the files you want to view the history of:

- ❖ To select or deselect individual files, click a file.
- ❖ To select or deselect multiple files, CTRL-click the files.

4 Click Show.

A dialog box that was implemented by your version control system appears.

5 Show the history of other files, if needed, then click Close when you're finished.

The files' version control history is displayed.

Consult your version control software's documentation for more information about what happens next.

Showing the differences between files

If you're using the SCC interface, within Visual Cafe you can show the differences between files on your local computer and files in a version control system.

To show differences between files in version control:

1 Activate the project you want to work with.

2 Choose Version Control from the Tools menu, then Show Differences from the submenu.

The Differences dialog box appears. It lists the files in your project that have been added to the version control system by you or another user; if the project file (`.vep`) has been added, it is also listed.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

3. Select the files you want to view the differences of:
   - ❖ To select or deselect individual files, click a file.
   - ❖ To select or deselect multiple files, CTRL-click the files.
4. Click Show.

   A dialog box that was implemented by your version control system appears.
5. Show the differences between other files, if needed, then click Close when you're finished.

   The files' differences are displayed.

   Consult your version control software's documentation for more information about what happens next.

Showing version control properties for files

If you're using the SCC interface, within Visual Cafe you can show the properties of files in a version control system.

To show version control properties for a file:

1. Activate the project you want to work with.
2. Choose Version Control from the Tools menu, then VCS Properties from the submenu, where *VCS* is the name of the version control system.

   The Properties dialog box appears. It lists the files in your project that have been added to the version control system by you or another user; if the project file (`.vep`) has been added, it is also listed.

   Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.
3. Select the files you want to view the properties of:
   - ❖ To select or deselect individual files, click a file.
   - ❖ To select or deselect multiple files, CONTROL-click the files.
4. Click Show.

   A dialog box that is implemented by your version control system appears.

5. Show the properties of other files, if needed, then click Close when you're finished.

   The files' version control properties are displayed.

   Consult your version control software's documentation for more information about what happens next.

Running your version control system

If you're using the SCC interface, within Visual Cafe you can launch the main interface to the version control system. This main entry point into the program is sometimes called the *front end*.

To launch the version control system:

1. Activate the project you want to work with.
2. Choose Version Control from the Tools menu, then choose the name of the version control system from the submenu.

   The version control application launches.

Note: The list of items you see in the Tools menu's Version Control submenu depends on what features your version control system supports.

Setting the default version control user name

If you're using the SCC interface, you can set the default version control user name in Visual Cafe for a number of reasons. For one thing, not all version control systems prompt for a user name. Also, the default Windows user name might be undefined (for example, under Windows 95 if you don't log in) or different from the version control user name that you use.

To specify the default version control user name:

1. From the Tools menu, choose Environment Options, then click the General tab.
2. In the Version Control User Name field, type the name you want to use with your version control system.
3. Click Apply or OK to save the change.

   The change takes effect immediately.

CHAPTER 13

Localizing Your Java Programs

After you create a Java program using Visual Cafe, you may at some point want to distribute it to users who speak a different language than you do. Visual Cafe helps you with the process of localizing your programs, providing a list of locales and a tool that simplifies the localization process. This chapter gives you the information you'll need to localize your programs.

About localization

If you want to deploy your applet or application to users who speak a different language or dialect than the one you're programming in, you'll want to localize your program so that those users can understand it. When you're changing your applet or application to be appropriate for a different country, dialect, or region, you're **localizing** your program. Primarily, this involves the time-intensive task of translating text strings, but it can involve much more, depending on the culture that you're localizing to. You may have to resize user interface elements such as buttons and text fields in order to accommodate longer or shorter text strings. In addition, you may have to rearrange components to fit the culture's sensitivities and preferences, redo graphics, and translate your program's documentation.

Visual Cafe assists you in using resource bundles to localize your Java programs. A **resource bundle** is a collection of elements that correspond to a specific geographical or cultural locale. For example, after creating a Java program, Visual Cafe can search for strings in your file, then replace them with a reference to a resource bundle. In addition, Visual Cafe can create one or more resource bundles that translators can use to apply words in different languages for different countries.

For example, this line:

```
label1.setText("hello");
```

might be replaced with this line:

```
label1.setText(resourceBundle.getString("label1_text"));
```

As a result, a resource bundle is added to the Visual Cafe project. Note that the default variable (key) names are different when Visual Cafe automatically localizes auto-generated code and when you use the Localization tool (see “Localizing individual strings with the Localization tool” on page 13-6).

At the least, you’ll need to create a resource bundle. After you’ve created a resource bundle, you’ll need to create and edit locales. **Locales** can include a language, country, and variant (which might be a dialect of a language). The default is the locale configured for your computer, such as US English. In some cases you might need to create the resource bundle for your localization team, which then creates and edits locales; alternatively, you may do the whole localization process yourself.

The translation for a word is searched for in the following order when you specify a language, country, and variant:

- The resource bundle for a language, country, and variant
- The resource bundle for the language and country
- The resource bundle for the language
- The default resource bundle

Here are some sample resource bundles:

- French language, France country, and Paris variant: `JFrame1Bundle_fr_FR_Paris.java`
- French language and France country: `JFrame1Bundle_fr_FR.java`
- French language: `JFrame1Bundle_fr.java`
- Default: `JFrame1Bundle.java`

Locale information is provided by two text files, which list the International Standards Organization (ISO) 639 and 3166 codes. The ISO 639 codes provide the standard list of languages, and the ISO 3166 codes provide the standard list of countries.

Visual Cafe can automatically generate two types of resource bundles: a Java type or a properties type.

A Java type of resource bundle contains a class definition that gets compiled into the object that is being localized. As a result, it's faster to use once the Java program is compiled and used outside the Visual Cafe environment.

Here's an example of a Java type of resource bundle called `Applet1Bundle.java`:

```
public class Applet1Bundle extends
 java.util.ListResourceBundle
{
   public Object[][] getContents()
   {
      return contents;
   }
   static final Object[][] contents =
   {
      {"label1_text", "hello"}
   };
}
```

The property type of resource bundle is a text file, and can be easier to replace during development if you want to modify the file outside the Visual Cafe environment. You must use one type for all resource bundles in a project. If you switch types during development, the necessary resource bundles are generated.

Here's an example of a properties type of resource bundle called `Applet1Bundle.properties`:

```
label1_text=hello
```

Using localization

This section describes the specifics of localizing your Java programs. You'll learn about:

- Localizing a project with the Localization tool (page 13-4)
- Localizing individual strings with the Localization tool (page 13-6)
- Localizing auto-generated code (page 13-9)
- Adding or deleting a locale (page 13-11)
- Adding information to the resource bundle for a locale (page 13-13)
- Editing a resource bundle (page 13-13)

Read on for information about each of these tasks.

If you proceed in localizing a project by choosing to either localize a project or localizing individual strings with the Localization tool, Visual Cafe will automatically localize both the code it generates automatically and the code that you write yourself.

After you've run the Localization tool at least once, or you choose the Localize Generated Code option in the Project Options dialog box, thereafter any code you generate will automatically be localized, but code that you've written yourself won't be localized. You'll need to localize yourself any code that you've written manually.

Note: After running the Localization tool, if you add an interaction that uses a string, you need to run the Localization tool again in order for that string to be localized. For more information about interactions, see Chapter 9, "Working with Events and Interactions."

Localizing a project with the Localization tool

Visual Cafe can automatically localize a whole project at once. To localize code on a per-string basis, see the next section, "Localizing individual strings with the Localization tool" on page 13-6.

