

## No Help

There is no help available on this topic.



## No Help

There is no help available on this object.

This functionality is currently not supported.

# Main Windows

Author note: Popup listings were added to the menus topic because of the number. There are too many for the See Also button.

- How To
- See Also
- Concept
- API
- Components



{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Visual Cafe window is the main interface to the Visual Cafe development environment.

The window consists of three main elements:

- menus
- toolbars
- Component Palette

Closing the Visual Cafe window closes the application.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 Concept

 See Also

The Visual Cafe menus provide access to all Visual Cafe functionality.

These are the standard menus that are always available:

File	Edit	View
Search	Project	Insert
Tools	Window	Help

{button ,PI('VCAFE.HLP','Popup\_Standard\_menu\_listing')} [Click here for quick access to standard menu help.](#)

The availability of the following menus depends on the tool you are currently using:

Object	Run	Source
Layout	Calls	Threads
Breakpoint	Variables	

{button ,PI('VCAFE.HLP','Popup\_secondary\_menu\_listing')} [Click here for quick access to menu help.](#)

There are also several pop-up menus that are dependent on the active object:

Project window	Form Designer
Class Browser	Menu Designer
Source window	Hierarchy Editor

{button ,PI('VCAFE.HLP','Popup\_popup\_menu\_listing')} [Click here for quick access to popup menu help.](#)

[File Menu](#)

[Edit Menu](#)

[View Menu](#)

[Search Menu](#)

[Project Menu](#)

[Insert Menu](#)

[Tools Menu](#)

[Window Menu](#)

[Help Menu](#)

[Object Menu](#)

[Run Menu](#)

[Source Menu](#)

[Layout Menu](#)

[Calls Menu](#)

[Threads Menu](#)

[Breakpoint Menu](#)

[Variables Menu](#)

[Objects View Pop-up Menu](#)  
[Packages View Pop-up Menu](#)  
[Files View Pop-up Menu](#)  
[Form Designer Pop-up Menu](#)  
[Menu Designer Pop-up Menu](#)  
[Class Pane Pop-up Menu](#)  
[Member Pane Pop-up Menu](#)  
[Hierarchy Editor Pop-up Menus](#)  
[Source Window Pop-up Menu](#)

# Menus

Author's note: individual command topics were added because the engineers did not have time to fix the software to assign unique helpid numbers to each command. This caused the wrong menu topic to display for the relative position of the command. For example, the Cut command is used 4 places in the software and can only be mapped to one helpid string.

### **Third-party menu item**

This menu item was created by a third-party product you have integrated into Visual Cafe. Consult the documentation that came with this product for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Help menu:

**Help Topics**

Opens the Content listing for the Visual Cafe online help. The online help provides task and context sensitive information.

**Java API Reference**

A reference of all Java packages, classes and interfaces for each package. Variables, constructors, and methods are also included.

**Java Language Reference**

Opens the Content listing for the Java Language Reference.

**Macro Reference**

Opens the Content listing for the Java Language Reference.

**Visual Cafe Demo**

Displays an online demo of Visual Cafe.

**On the Web**

Starts your default browser and takes you to Internet links that you might find useful.

**LiveUpdate**

Helps you update your software across the Internet through a Symantec Web site. Before you can use LiveUpdate, you must register at the Symantec update center; after registering, you are given a file that enables LiveUpdate.

**About Visual Cafe**

Displays version and copyright information for this release.

## **User's Guide**

Opens the Content listing for the Visual Cafe User's Guide. This document provides conceptual information about Visual Cafe.

## Help Topics

Opens the Content listing for the Visual Cafe online help. The online help provides task and context sensitive information.

## Java API Reference

A reference of all Java packages, classes and interfaces for each package. Variables, constructors, and methods are also included.

## Java Language Reference

Opens the Content listing for the Java Language Reference.

## Macro Reference

Opens the Content listing for the Macro Reference.

## **Visual Cafe Demo**

Displays an online demo of Visual Cafe.

## **On the Web**

Starts your default browser and takes you to Internet links that you might find useful.