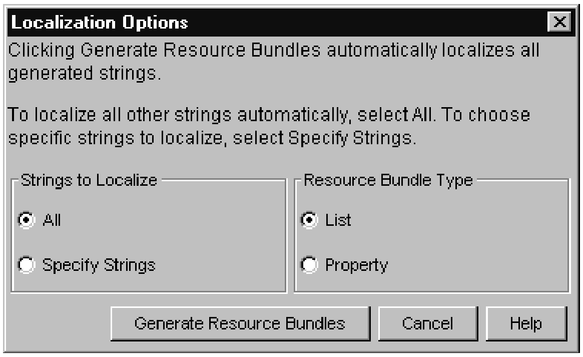
Note: The Localization tool localizes both code that you've written yourself and code that is automatically generated. Auto-generated code is delimited by `//{{INIT_CONTROLS//}}` tags. You can also choose to localize only auto-generated code; for more information, see "Localizing auto-generated code" on page 13-9.

To localize a project:

1 Open the project you want to localize.

2 From the Tools menu, select Localization, then select Resource Bundle Strings.

The Localization Options dialog box appears:



Clicking Generate Resource Bundles automatically localizes all generated strings.

To localize all other strings automatically, select All. To choose specific strings to localize, select Specify Strings.

Strings to Localize: [✓ All] Specify Strings

Resource Bundle Type: [✓ List] Property

3 In the Strings to Localize area, select All (if it's not already selected).

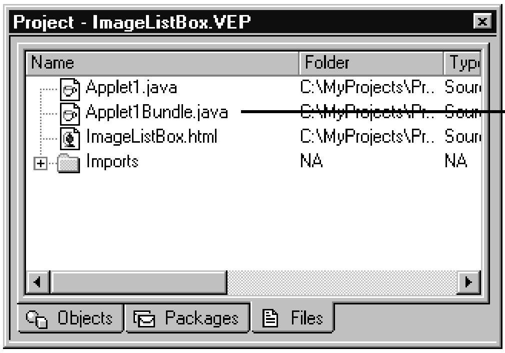
4 By default, the Resource Bundle Type is set to List. If you want Visual Cafe to generate the property type of resource bundles, select Property.

Note: After you've finished running the Localization Tool, if you change the type of resource bundle, Visual Cafe will then start to generate the new type of resource bundle.

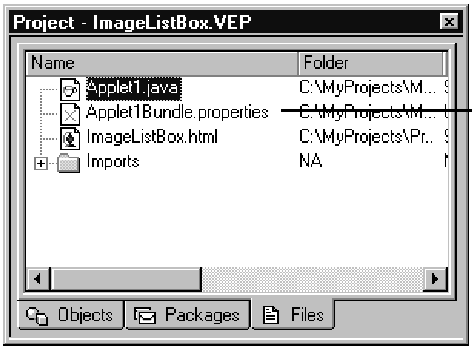
5 Click Generate Resource Bundles.

Visual Cafe localizes the project immediately and continues to localize strings that Visual Cafe automatically generates.

You can view the resource bundles in your project by looking at the Files view of the Project window:



After completing the Localization tool, if you've selected the list type of resource bundle, a Java resource bundle file appears in your project.



After completing the Localization tool, if you've selected the property type of resource bundle, a properties resource bundle file appears in your project.

If you want to localize individual lines of code, see the next section, "Localizing individual strings with the Localization tool."

Localizing individual strings with the Localization tool

You can also use the Localization tool to localize code string by string. To localize a whole project, see the previous section "Localizing a project with the Localization tool."

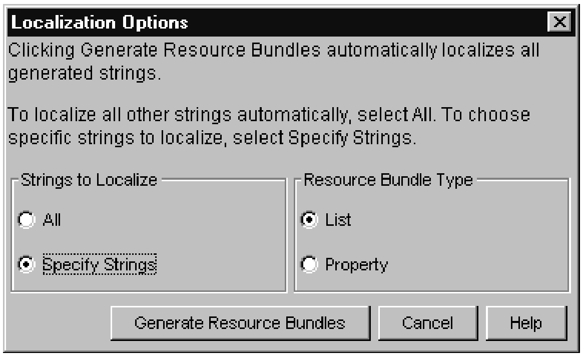
Note: The Localization tool localizes both code that you've written yourself and code that is automatically generated. Auto-generated code is delimited by `//{{INIT_CONTROLS//}}` tags. You can also choose to localize only auto-generated code; for more information, see "Localizing auto-generated code" on page 13-9.

By selecting multiple files in the Project window, you can localize your code string by string in those selected files when using the Localization tool.

To localize individual code strings with the Localization tool:

1. Open the Project window for desired project. In the Files view of the Project window, select the file or files that you want to localize.
2. From the Tools menu, select Localization, and then select Resource Bundle Strings.

   The Localization Options dialog box appears:

   Localization Options

   Clicking Generate Resource Bundles automatically localizes all generated strings.

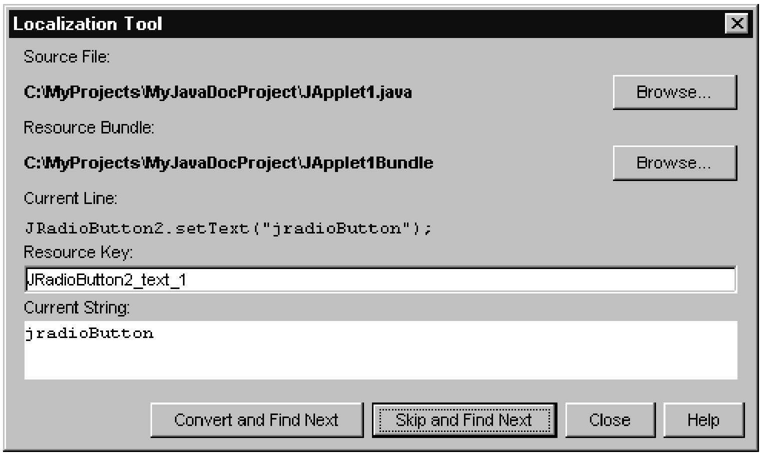
   To localize all other strings automatically, select All. To choose specific strings to localize, select Specify Strings.

   Strings to Localize: ( ) All [✓ Specify Strings]

   Resource Bundle Type: [✓ List] ( ) Property

3. In the Strings to Localize area, select Specify Strings.
4. By default, the Resource Bundle Type is set to List. If you want Visual Cafe to generate the property type of resource bundles, select Property.
5. Click Generate Resource Bundles.

The Localization tool appears.



Source File:
C:\MyProjects\MyJavaDocProject\JApplet1.java
Resource Bundle:
C:\MyProjects\MyJavaDocProject\JApplet1Bundle
Current Line:

```
JRadioButton2.setText("jradioButton");
```

Resource Key:
JRadioButton2_text_1
Current String:

```
jradioButton
```

Visual Cafe searches for strings in the file you selected, then displays the first string and the line of code it was taken from.

6 In the Localization tool, check that the source filename and resource bundle are correct. If not, click the Browse button to specify a different source file or resource bundle.

7 For each string, optionally type a variable name for the string in the Resource Key field and a default string in the Current String field, then click Convert and Find Next. To skip a string, click Skip and Find Next.

Note: If you decide to skip a string, Visual Cafe will add a `//@Skip when resource bundling` comment automatically. If you use the Localization tool again, Visual Cafe will skip the strings it skipped before.

When the entire file is localized, the Current String field displays `No Further Strings Found`.

8 To localize other files, click Browse to specify a source file. If needed, also specify a resource bundle. Then localize the files as before.

9 When you're finished using the Localization tool, click Close.

You can view the resource bundles in your project by looking at the Files view of the Project window.

Localizing auto-generated code

You can choose to have the Localization tool localize *only* the code that Visual Cafe automatically generates. Visual Cafe automatically generates code that appears between `//{{INIT_CONTROLS//}}` tags. If you choose to do this, then any code you've written yourself won't be localized.

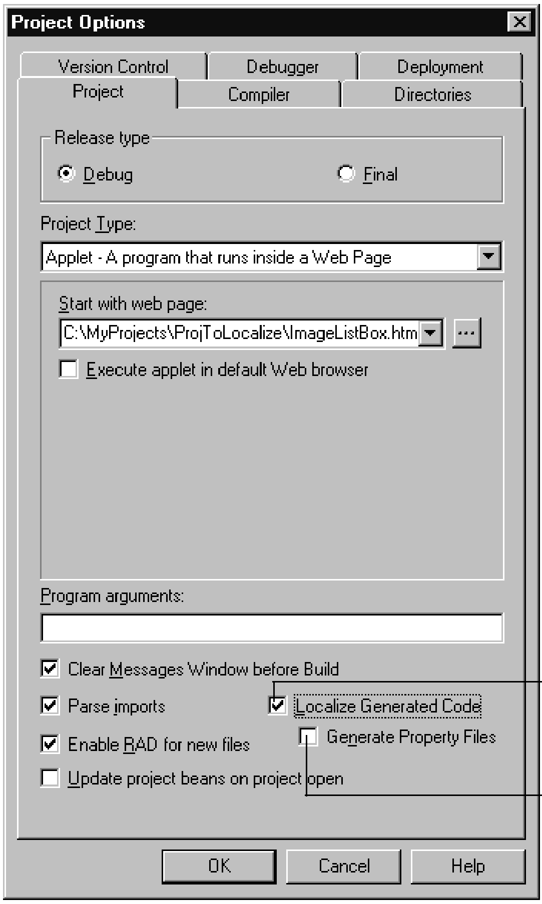
Note: If you want to localize code that you've written yourself, you can localize your whole project at once or localize individual files on a per-string basis. For more information, see "Localizing a project with the Localization tool" on page 13-4 and "Localizing individual strings with the Localization tool" on page 13-6.

To localize auto-generated code:

1. Activate the Project window of the project you want to work with.
2. Choose Options from the Project menu.

   The Project Options dialog box appears.

3 In the dialog box, click the Project tab.



Release type: [✓ Debug] Final

Project Type: Applet - A program that runs inside a Web Page

Start with web page: C:\MyProjects\ProjToLocalize\ImageListBox.htm

☐ Execute applet in default Web browser

Program arguments:

☑ Clear Messages Window before Build

☑ Parse imports

☑ Localize Generated Code

☐ Generate Property Files

☑ Enable RAD for new files

☐ Update project beans on project open

*Select this option to set Visual Cafe to localize **only** auto-generated code.*

Once you've run the Localization tool, and you deselect this option, Visual Cafe undoes the localization for auto-generated code.

Select this option to set Visual Cafe to generate the property type of resource bundle. Deselect it to generate the list type of resource bundle.

4 Select the Localize Generated Code option to specify that you want Visual Cafe to automatically localize the code it generates. Or deselect it if you want to undo localization for automatically generated code.

Note: If you deselect the Localize Generated Code option after it was selected, Visual Cafe undoes localization for auto-generated code. To fully undo localized code you'll need to also undo localized strings in manually entered code.

5 Select the Generate Property Files option to specify that you want Visual Cafe to generate the property type of resource bundles. Or deselect it generate the Java type of resource bundles.

Tip: You can also choose the type of resource bundle in the Localization tool.

6 Click OK.

If you selected the Localize Generated Code option, Visual Cafe localizes code immediately and continues to localize it as you work on your files.

Adding or deleting a locale

The default value for a resource bundle is the locale configured for your computer (US English, for example). You can add other locales in separate resource bundles, or delete locales if you want. Once you've added locales, you can choose to use one in the Visual Cafe environment.

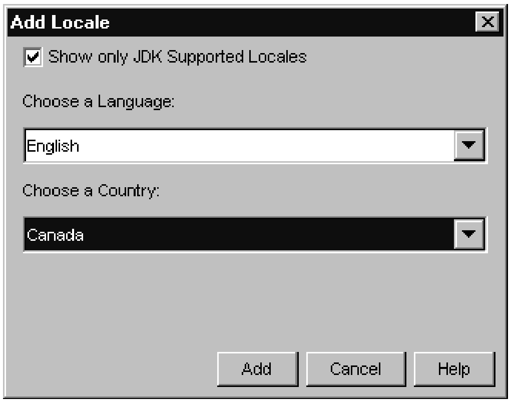
Note: You can only add a locale to a project that has been localized.

To add a locale:

1 Open the project you want to add a locale to.

2 Choose Localization from the Tools menu, then choose Add Locale from the submenu.

The Add Locale dialog box appears.



Add Locale

[✓] Show only JDK Supported Locales

Choose a Language: English

Choose a Country: Canada

3. In the dialog box, select the Show Only JDK Supported Locales option if you want to restrict the languages and countries list in this way. Otherwise, deselect the option.
4. Choose a language.

   The list is derived from the ISO 639 codes.
5. Optionally choose a country.

   The list is derived from the ISO 3166 codes.
6. If you did not select the Show Only JDK Supported Locales option, you can type a variant. For example, you could specify a dialect of Spanish, such as Mexican.
7. Click Add.

   A resource bundle specific to that locale is added to your project. The locale now appears in the Locales submenu, which you can see when you choose Localization from the Tools menu.
8. Click Close after you've added all the locales you want.

   The next time you work with locales, the locale you most recently added is automatically selected for you.

   If you want to add words for a dialect, see "Adding information to the resource bundle for a locale" on page 13-13.

To delete a locale:

1. In the Project window, select the file for which you want to delete a locale. Or activate the Form Designer or Source window for the file.
2. Choose Localization from the Tools menu, then choose Delete Locale from the submenu.
3. Select one or more locales, then click Delete.

   The specified locales are deleted.

To choose a locale to use for a project:

1. Open the project you want to choose a locale for.
2. Choose Localization from the Tools menu, choose Locales, and then choose a locale from the submenu.

   The locale you've chosen is now applied to the active project.

Adding information to the resource bundle for a locale

You can add words to a resource bundle from within Visual Cafe. Simply choose the locale and type in the Form Designer or Property List.

Note: You need to add the locale before you can add information to it.

To add information to the resource bundle for a locale:

1. In the Project window, select the file for which you want to specify locale-specific information. Or activate the Form Designer or Source window of the file.
2. Choose Localization from the Tools menu, choose Locales, and then choose a locale from the submenu.

   The most recently added locale is automatically selected for you.
3. Enter one or more words in the Form Designer or Property List.

   The value is added to the appropriate resource file in your project.

Editing a resource bundle

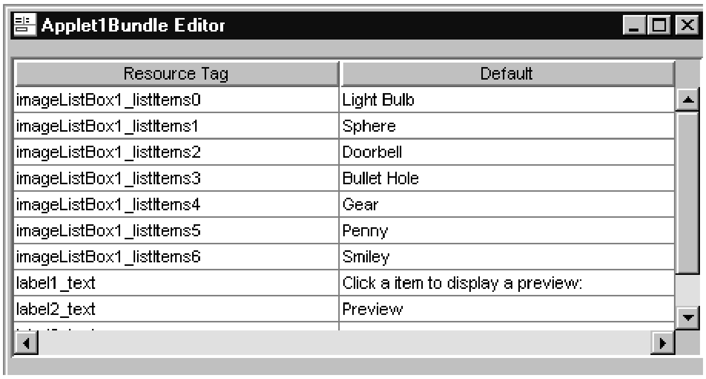
You can edit a resource bundle from within Visual Cafe with the Resource Bundle Editor.

Note: Before you can edit a resource bundle with the editor you must have already localized the current project, and you need to add a locale before you can add language- or country-specific information to it.

To edit a resource bundle with the Resource Bundle Editor:

1. In the Project window, select the resource bundle or the form that includes a resource bundle.
2. Choose Edit Resource Bundle from the Tools menu.

The Resource Bundle Editor appears.