## **About Visual Cafe**

Displays version and copyright information for this release.

## **LiveUpdate**

Helps you update your software across the Internet through a Symantec Web site. Before you can use LiveUpdate, you must register at the Symantec update center; after registering, you are given a file that enables LiveUpdate.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window File menu:

**New File**

Opens the New File dialog box so you can select the type of file to create.

**Open**

Opens the Open File dialog box so you can select and open a Visual Cafe project (**.vep**), HTML file, Java source file (**.java**), or a text file.

**Close *item***

Closes the active file, view, or project. The menu option reflects the name of the active object. The default command is “close project.” When a project closes, all windows associated with that project also close. Visual Cafe prompts you about any unsaved changes.

**New Project**

Opens the New Project dialog box so you can create and open a new [project](#).

**Open Project**

Opens the Open File dialog box so you can select and open a Visual Cafe project (**.vep**).

**Close Project**

Closes the active project.

**Save *item*, Save As, Save All**

Saves files to disk. Functionality is the same as MS Windows. If no window is open, the default is “Save Project.”

**Add Component to Library**

Opens a dialog box where you can add an external component to the Component Library.

**Print Setup, Print**

Configure printing and print active window. Functionality is the same as MS Windows.

**Recently Opened Files**

Lists recently opened files. Choose a file to open it.

**Exit**

Closes any open projects and files after prompting to save unsaved changes, then closes Visual Cafe.

## New Project

Opens the New Project dialog box so you can create and open a new [project](#).

## **New File**

Opens the New File dialog box so you can select the type of file to create.

## Open

Opens the Open File dialog box so you can select and open a Visual Cafe project (**.vep**), HTML file, Java source file (**.java**), or a text file.

**Close *item***

Closes the active file, view, or project. The menu option may reflect the name of the active object. When a project closes, all windows associated with that project also close. Visual Cafe prompts you about any unsaved changes.

**Save *item***

Save the selected item. Functionality is the same as MS Windows. If no window is open, the default is “Save Project.”

## **Save As**

Allows you to save the selected item with a new name. Functionality is the same as MS Windows. If no window is open, the default is "Save Project."

## **Save All**

Saves all open items. Functionality is the same as MS Windows. If no window is open, the default is “Save Project.”

## **Add Component to Library**

Opens a dialog box where you can add an external component to the Component Library.

## **Print Setup**

Configure printing. Functionality is the same as MS Windows.

## **Print**

Print active window. Functionality is the same as MS Windows.

## Recently Opened Files

Lists recently opened files. Choose a file to open it.

## **Activate Project**

Makes the selected open project active.

## **Exit**

Closes any open projects and files after prompting to save unsaved changes, then closes Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Visual Cafe Window Edit menu provides standard system edit commands such as undo, cut, copy, paste, and delete.

Shortcuts for Edit commands:

Undo	CONTROL + Z
Cut	CONTROL + X
Copy	CONTROL + C
Paste	CONTROL + V
Delete	DELETE
Select All	CONTROL + A

The Select All command has this effect:

<u>If the active window is</u>	<u>Selects all</u>
Form Designer	components
Source window	source in the current method
Breakpoint window	breakpoints in the list

## Cut/Copy/Paste

Same functionality as MS Window/NT.

Shortcut:

Cut	CONTROL + X
Copy	CONTROL + C
Paste	CONTROL + V

## **Undo**

Undoes the last edit. Same functionality as MS Window/NT.

Shortcut: CONTROL + Z

**Edit ▶ Delete  
Breakpoint**

**▶ Clear**

Deletes the selected text, or clears the selected breakpoint. Same functionality as MS Window/NT.

Shortcut: DELETE

## Select All

The Select All command has this effect:

<u>If the active window is</u>	<u>Selects all</u>
Form Designer	components
Source window	source in the current method
Breakpoint window	breakpoints in the list

Shortcut: CONTROL + A

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The View menu lets you change what windows are displayed in the Visual Cafe window.