Resource Tag	Default
imageListBox1_listItems0	Light Bulb
imageListBox1_listItems1	Sphere
imageListBox1_listItems2	Doorbell
imageListBox1_listItems3	Bullet Hole
imageListBox1_listItems4	Gear
imageListBox1_listItems5	Penny
imageListBox1_listItems6	Smiley
label1_text	Click a item to display a preview:
label2_text	Preview

The Resource Tag column shows the resource tags in the bundle. The Default column shows the information in the default resource bundle file, `Applet1Bundle`.

3 Edit the resource bundle as needed.

- ❖ To rearrange columns, drag a column heading to another location.
- ❖ To resize a column, drag the border.

The changes take effect immediately.

Converting between native and ASCII characters

If you're localizing your code and have added a locale that matches the locale set for your operating system (if you're working on a non-US English based system), you can automatically convert native characters to Unicode characters (ASCII format).

ASCII is a way to represent up to 256 different non-US English characters, each character being one byte in size. **Unicode** is a superset of ASCII; it can represent up to 32,768 characters, each two bytes in size.

A *non-US English based system* is either a non-US English operating system (such as British, French, German, or Japanese, for example), or the US English operating system whose system locale is set to a language other than US English.

If you're using a non-US English based system, the Java compiler might not understand some characters, such as those for Asian or some European languages. If these native characters appear in your resource bundle's source code, you need to convert them to Unicode before compiling. Visual Cafe can handle this conversion for you; this way you can view native characters in the Source window and and still compile your code.

For example, if Japanese is the current system locale, and you type a character native to Japanese, Visual Cafe can automatically convert the character to a Unicode character (for example, `\u3042`) in the resource bundle file when you save and open it. You are then able to compile this file.

Note: In order for this native to Unicode conversion to work, the resource bundle's locale must match the system locale of the computer running Visual Cafe. For example, if you're working on a computer with a Japanese system locale, you can only convert native characters to Unicode for resource bundle files that also have the locale set to Japanese (such as `Applet1Bundle_ja.java` and `Applet1Bundle_ja_JP.java`).

If you want to add native characters for a non-US English locale file that doesn't match your system locale (for example, Korean characters while your system locale is set to Japanese), you'll need to convert these native characters by hand into ASCII to make sure the code compiles correctly.

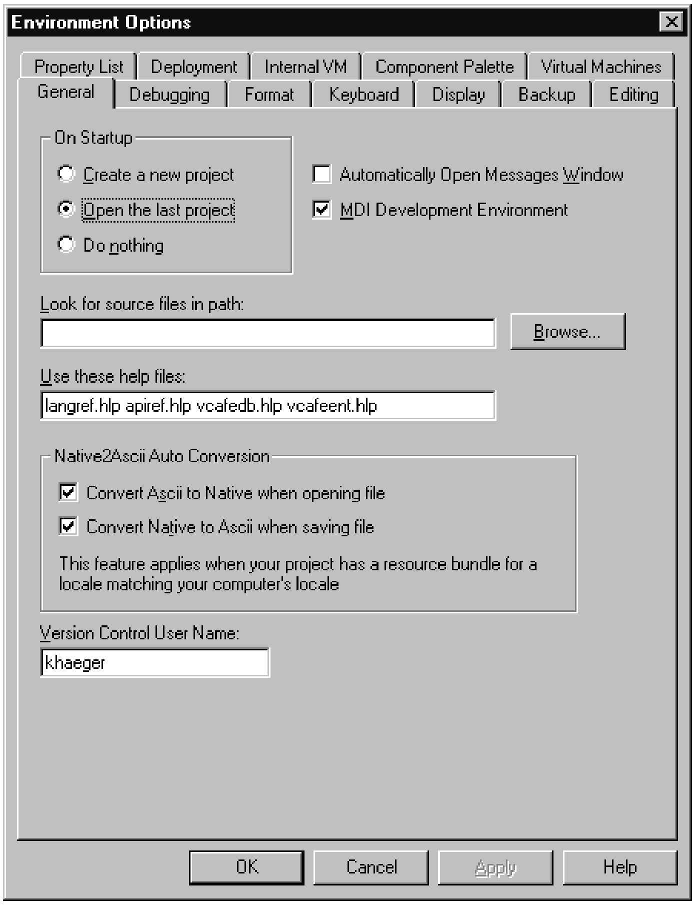
If you're inserting resource bundle files that you wrote in native characters into the project, you must convert them to Unicode before using them in the Visual Cafe environment.

Note: To ensure that your code compiles correctly, 1) use the Symantec Java compiler (provided and automatically in use in Visual Cafe) to compile your native character files, and 2) do not compile files with native characters on non-Win32 operating systems.

To convert native characters to ASCII:

1 Use the Visual Cafe localization features to create a resource bundle that matches the locale used by the operating system on your computer. Make sure this locale is the active locale in the Visual Cafe environment.

2 Choose Environment Options from the Tools menu, then click the General tab.



On Startup
- ○ Create a new project
- ● Open the last project
- ○ Do nothing

☐ Automatically Open Messages Window
☑ MDI Development Environment

Look for source files in path:

Use these help files: langref.hlp apiref.hlp vcafedb.hlp vcafeent.hlp

Native2Ascii Auto Conversion
- ☑ Convert Ascii to Native when opening file
- ☑ Convert Native to Ascii when saving file

This feature applies when your project has a resource bundle for a locale matching your computer's locale

Version Control User Name: khaeger

3 Select the Native2ASCII Auto conversion options you want:

❖ To have Unicode characters converted to native characters when you open a file (so that you can see the native characters in the Source window), select Convert ASCII to native when opening file. If you don't select this option, the Unicode value (in the form of \u*XXXX*) appears when you open the file in the Source window.

❖ To convert native characters to ASCII before you save a file, thus making it possible for the file to be compiled, select Convert native to ASCII when saving file. If you don't select this option, you won't be able to compile your file.

Note: In most cases, you'll want to select both Convert ASCII to native when opening file and Convert native to ASCII when saving file.

4 Click Apply or OK to save your changes.

If the resource bundle file matches your system locale, the changes take effect immediately.

IV

Appendixes

A P P E N D I X A

Updating Visual Cafe with LiveUpdate

LiveUpdate is a Symantec technology that allows you to upgrade selected Symantec products online. LiveUpdate keeps track of packages that are downloaded to your computer so that it can determine the appropriate upgrade for your software.

About LiveUpdate

LiveUpdate enables you to update your version of Visual Cafe. It communicates with the Symantec Update Center using TCP/IP in either of two ways: over a network connection to the Internet, such as your office workstation uses, or through a connection with your online service or Internet service provider. LiveUpdate can communicate using either FTP or HTTP.

If you don't have access to the Internet or your access is restricted by firewalls, you can access LiveUpdate through a modem connection to a Bulletin Board System (BBS).

When you install Visual Cafe, LiveUpdate is also installed. When you register your copy of Visual Cafe through online registration, LiveUpdate's services are enabled for Visual Cafe. LiveUpdate is then available from Visual Cafe's Help menu.

Installation also adds a LiveUpdate program icon to your Windows Control Panel. You can use this icon to help configure your LiveUpdate settings, to troubleshoot connection problems, or to update your copy of the LiveUpdate product itself.

To connect to the Symantec Update center:

1. Choose LiveUpdate from the Help menu.

   If you haven't already registered at the Update center, you'll be prompted to do so. The Update Center will ask you for the User ID and Password found on a sheet of paper in your Visual Cafe package. When you've entered that information, your browser takes you to a Visual Cafe page at the Update Center site.

2. Click the link labeled Download Visual Cafe software.

   Your browser displays a page about downloading Visual Cafe. On that page is a link to the LiveUpdate registration utility.

3. Click the Go beside that information.

   Your browser displays a page with directions about getting the LiveUpdate Registration Utility. Follow those directions to obtain a small utility that you must execute to enable LiveUpdate.

4. Run the LiveUpdate Registration Utility.

   You're connected to the Update Center.

After you connect to the Update Center, LiveUpdate presents you with the most recent update to your version and edition of Visual Cafe. Note that if you have a version more than one generation older than the new version, you do not need to install the intervening updates. For example, if you have Visual Cafe 3.0a on your hard drive and you want to upgrade to 3.1, you need not install Visual Cafe 3.0b and any other intervening versions of Visual Cafe before updating to the 3.1 version.

Using LiveUpdate over the Internet

If you already have a working connection to the Internet, select the Use Existing Internet Connection option. No further setup is required.

If you do not have a working connection to the Internet or you wish to connect by modem instead, you should proceed to the Select a Modem page of the LiveUpdate Setup Wizard and select your modem (and the communication port your modem is using) from the available choices. See the following section for details.

Using LiveUpdate with your modem

If you have a TCP/IP-based connection to the Internet through your Internet service provider, you can use LiveUpdate to update your copy of Visual Cafe.

Note: Your dialup or ISDN account with your Internet service provider must be TCP/IP-based, such as a PPP, SLIP, or CSLIP account. You cannot use LiveUpdate through a shell account.

You can also use LiveUpdate if you have an account with an online service such as America Online. You must have the appropriate software installed and configured to use TCP/IP.

Configuring your modem

When LiveUpdate starts, the first wizard page contains a button labeled Modem Setup. Clicking this button allows you to change your modem settings at any time. If you choose not to set up your modem initially and use your existing Internet connection instead, you will still be able to set up a modem in the future.

Identifying your modem

If you have an external modem, examine it for a manufacturer and/or model label. Sometimes this information is on the underside of the modem. If you can't identify your modem in this manner, or if you have an internal modem, try selecting Standard Modem Types from the list of manufacturers and then selecting the Standard modem model that most closely matches your modem's speed, if you know it. For example, if you have a 14,400 baud (14.4) modem you might try selecting 14000 bps Modem as the model.

If you don't know the speed of your modem, try selecting 9600 bps Modem. If that doesn't work, try 2400 bps Modem. As a final alternative, try selecting Hayes as the manufacturer and either Compatible (default) or Compatible (alternate) for the model. If none of these selections results in a successful connection to a LiveUpdate service, you will need to consult your computer equipment vendor or company MIS/Help Desk for assistance in determining your modem type.

If your modem's manufacturer is listed but your particular model is not, you should select Other from the model list. This choice will usually enable your modem to properly connect, download, and disconnect.

Configuring your modem's INIT string

If LiveUpdate has your modem manufacturer and model listed, but you'd like to modify the INIT string to better suit your preferences, LiveUpdate allows you to do this by clicking the Edit String button in the Select a Modem wizard page.

Most users will never need to change their modem's default INIT string; this option is provided for the few who need or want to modify it. Only advanced users who know the implications of changing their modem's INIT string should attempt to do so.

Note: LiveUpdate requires E0 and V1 to be present in the INIT string for proper operation.

If you've made modifications to the default INIT string but would like to restore the original string for your particular modem, you can do this by reselecting your modem manufacturer and model from the list. Your custom INIT string will be replaced by the default string for the selected modem.

Selecting the COM port

In the Select a Modem wizard page, you're asked to select the COM port for your modem. If you aren't sure which is the correct COM port, click the Find My Modem button. The LiveUpdate Setup Wizard will attempt to locate any modem(s) you have attached to any of the four basic COM ports. If LiveUpdate finds a modem, it places a red dot next to the corresponding COM port.

If you have two or more modems on your system, LiveUpdate allows you to choose the correct one if you know the COM port number of the modem you want to use. To select the corresponding COM port, highlight that COM port in the listbox. If you don't know the COM port, press the Find My Modem button and LiveUpdate will place red dots corresponding to all found modems next to the appropriate COM port number or numbers. Select the COM port from among the entries marked with a red dot.

Selecting dialing parameters

In the Number to Dial wizard page, a number of countries are listed. Select the appropriate country, based on where you're located geographically at the time you run LiveUpdate. If your location isn't listed, choose the nearest location. Users in the United States and Canada should choose the service in Eugene, Oregon. Users in New Zealand should connect to the service in Australia. If you are a laptop user and travel extensively, you may need to change the service location to which you connect based on the service that is closest to you at the time you run LiveUpdate.

If your company's phone system requires that you dial a "9" or other code to access an outside line, you must enter the number in the Number to Dial wizard page. There's a box at the bottom of the page labeled LiveUpdate will dial. Normally, this box contains the phone number of the service to be dialed. However, you can add whatever codes you need before or after the number. Specifically, you may enter codes to access outside lines, dial country codes, enter a calling card number to bill the phone call to, disable call waiting, access alternate long distance carriers, or add any other required codes.

If you have call waiting on your phone line, you can disable it so that you won't get disconnected if you receive a phone call while you're using LiveUpdate. The code required to disable call waiting varies depending on your phone system. Commonly, codes such as *70, 70#, or 1170 are dialed before the phone number to temporarily disable call waiting. For the specific code to disable call waiting in your area, contact your local phone company.

If you need to dial a particular access code and then wait for a few seconds before dialing the remainder of the number, you can tell LiveUpdate to pause by entering a comma in the LiveUpdate dial edit box after the last digit where you want the pause to occur. The comma causes a modem-dependent pause period, commonly a few seconds. To increase the time, add more commas.

Troubleshooting a LiveUpdate connection

If you're informed that LiveUpdate detected a problem while retrieving your software update, run LiveUpdate again.

While a number of errors may have occurred, the most likely problem is that you have insufficient hard disk space to accommodate the size of the software update being downloaded. The updates are compressed, so they require additional hard drive space beyond their original (compressed) size. If there is not enough space for the final files, this error may be displayed. If this error occurs, you should clear some hard drive space (usually on the drive that contains Windows) and try LiveUpdate again.

LiveUpdate informs you if you have the latest software update for your product. When you next use LiveUpdate, it will retrieve any updates that have been released since you last ran it.

You can also try using the LiveUpdate Control Panel to help you in your troubleshooting efforts.

Uninstalling LiveUpdate upgrades

After you've uninstalled previous software upgrades, your Windows 95, 98 or Windows NT registry may still be "dirty" and therefore not allow LiveUpdate to install upgrades to Visual Cafe.

LiveUpdate includes a utility within the Visual Cafe directory to "clean up" the registry for Windows 95, 98, or Windows NT and allow you to use LiveUpdate again to install upgrades to Visual Cafe.

Using LUCLEAN.EXE

`LUCLEAN.EXE` removes any changes LiveUpdate leaves behind in the Windows registry. `LUCLEAN.EXE` is non-destructive to any component of the operating system. This utility cleans up only LiveUpdate entries for Visual Cafe.

To run LUCLEAN.EXE:

1. Open a DOS session window.
2. Switch to the `Visual Cafe` directory.
3. Type `LUCLEAN.EXE /LU`.
4. Restart your computer and run LiveUpdate again.

A P P E N D I X B

Troubleshooting

This chapter describes various ways you can configure the Visual Cafe environment to your development preferences. It also discusses common programming problems and concerns you may encounter in Java or Visual Cafe, and presents solutions.

For additional support information, visit Symantec's technical support website at the following URL:

```
http://service.symantec.com
```

You can access an FAQ (*Frequently Asked Questions* listing), a knowledge base, and newsgroups. You can access the newsgroups either with your newsgroup reader software, or by way of your Web browser. After posting a message to a newsgroup, you'll get a response within 24 hours from a Visual Cafe technical support representative. You can access the following newsgroups either by way of a Web browser or a newsgroup reader:

```
symantec.support.itools.win.vcafe.compilation
symantec.support.itools.win.vcafe.database
symantec.support.itools.win.vcafe.deployment
symantec.support.itools.win.vcafe.development_environment
symantec.support.itools.win.vcafe.distributed
symantec.support.itools.win.vcafe.execution
symantec.support.itools.win.vcafe.installation&upgrade
symantec.support.itools.win.vcafe.other
```

Programming concerns

As with all development tools, your success with Visual Cafe can depend on a number of things. Your programs may or may not work correctly, depending on factors such as your system configuration, Visual Cafe environment settings, and the capabilities of the Java language.