You can activate the display of these tools:

Property List	Component Library
Hierarchy Editor	Class Browser
Javadoc Viewer	

You can activate the display of these debugging windows:

Watch	Breakpoint
Messages	Threads
Variables	Call Stack

You can also toggle the display of the Status Bar and Workbook tabs (for MDI).

#### **Status Bar**

Toggle the display of the Status Bar at the bottom of the Visual Cafe main window.

#### **Workbook**

Toggle the display of Workbook tabs at the bottom of the workspace area (MDI mode).

#### **Project**

Allows you to select an open project as the new active main project. The main project is the one that the Project menu commands apply to.

## **Status Bar**

Toggle the display of the Status Bar at the bottom of the Visual Cafe main window.

## **Workbook**

Toggle the display of Workbook tabs at the bottom of the workspace area (MDI mode).

## Open the selected view

You can activate the display of these views:

Property List	Component Library
Hierarchy Editor	Class Browser

## Open the selected window

You can activate the display of these debugging windows:

Watch	Breakpoint
Messages	Threads
Variables	Call Stack

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Project menu:

### Execute

Compiles and runs the current project with no debugging. Applets are run in the Symantec Applet viewer by default. You set the default viewer in the Project tab of the Project Options dialog box.

### Run in Debugger

Runs the program in debug mode and stops at the first breakpoint.

### Step Into

Begins running the program by stepping into the first line of source code. When used on an applet, this command takes you the applet `void init()` method if one is implemented.

### Debug in Waiting VM

Lets you attach to a process you started outside of the Visual Cafe environment so you can debug. See the remote debugging instructions for more information.

### Build Applet/Application

Builds the applet or application.

### Compile Item

Compiles the active source file.

### Rebuild All

Builds all files in the active project.

### Parse All

Parses all files in the project.

### Produce Javadoc

Produces Javadoc documentation for a project.

### Update Project Beans

Updates instances of top-level beans in your project, which is useful for testing beans during development.

### Add Item

### Remove Item

Adds or removes the active file in the Source window to the project.

### Create Project Template

Opens the Create Template dialog box so you can add a template to the Component Library. Once added, the template is available for the New Project command.

### AutoJAR

Compiles your project and creates a JAR file from it.

### Deploy

Compiles your project and deploys it according to your project option settings.

**Options**

Opens the Project Options dialog box where you can define debugger, compiler, project, version control, deployment, and directory options.

## **Execute or Compile**

Compiles and runs the current project with no debugging. Applets are run in the Symantec Applet viewer by default. You set the default viewer in the Project tab of the Project Options dialog box.

**Project ▶ Run in Debugger  
Debug**

**▶ Continue**

Runs the program in debug mode and stops at the first breakpoint.

## Step Into

Begins running the program by stepping into the first line of source code. When used on an applet, this command takes you the applet `void init()` method if one is implemented.

## **Debug in Waiting VM**

Lets you attach to a process you started outside of the Visual Cafe environment so you can debug. See the remote debugging instructions for more information.

## **Build Applet/Application**

Builds the applet or application.

## **Rebuild All**

Builds all files in the active project.

## **Compile *Item***

Compiles the active source file.

## **Parse All**

Parses all files in the project.

## Parse

Parses the selected file.

## **Produce Javadoc**

Produces Javadoc documentation for a project.

## **Update Project Beans**

Updates instances of top-level beans in your project, which is useful for testing beans during development.

## JAR

Helps you create a Java Archive (JAR) file from the active project.

**Add *Item***  
**Remove *Item***

Adds or removes the active file in the Source window to the project.

## Create Project Template

Opens the Create Template dialog box so you can add a template to the Component Library. Once added, the template is available for the New Project command.

## **Switch Project**

Allows you to select an open project as the new active main project. The main project is the one that the Project menu commands apply to.

## Options

Opens the Project Options dialog box where you can define debugger, compiler, project, version control, deployment, and directory options.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Insert menu. This menu allows you to create new project objects and add forms, files, and objects to the project.

#### **Form**

Opens the Insert Form dialog box so that you can insert a form from the Component Library into the current project.

#### **Applet**

Inserts an applet if there are no saved applet objects, or opens the Insert Applet dialog box where you can select an applet from a list of applet templates.