The following sections provide information that will help you solve some of the most frequently encountered development problems with the Java language and Visual Cafe.

Limitations of the Java language

Java has some architectural limitations that keep it from performing certain tasks. You should keep these limitations in mind when you develop programs using Visual Cafe.

Java and case sensitivity

Java compilers and interpreters are case sensitive, meaning that upper-case letters are distinguished from lower-case letters. `Hello.java` is not the same file as `hello.java`. The different use of capital letters indicates that these files are not the same.

When a `.java` source file is compiled, the compiler creates a file with a `.class` extension. The name that the compiler assigns to that `.class` file is taken from the class definition in the source file. Essentially, the class you define in the source code becomes the name of your program. Like all things in Java, the class definition is case sensitive.

Because of the case-sensitive nature of Java code, compilers, and interpreters, it's important to be careful and consistent when you name projects, source files, and classes.

To prevent problems resulting from case sensitivity, get in the habit of always naming the project and the main source file with the same case as the class you're creating.

Hardware limitations

Another limitation of the Java language is its inability to communicate directly with computer hardware, such as video cards, modems, or any other devices that use some kind of port on a computer. The Java VM is blind to system-specific resource details, making it impossible for Java to access hardware.

Browser issues

Although Visual Cafe is designed to be an efficient and fast development tool, you must also develop your Java applet to run on other combinations of platforms and Web browsers.

Besides the built-in security restrictions in Java applets (see "About applets, applications, servlets, and libraries" on page 2-9), your Java applet will probably have a different look and feel (and sometimes behavior) on other platforms and browsers. Here are some things to consider while you're developing your Java program:

- Select or design your GUI components carefully. For example, a button being viewed with the Macintosh version of Netscape Navigator will look different when you run the same program in Microsoft Internet Explorer for Windows NT. The same button could look different between the different versions of Netscape Navigator for the various flavors of UNIX.
- Understand how your platform and browsers handle threads. Macintosh, Windows 95, Windows NT, and UNIX operating systems each handle threading differently, as do their respective browsers.
- Each platform and browser has its own way of terminating applets. For example, when you start an applet in one browser, move to another page, then move back to the original page, the applet still runs and is consuming resources. In other browsers the same applet will terminate when you move to another Web page and then back. As of this writing, this inconsistency is not due to a Java language-specific problem, but rather a lack of standards for implementing Java capabilities in Web browsers.

When do you have to write your own code?

Generally, Visual Cafe generates all the source code you need to quickly create basic applets and applications. In many cases you need to write little or no source code. Visual Cafe also provides the framework for developing complicated applets and applications, eliminating the need for writing the routine and tedious source code and allowing you to develop and write the more sophisticated aspects of your Java program. In some cases you'll need to write your own source code, such as when you need to implement event handling that can't be done through the Interaction Wizard.

Disabling automatic code generation in Visual Cafe

If you don't need to use some of the RAD features of Visual Cafe, such as automatic code generation, you can disable them. For information about turning RAD off, see "Enabling and disabling RAD and automatic code generation" on page 4-45.

Disabling sections of automatically-generated code

You can tell Visual Cafe not to execute code that it generates. Place the Visual-Cafe-generated code between a block like this:

```
if(false) {
//{{INIT_CONTROLS
.
.
.
//}}
}
```

Visual Cafe sees your code at design time, but at run time the code that is automatically generated is ignored.

Repairing a corrupted Visual Cafe environment

When you start Visual Cafe, if your Component Palette is empty or if you get an error saying "There are no starter templates in your repository," your Visual Cafe configuration files have somehow become corrupted. You can restore these files to repair your Visual Cafe environment.

To restore your Visual Cafe environment back to its working state:

1. Exit Visual Cafe.
2. In the `VisualCafe\Bin` directory, delete the following files:
   - ❖ `local.rps`
   - ❖ `vcafe.reg`
   - ❖ `VisualCafe.vws`
3. Restart Visual Cafe.

   Visual Cafe should now run normally.

G L O S S A R Y

A

action component	In an interaction, the component on which a defined action happens.
anchor component	A component that acts as the reference point when you're modifying a group of components. When you select multiple components, the last component selected is the anchor component. It has distinct, colored selection handles. *See also* component.
API	(Application Programming Interface) The interface used by an application program to access an operating system and other services.
applet	A Java program that you can add to a web page and run using a Java-enabled web browser. *See also* application.
applet tag	HTML code that causes an applet to appear in a web page. It has the following basic format: `<APPLET code="`*applet.*`class" width=`*pixw* `height=`*pixh*`></APPLET>` *applet* is the name of the applet. *pixw* is the number of pixels for the width. *pixh* is the number of pixels for the height.
AppletViewer	A Java utility that lets you run and debug applets.
application	A Java program you can run from a computer that has the Java Virtual Machine. An application is a stand-alone program, while an applet runs in a web page. *See also* applet; Java Virtual Machine.
application template	A Visual Cafe template that creates a class file that's an extension of the Frame class. This class is a good base for a main application window.
AutoJAR	A Visual cafe feature that, when enabled, automatically shows changes made to JavaBeans components and updates the JAR (Java Archive) file.
AWT	(Abstract Windowing Toolkit) A standard and portable GUI library that you can use to create visual user interfaces.

B

Bean	A component that complies with the JavaBeans standard. Also called a *JavaBeans component*. A Bean is a reusable component that can be visually manipulated in a builder tool such as Visual Cafe. For more information, see the Java web page at `http://java.sun.com/beans/`.
Boolean expression	An expression that evaluates to true or false. The most common Boolean functions are AND, OR, and NOT.
breakpoint	A flag you can insert into code at certain points to pause program execution. You can set simple breakpoints that stop execution at a certain line or method, or conditional breakpoints based on an expression.
Breakpoints window	A debugging window that contains a list of all breakpoints in a project. Use this window to add, remove, or modify breakpoints. *See also* breakpoint.
buffer	An already open file or Class Browser window that is in temporary memory.
bytecode	Machine-independent code generated by the Java compiler and executed by the Java interpreter.

C

CAB file	(Microsoft Cabinet file) A single file created to hold a number of compressed files, for use in Microsoft program development. A cabinet file usually has the file name suffix of `.cab`.
call	A reference made from one class to methods in another class.
call chain	The sequence of functions that were called to get to the current function. *See also* call.
call stack	An area reserved in memory by the compiler to keep track of all method calls that are made. When an applet or application calls a method, the Java Virtual

	Machine (Java VM) adds the method to the stack. When the method returns, the Java VM takes it off the stack. The currently executing method is on the top of the stack and the previously called methods are below it.
Call Stack window	A debugging window that shows all the active method calls leading to the current process. This window is useful for following the flow of your code.
class	A collection of variables and methods that you can use to define an object. The variables define the class structure and the methods define the class behavior. When compiled as a bytecode program (as opposed to a native Win32 executable), a Java source file becomes one or more class files. *See also* Class Browser; class path; inner class; main class.
Class Browser	A three-paned window that lists the Java classes in your project and the methods and data members contained within each class. *See also* Classes pane; Members pane; Source pane.
class path	The environment setting that indicates where class files are located on your computer.
Classes pane	A Class Browser pane that displays all the classes that are part of your project. By default, the classes are displayed by package. *See also* Class Browser.
Code Helper	A Visual Cafe helper tool that interprets the current source code context and provides either a list of methods in a given class, a list of the different versions of a particular method, or a list of class objects that begin with a particular character sequence.
component	A reusable object that allows you to interact with your program. Examples of components include user-interface elements such as scroll bars, buttons, and text-entry fields. All components supplied with Visual Cafe are JavaBeans components. *See also* action component; anchor component; container; Component Library; Component Palette; JavaBeans component; non-visual component; peer component; top-level component; visual component.