#### **Component**

Opens the Insert Object dialog box so you can add one or more objects from the Component Library to the active form or to the project.

#### **JavaBean**

Start the JavaBean Wizard, which helps you create a new bean. The bean is inserted in the active project.

#### **Class**

Opens the Insert Class Wizard, where you can add and define a new class or interface for the current project.

#### **Member**

Opens the Insert Member dialog box, where you can declare a method to be added to the current class.

#### **Group**

Adds a group to the Component Library. Groups can only be inserted at the root level of the Component Library window or inside other groups.

#### **Files into Project (Insert/Remove Files)**

Opens a dialog box where you can select one or more files to be added to the project. You can also remove files from the project with this dialog box.

## **Insert Form**

Opens the Insert Form dialog box so that you can insert a form from the Component Library into the current project.

## **Insert Applet**

Inserts an applet if there are no saved applet objects, or opens the Insert Applet dialog box where you can select an applet from a list of applet templates.

## **Insert Component**

Opens the Insert Component dialog box so you can add one or more objects from the Component Library to the active form or to the project.

## **Insert Class**

Opens the Insert Class Wizard, where you can add and define a new class or interface for the current project.

## **Insert Member**

Opens the Insert Member dialog box, where you can declare a method to be added to the current class.

## **Insert Group**

Adds a group to the Component Library. Groups can only be inserted at the root level of the Component Library window or inside other groups.

## **Files into Project**

Opens a dialog box where you can select one or more files to be added to the project. You can also remove files from the project with this dialog box.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Tools menu:

#### **Compare Files**

Opens the Compare Files dialog box, where you can specify two file to compare. The differences are presented in editing windows, allowing you to view the lines that are different.

#### **Macro**

As you use the Source editor to create and edit program code, you may find yourself performing some tasks again and again. You can automate the task by turning on the Macro Recorder and having it create a macro for you.

**Record macro** Choose this command before beginning a task in an editing window. Your keystrokes and mouse actions are then recorded. After completing the task, choose Stop Recording from the Macro Menu. You can save the macro for later use.

**Play** After creating a macro, repeat the task you recorded by playing the macro.

**ScriptMaker** Use the ScriptMaker dialog box to copy, name, and edit macros.

#### **Migrate 1.0 to 1.1**

Helps you migrate the active source file from the JDK 1.0 event model to the JDK 1.1 event model.

#### **JAR Viewer**

Opens a JAR Viewer tool that you can use within its environment to quickly view JAR files.

#### **Localization**

Choose a localization menu item to help you localize files in a project, edit resource bundles, add and delete locales, and choose a working locale.

#### **Version Control**

Choose a menu item to use a version control system with the active project.

#### **Environment Options**

Opens the Environment Options dialog box so that you can customize your Visual Cafe development environment. The options are global and apply to all projects.

## **Compare Files**

Opens the Compare Files dialog box, where you can specify two file to compare. The differences are presented in editing windows, allowing you to view the lines that are different.

## Macro ► Record macro

As you use the Source editor to create and edit program code, you may find yourself performing some tasks again and again. You can automate the task by turning on the Macro Recorder and having it create a macro for you.

**Record macro** Choose this command before beginning a task in an editing window. Your keystrokes and mouse actions are then recorded. After completing the task, choose Stop Recording from the Macro Menu. You can save the macro for later use.

## Macro ▶ Play

As you use the Source editor to create and edit program code, you may find yourself performing some tasks again and again. You can automate the task by turning on the Macro Recorder and having it create a macro for you.

**Play** After creating a macro, repeat the task you recorded by playing the macro.

## Macro ▶ ScriptMaker

Use the ScriptMaker dialog box to copy, name, and edit macros.

## Macros you can choose.

For example:

### Macro ▶ Comment block

A built-in macro that inserts a comment code block at the cursor insertion point in the Source window.

### Macro Save/close source windows

A built-in macro that saves the contents of open Source windows and then closes them.

## Macro Comment block

A built-in macro that inserts a comment code block at the cursor insertion point in the Source window.