Component Library	A collection of components that you can add to your project.
Component Palette	A configurable toolbar that provides easy access to components. You can place any component that is in the Component Library on the Component Palette.
conditional breakpoint	A breakpoint that lets you stop program execution when a specified expression evaluates to true. *See also* breakpoint.
container	A component that can contain other components, such as an application window that contains a button. A top-level container, also called a *form* in Visual Cafe, is at the top level in the Objects view of the Project window. It has a corresponding Java file that appears in the Packages and Files views. *See also* form.
contextual menu	A menu that appears when you click the secondary mouse button (usually the right button) in a window. A contextual menu is often called a *pop-up menu.*
custom code	Java code that you add yourself, as opposed to code that Visual Cafe automatically creates for you.

D

data member	*See* member.
debug mode	An integrated workspace that provides facilities to find and fix errors in your program's source code. You can manage threads, set or clear breakpoints, and view the methods on the call stack. After editing your code, restart program execution to see the effect of your changes. *See also* incremental debugging.
Debug Toolbar	The Visual Cafe toolbar that has buttons for debugging tools.
declaration	A statement that establishes an identifier and associates attributes with it, without necessarily

	reserving its storage for data or providing the implementation for methods.
definition	A declaration that reserves storage for data or provides implementation for methods. *See also* declaration.
deploy	To package your completed program for distribution.
deployment target	The destination of a deployed program, which can be a JAR, CAB, ZIP, or directory.
design time	The time during which you create and build an applet or application in the development environment.
dialog box	A window that can receive and process input from a user (for example, clicking a button or selecting an item in a list).
display order	*See* z-order.
DLL	(Dynamic Link Library) A library which is linked to application programs when they are loaded or run rather than as the final phase of compilation.

E

environment options	Options that affect various aspects of the way you work in Visual Cafe. You select these options in the Environment Options dialog box. You can customize display, deployment, debugging, editing, and backup options, among others. *See also* project options.
event	An action or occurrence to which an object can respond. Events are typically user actions that the program can capture and respond to. For example, mouse clicks, key presses, and mouse movements are events.
event adapter	The event listener interface that's implemented as a class. *See also* event listener.
event binding	The code that enables an object to receive and process an event. It's made up of three parts: the event handler, the listener or adapter implementation,

	and the code to register the listener or adapter to the object triggering the event. *See also* event adapter; event handler; event listener.
event handler	A method that's called when a certain type of event is triggered. Visual Cafe automatically generates the code needed to bind the occurrence of an event to an event handler when you create an interaction with the Interaction Wizard or the Interaction Tool. The default name of an event handler is the object name, followed by an underscore, then the name of the action that triggers the event.
event listener	An object that has defined the listener interface for a specific event. After this interface has been implemented in a class, an instance of this class may be registered as an event listener. When an event is generated, the event is sent to the object, as well as all other registered listeners.
event source	An object that generates events, such as an AWT component.
exception	An event that occurs during the execution of your program that interferes with, disrupts, or stops the normal flow of your program.

F

Files view	A Project window view that lists all the files in a project.
form	A container for components that allow a user to interact with your program. A form might contain a series of buttons for a user to click, a text-entry field, or a menu, for example. Applet, Frame, Window, and some dialog components are forms. You can work on forms in the Form Designer.
Form Designer	The window that visually displays form components so that you can design the form's user interface.
frame	A window that contains user-interface components such as buttons, text-entry fields, and menus. You use

	frames when creating a stand-alone application, as opposed to an applet.

G

garbage collection	The process of automatically freeing memory that is no longer in use. The Java run-time system performs garbage collection so that you don't have to explicitly free the memory associated with objects and other data.
group	A collection of related files in the Component Library.
GUI	(Graphical user interface) A user interface made up of visual elements such as windows, menus, icons, and so on.

H

Hierarchy Editor	The window that displays the class hierarchy for your project.
HTML	(Hypertext Markup Language) The formatting that makes a text file into a web page. HTML pages display in a Web browser.

I

IDDE	(Integrated Development and Debugging Environment) A programming environment that provides tools that are interrelated; if you make a change in one part of the environment, it will automatically be reflected in other parts.
identifier	A unique term that names a Java language object, such as a variable, method, or class.
incremental debugging	A feature that enables you to edit code while your program is executing or paused in the Visual Cafe debugger. Also called *run-time editing*. This feature is found in Visual Cafe Professional and Database Editions.

inheritance	The concept wherein classes automatically contain the variables and methods defined in their superclasses.
inner class	A class that's included within the body of another class, even within a method (called a *local class*). An inner class is also called a *nested class*. This feature is new for JDK 1.1 and is useful for creating adapter classes. After compilation, the inner class ends up in its own class file, which has a dollar sign ($) in its name.
instance	A data item that's based on a class. An instance of a class is usually called an *object*. For example, multiple instances of a Form class share the same code and are loaded with the same components with which the Form class was designed. At run time, the individual properties of each instance can be set to different values.
Interaction Tool	A tool that lets you visually create interactions between components in the Form Designer.
Interaction Wizard	A wizard that provides a step-by-step interface for building relationships between components.
interaction	The relationship between two or more components, or a component and itself. The components may be on the same form or on different forms. An interaction consists of: one or more components (trigger component and action component), a trigger event, and an action. For example, you can connect a button (the trigger component) to a text box (the action component) so that when the user clicks on the button (the trigger event), the associated text box is enabled for user input (the action).
interface	A set of methods and constants to be implemented by another object. It defines the behavior, or certain characteristics, that another object implements. An interface can define abstract methods and final fields, but not the implementation of them.
introspection	A JavaBeans component's ability to make public (publish) the operations, methods, and properties it supports, and its ability to discover operations,

	methods, and properties of other Beans. Introspection calls on two API processes: the Java Reflection API and the Java Serialization API.
invisible component	*See* non-visual component.

J

JAR file	(Java Archive file) An archive file that complies with the JavaBeans standard. A JAR file can contain one or more Beans and related support files, including classes, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. You can deploy applets and applications from a JAR file, which can optionally be compressed.
JAR Viewer	A utility that lets you view the contents of a JAR file.
Java API	(Java Application Programming Interface) An API provided by the JDK (Java Development Kit). For more information, see Sun's Java web page at `http://java.sun.com`. *See also* API.
Java file	A file that contains Java components and source code.
Java Archive file	*See* JAR file.
Java Development Kit	*See* JDK.
Java Virtual Machine	(Java VM) The virtual machine provided with the JDK (Java Development Kit). The Java VM contains a bytecode translator that converts a downloaded binary Java file into instructions that the client machine can execute, and also library routines that a Java applet calls. For more information, see Sun's Java web page at `http://java.sun.com.`
JavaBeans	A standard for creating portable, cross-platform components. For more information, see the Java web page at `http://java.sun.com/beans/.`
JavaBeans component	*See* Bean.

Javadoc	A utility that scans Java source code and generates HTML documentation from it. Extra HTML files can also be generated which show all the classes in a particular package, an index, and a hierarchical class tree list.
JDK	(Java Development Kit) Tools for developing Java applets and applications. The JDK is included in Visual Cafe. It's also available on Sun's Java web page at `http://java.sun.com.`
JFC	(Java Foundation Classes) A set of Swing components plus several accessibility features. *See also* Swing.
JIT	(Just-In-Time compiler) An integrated tool that compiles bytecode into machine language and then caches it for much faster execution than by the VM itself. Instead of using a command-line interface, as does Sun's Java compiler, the Symantec JIT compiler lets you click on a single icon to compile and execute a program.
Just-In-Time compiler	*See* JIT.

L

layout	(Also called *layout manager*) A property for container objects like forms and panels. The Layout property automatically arranges components in a container so that they display well on different platforms, web browsers, screen sizes, and screen resolutions.
LiveUpdate	A service that lets you download the latest version of Visual Cafe.
local variable	A data item that's defined within a block of code and accessible only by the code within the block. For example, a variable defined in a Java method is a local variable and can't be used outside the method.

localization	Changing a program to make it appropriate for a particular country, region, or dialect. Visual Cafe includes a tool to speed up localization.

M

macro	A sequence of keystrokes that automate a task.
main class	A class that contains a `main` method. The main class is the starting point for program execution.
manifest file	A file that describes the contents of an archive such as a JAR. The file provides information on certain parts of the archive and can provide information about any JavaBeans in a JAR.
MDI	(Multiple Document Interface) A workspace setting that allows you to float or dock certain windows. *See also* SDI.
member	A variable or method defined in a class.
method	Behavior that acts on an object. A method is defined in a class. A *class method* is a method that's invoked by using a class name. An *instance method* is a method that's invoked by using the name of an instance (the object).
Messages window	A window that displays informational and error messages. For example, if Visual Cafe detects an error during parsing, the error message is displayed here. You can double-click an error message to open the file in which the error was detected.
modal	Window or dialog box behavior that requires you to take some action before the focus can switch to another window or dialog box.
modeless	Window or dialog box behavior that doesn't require you to take some action before the focus can switch to another window or dialog box.
multithreading	A program's use of threads to perform several processes simultaneously. *See also* thread.

N

nested class	*See* inner class.
non-visual component	A component that is not visible at run time, such as a Timer, or displays in a different way in the Form Designer and at run time, such as a MenuBar. A non-visual component doesn't extend from the Java `component` class. In the Form Designer, a non-visual component is represented by an icon that doesn't affect the form layout. *See also* visual component.

O

object	An instance of a class. *See also* instance.
Objects view	A Project window view that displays the components and HTML files (if any) in a project.

P

package	A group of related classes and interfaces that can be used by programs that import the package. Java source code is organized into packages. Each part of a package name generally refers to a hierarchical directory structure. For example, `COM.sun.java.swing` is in the directory structure `/com/sun/java/swing`. If you're going to distribute your packages, you should create a globally unique name based on an Internet domain name. For example, `sun.COM` is specified as `COM.Sun`. This first part of the name is in uppercase letters; if the first part isn't in uppercase letters, it's for local use, except for packages that are part of the Java language and system (which start with `java`). Packages help prevent naming conflicts. For example, two Java files could have the same names, but as long as they're in different packages, there's no naming conflict.
Packages view	A Project window view that displays the packages in your project. *See also* packages.

panel	A container that you can add to another container. You can use a panel to group a window into logical regions.
project	An organized collection of related files that make up a Java program you're creating in Visual Cafe. The program can be an applet, application, JavaBeans component, or library.
project file	The central element of a project. The project file contains all the information you need to manage the project, such as the locations of items in a project, as well as additional information such as compiler options and browser data. *See also* project.
project folder	Most of the elements of a project, together with the project file, are kept in this folder. *See also* project file.
project options	Options that affect various aspects of the way you work in a particular Visual Cafe project. You select these options in the Project Options dialog box. You can customize version control, debugger, compiler, and deployment options, among others. *See also* environment options.
project path	The path for components that are specific to a particular project. The default project path is the project folder, along with all subfolders it contains. *See also* project folder.
project template	A collection of components that you can use as the foundation for an applet, application, library, or Bean. When you choose a template for a new project, the new project inherits all of the template's components. You can create project templates for web sites, single documents, and applets.
Project window	The window that displays a project's contents. You can see a list of objects, packages, or files, depending on whether you select the Objects, Packages, or Files tabs. *See also* project.
property	An attribute of a component. Properties define component characteristics such as size or color, or the

state of an object, such as enabled or disabled. *See also* Property List .

Property List A window that displays the properties of a project's components.

R

RAD (Rapid Application Development) The process of creating programs in a visual development environment such as Visual Cafe.

resource bundle A file containing locale-specific information. When a program needs a resource specific to a locale, such as a string in French, the program can load the resource from a resource bundle for that locale. This way, you can write locale-independent code that stores locale-specific information in resource bundles.

run time The time when your applet or application is running. At run time, you can interact with your program as a user. If you encounter a problem while running the program, you can switch from run mode to debug mode. *See also* debug mode.

run-time editing *See* incremental debugging.

S

scope The access type of a variable or method.

- Public: Accessible from any class.
- Protected: Accessible from the class that defines the variable or method and from the class's subclasses.
- Private: Accessible from the class that defines the variable or method.
- Package: Accessible from any class in the package that defines the variable or method. This is the default access type.

SDI	(Single Document Interface) The user interface employed by Visual Cafe versions prior to 2.5. In version 3.0, you can use SDI or MDI. *See also* MDI.
servlet	A Java program that runs on a Java-enabled server and can dynamically extend server-side functionality. A servlet can provide services using a request-response paradigm; it doesn't have a GUI. For example, a servlet can provide secure access to data presented in an HTML Web page and let users interactively view or modify data. *See also* applet
SJ	(Symantec Javac) A utility that lets you compile Visual Cafe programs from a DOS window, thereby saving memory while compiling.
Source pane	The pane in the Class Browser that lets you view and edit Java source code. (The Source pane serves the same functions as the Source window.) *See also* Source window.
Source window	The window that lets you view and edit Java source code, an HTML file, or a text file. The Source window is also available when you're debugging. (The Class Browser's Source pane serves the same functions as the Source window.) *See also* Source pane.
Swing	Refers to the Java Foundation Classes (JFC), which provide a comprehensive set of GUI components that extend the functionality of AWT components. Swing is not platform-specific and offers a pluggable look and feel.
Syntax Checker	A Visual Cafe troubleshooting tool that highlights possible Java syntax errors while you're typing code in the Source window.
system path	The path where classes are located. Components that will be used in many projects should be placed in folders in the system path. The default system path is the `Java Libraries\Classes` in your Visual Cafe folder.

T

target	In Java, the applet or application that is the result of a project. In HTML, the point to which a link directs a user.
template	*See* project template.
thread	A path of execution in a running application. For example, an application can have threads that handle background processes that aren't visible to the user.
Threads window	The debugger window that displays all the threads your program has created, as well as the state of each thread. You can pause individual threads, which causes their execution to cease temporarily while all other threads continue to execute, then resume them. This helps you check for and resolve thread synchronization errors, where more than one thread is in contention for the execution of a method. *See also* thread.
top-level component	*See* form.
trigger event	The originator of an interaction between components.

V

variable	The structure in memory that holds data that has been assigned to it. A variable that is defined in a class defines the class's structure. A *class variable* is a variable associated with a class and not with a particular instance of a class. An *instance variable* is a variable associated with an instance of a class (an object)
Variables window	The debugger window that shows the variables that are active in the current context. To change a variable value at run time, edit its value in this window.
visual component	A user-interface element, such as a window, menu, or button, that appears in the Visual Cafe Project window and Form Designer and is visible at run time. It extends from the Java Component class and has a screen position, a size, and a foreground and background color. *See also* non-visual component.

Visual Page	A Symantec application that lets users design and publish web pages in a visual environment. Visual Page is included with Visual Cafe Professional and Database Editions.
VM	*See* Java Virtual Machine.

W

Watch window	The debugging window that displays the value of a variable or expression that you enter. The values update when you pause execution or step through code. You can also examine the contents of a class member. You can modify values directly in this window and continue debugging without having to stop and restart the debug session.
web server	A computer that stores and sends out web pages in response to HTTP (Hypertext Transfer Protocol) requests from web browsers.
wizard	An interactive help utility that guides you through a task by presenting a series of screens where you enter information.
workspace	A saved arrangement of windows that have related functions. For example, you might use one configuration of windows when you're building interactions between components, and another when you're debugging your code.

Z

z-order	(Also called *display order*) The order in which components are diaplayed. The z-order affects the visibility of visual components when they're layered on top of one another.

INDEX