## Macro **Save/close source windows**

A built-in macro that saves the contents of open Source windows and then closes them.

## **Migrate 1.0 to 1.1**

Helps you migrate the active source file from the JDK version 1.0 to the JDK version 1.1 event model.

## JAR Viewer

Look at the contents of a JAR file.

## **Environment Options**

Opens the Environment Options dialog box so that you can customize your Visual Cafe development environment. The options are global and apply to all projects.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window menu:

#### **New Window**

Creates a new instance of the active Source window. Each new instance of the same window is incremented to indicate the number of open windows.

#### **Docking View**

Toggles whether a dockable window can be docked. (MDI development environment)

#### **Workspaces**

##### ***workspace names***

Selecting a workspace from the list saves your current workspace and configures your environment to the new layout.

<b>New</b>	allows you to create a new workspace.
<b>Rename</b>	allows you to rename a workspace.
<b>Delete</b>	allows you to delete a workspace.

#### **Next or Previous**

Activates the next or previous window in the workspace area.

#### **Cascade, Tile Horizontally, Tile Vertically**

Arranges the windows in the workspace area in a cascaded or tiled orientation. (MDI development environment)

#### **Windows**

Displays a list of recently used windows.

## **New Window**

Creates a new instance of the active source window. Each new instance of the same window is incremented to indicate the number of open windows.

## Docking View

Toggles whether a dockable window can be docked. (MDI development environment)

## **Next or Previous**

Activate the next or previous window in the workspace area. (MDI development environment)

## **Cascade, Tile Horizontally, or Tile Vertically**

Arrange the windows in the workspace area in a cascaded or tiled orientation. (MDI development environment)

## Workspaces

### ***workspace names***

Selecting a workspace from the list saves your current workspace and configures your environment to the new layout.

<b>Edit</b>	refers to the development workspace.
<b>Debug</b>	refers to the debug workspace.
<b>New</b>	allows you to create a new workspace.
<b>Rename</b>	allows you to rename a workspace.
<b>Delete</b>	allows you to delete a workspace.

## Recently Used Windows

To display one of the windows that you used recently, you can select the window name from the numbered list on the menu.

## **Windows**

Displays a list of recently used windows.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Object menu enables when the active window is the Project window or Form Designer. The following options are available from the Visual Cafe Window Object menu:

**Edit *object***

Opens the selected item in its corresponding editor. This command is enabled only when the selected object is a form or visual object. The menu name changes to reflect the type of object selected.

**Edit Source**

Opens the Source window for the selected object. Allows you to view the code associated with an object.

**Add Interaction**

Opens the Interaction Wizard where you can create an interaction for objects in your project. This command uses the selected object as the trigger component.

**Bind Event**

Opens the Event Binding dialog box for the selected object. Helps you bind events to that object.

**Add to Library**

Opens the Add to Library dialog box, where you can save the selected object in the Component Library.

**Customize**

Opens a customizer so you can configure the characteristics or behavior of a bean. Not all beans have customizers.

**Edit *object***

Opens the selected item in its corresponding editor. This command is enabled only when the selected object is a form or visual object. The menu name changes to reflect the type of object selected.

## **Edit Source or Open**

Opens the Source window for the selected object or file. Allows you to view the code associated with an object or file.

## **Bind Event**

Opens the Event Binding dialog box for the selected object. Helps you bind events to that object.

## **Add Interaction**

Opens the Interaction Wizard where you can create an interaction for objects in your project. This command uses the selected object as the trigger component.

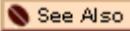
## **Add to Library**

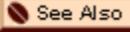
Opens the Add to Library dialog box, where you can save the selected object in the Component Library.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

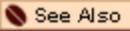


The following options are available from the Visual Cafe Window Layout menu. Alignment commands are available only when you do not specify a Layout manager for the form.

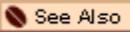
**Interactions**  **Show Interactions From/To/To and From Interactions**

 **Show All/Hide All**

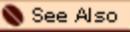
Shows interactions from, to, or to and from the select component, or shows all or no interactions for the entire form. Interactions are represented by a line and arrow.

**Align**  **Right/Left/Bottom/Top Edges**

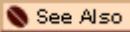
Aligns the selected objects relative to the specified edge. You can use the Form Designer's pop-up menu to align object to the left and top edges.

**Align**  **To Grid**

Aligns the upper left corner of the selected object(s) to the nearest grid point.

**Center**  **Vertically/Horizontally**

Centers the selected object in the form vertically or horizontally. If multiple objects are selected, the relative position of the objects to each other is maintained.

**Space Evenly**  **Vertically/Horizontally**

Spaces the selected objects evenly on a vertical or horizontal axis.

**Make Same Size**  **Vertical/Horizontal/Both**

Resizes the selected objects' vertical and/or horizontal dimensions based on the first selected object.

**Look and Feel**  **Metal/ CDF Motif / Windows**

Lets you choose a JFC Swing look-and-feel to display in the Form Designer.

**Bring to Front**

Brings the selected object to the front of other objects on the form.

**Send to Back**

Puts the selected object behind other objects on the form.

**Grid Options**

Opens the Grid Options dialog box, where you can change the display, size of the Form Designer grid, and select or clear "snap to grid" and "show grid."

**Edit Constraints**

Opens the Gridbag Constraints Editor that lets you dynamically modify the properties of this layout manager on your form.

**Invisibles**

Toggles the display of invisible components on a form. The default state for invisible objects is visible.

**Borders**

When selected, displays a dotted line border around components displayed in the Form Designer.

**Interactions**  **Show Interactions From/To/To and From Interactions**

 **Show All/Hide All**

Shows interactions from, to, or to and from the select component, or shows all or no interactions for the entire form.  
Interactions are represented by a line and arrow.

## Look and Feel Metal/ CDF Motif / Windows

Lets you choose a JFC Swing look-and-feel to display in the Form Designer.

## **Align** **Right/Left/Bottom/Top Edges**

Aligns the selected objects relative to the specified edge. You can use the Form Designer's pop-up menu to align object to the left and top edges.

## **Align See Also To Grid**

Aligns the upper left corner of the selected object(s) to the nearest grid point.

## Center Vertically/Horizontally

Centers the selected object in the form vertically or horizontally. If multiple objects are selected, the relative position of the objects to each other is maintained.

## **Space Evenly** **Vertically/Horizontally**

Spaces the selected objects evenly on a vertical or horizontal axis.

## **Make Same Size** **Vertical/Horizontal/Both**

Resizes the selected objects' vertical and/or horizontal dimensions based on the first selected object.

## **Bring to Front**

Brings the selected object to the front of other objects on the form.

## **Send to Back**

Puts the selected object behind other objects on the form.

## **Grid Options**

Opens the Grid Options dialog box, where you can change the display, size of the Form Designer grid, and select or clear "snap to grid" and "show grid."

## **Edit Constraints**

Opens the Gridbag Constraints Editor that lets you dynamically modify the properties of this layout manager on your form.

## Invisibles

Toggles the display of invisible components on a form. The default state for invisible objects is visible.

## **Borders**

When selected, displays a dotted line border around components displayed in the Form Designer.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Search menu. These commands are useful when trying to find a string in multiple files, comparing files, and editing code in the Source window and Class Browser edit pane.

#### **Find**

Opens the Find dialog box, where you can enter search criteria for the file in the active editing window.

#### **Find Again**

Finds the next occurrence of the text string previously defined with the Find command.

#### **Replace**

Opens the Replace dialog box, where you can specify a search string and replacement text for the file in the active editing window.

#### **Find in Files**

Opens the Find in Files dialog box, where you can search for text across multiple files. The scope of the file search can be defined.

#### **Bookmarks**

Opens the Bookmark dialog box, where you can add, remove, or go to a bookmark. You can set and move to as many as ten different locations in your source files. Bookmarks are saved through the current Visual Cafe session only.

#### **Go to Buffer**

Opens the Go to Buffer dialog box, where you can change the options for the current edit buffer or those of another.

#### **Go to Line**

Opens the Go to Line window where you can specify a line number to move to. The Source pane is scrolled to the requested line. If any text is currently selected, the selection is extended to include that line.

#### **Go to Function**

Opens the Go to Function window so that you can select an available function from the list. The edit window's focus is moved to the selected function location.

#### **Go to Definition**

Opens the selected method or data member in the Class Browser. The associated code block displays in the editing pane. If there are multiple occurrences of the selection in the source file, then the Members window displays for a selection.

#### **Go to Matching Delimiter**

Finds the delimiter that matches the delimiter to the right of the current insertion point. The insertion point is moved to the front of the matching delimiter. This command can find matching parentheses, square brackets, or braces.

#### **Go to Current Error**

#### **Go to First Error**

#### **Go to Previous Error**

#### **Go to Next Error**

The Go to error commands moves the editing window focus to the location of the corresponding error.

## **Find**

Opens the Find dialog box, where you can enter search criteria for the file in the active editing window.

## **Find Again**

Finds the next occurrence of the text string previously defined with the Find command.

## **Replace**

Opens the Replace dialog box, where you can specify a search string and replacement text for the file in the active editing window.

## **Find in Files**

Opens the Find in Files dialog box, where you can search for text across multiple files. The scope of the file search can be defined.

## **Bookmarks**

Opens the Bookmark dialog box, where you can add, remove, or go to a bookmark. You can set and move to as many as ten different locations in your source files. Bookmarks are saved through the current Visual Cafe session only.

## **Go to Buffer**

Opens the Go to Buffer dialog box, where you can change the options for the current edit buffer or those of another.

## Go to Line

Opens the Go to Line window where you can specify a line number to move to. The Source pane is scrolled to the requested line. If any text is currently selected, the selection is extended to include that line.

## **Go to Function**

Opens the Go to Function window so that you can select an available function from the list. The edit window's focus is moved to the selected function location.

## **Go to Definition**

Opens the selected method or data member in the Class Browser. The associated code block displays in the editing pane. If there are multiple occurrences of the selection in the source file, then the Members window displays for a selection.

## **Go to Matching Delimiter**

Finds the delimiter that matches the delimiter to the right of the current insertion point. The insertion point is moved to the front of the matching delimiter. This command can find matching parentheses, square brackets, or braces.

**Go to Current Error**  
**Go to First Error**  
**Go to Previous Error**  
**Go to Next Error**

The Go to error commands moves the editing window focus to the location of the corresponding error.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are enabled when the Source window is the current window. These commands affect the debug setting or cursor placement in the Source window.

#### **Evaluate Expression**

Lets you evaluate variables and expressions from a dialog box, modify their values, and add an entry to the Watch window.

#### **Toggle Breakpoint**

Sets or clears a breakpoint on the current line. The toggling is based on whether or not a breakpoint is set for the current line of source code.

If no breakpoint is set, a breakpoint is set on the current line and a stop icon appears in the left margin next to the current line. If there is a breakpoint on the current line, the breakpoint is removed.

#### **Set Conditional Breakpoint**

Opens the Conditional Breakpoint dialog box, where you can set a breakpoint that occurs if a particular expression evaluates to true.

#### **Code Helper**

Starts the Code Helper at the insertion point in code, if possible.

#### **Edit Javadoc Comments**

Opens the Javadoc Editor, which helps you view and create Javadoc comments in your source code.

#### **Indent/Unindent**

Indents or unindents the selected text block.

#### **Uppercase/Lowercase**

Converts the selected text to upper or lower case.

#### **Tabs to Spaces**

Changes all tab characters in the selected text to spaces. The number of spaces used to replace each tab character depends on the current edit buffer's Tab Width value in the Format Options dialog box.

#### **Spaces to Tabs**

Changes spaces in the selected text to tab characters. The number of spaces used to create each tab character depends on the current edit buffer's Tab Width value in the Format Options dialog box.

#### **Format Options**

Opens the Format Options dialog box, where you can set editing options for the current edit buffers.

## Evaluate Expression

Lets you evaluate variables and expressions from a dialog box, modify their values, and add an entry to the Watch window.

## **Add Watch**

Adds an entry to the Watch window for the selected variable.

## **Toggle Breakpoint**

Sets or clears a breakpoint on the current line. The toggling is based on whether or not a breakpoint is set for the current line of source code.

If no breakpoint is set, a breakpoint is set on the current line and a stop icon appears in the left margin next to the current line. If there is a breakpoint on the current line, the breakpoint is removed.

## **Set Conditional Breakpoint**

Opens the Conditional Breakpoint dialog box, where you can set a breakpoint that occurs if a particular expression evaluates to true.

## **Indent/Unindent**

Indents or unindents the selected text block.

## Uppercase/Lowercase

Converts the selected text to upper or lower case.

## **Tabs to Spaces**

Changes all tab characters in the selected text to spaces. The number of spaces used to replace each tab character depends on the current edit buffer's Tab Width value in the Format Options dialog box.

## **Spaces to Tabs**

Changes spaces in the selected text to tab characters. The number of spaces used to create each tab character depends on the current edit buffer's Tab Width value in the Format Options dialog box.

## **Format Options**

Opens the Format Options dialog box, where you can set editing options for the current edit buffers.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Debug menu. The Debug menu replaces the Project menu when you start debugging.

#### **Continue**

Runs a paused program.

#### **Pause**

Temporarily stops the execution of a program while it is running and switches to debug mode. To begin the program again at the current location, choose Continue.

When an unhandled exception is encountered, Visual Cafe pauses automatically at the line where the error occurred.

#### **Stop**

Terminates the program execution.

#### **Restart**

Restarts the program. Debugging is restarted from the first line.

#### **Step Into**

Steps into the next line of source code. This command steps to the next source code statement even if it is contained within a method.

#### **Step Over**

Steps over the current method and stops when the method returns. Executes the program to the next statement, unless a breakpoint or exception is encountered before the next statement. If the current statement contains a method call, the entire method is called before control is returned.

#### **Step Out**

Executes the current method until it returns to its caller, unless a breakpoint or exception is encountered before execution reaches that point.

#### **Continue to Cursor**

Continues running a paused program while ignoring any breakpoints prior to the cursor location, then stops at the cursor location. When the cursor is reached, the program pauses and the debugger is invoked. If the selected line does not get executed before the end of the program, the program does not break.

#### **Continue to End**

Continues running a paused program, ignoring all breakpoints, from the pause point until the normal termination point. If any type of exception occurs, the program breaks at the point where the exception is called.

#### **Options**

Opens the Project Options dialog box, where you can set debugging options under the Debugger tab.

#### **Update Now**

Forces an incremental update and saves all files. (Professional edition only)

#### **Restart Method**

Acts like a "Pop" command. It takes you back to the method that called the currently active method. You should not think of it like an undo command, because it cannot undo some edits, such as variable edits. It does not undo side effects of code that was run; an example could be if part of a program runs two times and causes an exit. (Professional edition only)

**Continue**

Runs a paused program.

## **Pause**

Temporarily stops the execution of a program while it is running and switches to debug mode. To begin the program again at the current location, choose Continue.

When an un-handled exception is encountered, Visual Cafe pauses automatically at the line where the error occurred.

## **Stop**

Terminates the program execution.

## **Restart**

Restarts the program. Debugging is restarted from the first line.

## Step Into

Steps into the next line of source code. This command steps to the next source code statement even if it is contained within a method.

## **Step Over**

Steps over the current method and stops when the method returns. Executes the program to the next statement, unless a breakpoint or exception is encountered before the next statement. If the current statement contains a method call, the entire method is called before control is returned.

## Step Out

Executes the current method until it returns to its caller, unless a breakpoint or exception is encountered before execution reaches that point.

## **Continue to Cursor**

Continues running a paused program while ignoring any breakpoints prior to the cursor location, then stops at the cursor location. When the cursor is reached, the program pauses and the debugger is invoked. If the selected line does not get executed before the end of the program, the program does not break.

## **Continue to End**

Continues running a paused program, ignoring all breakpoints, from the pause point until the normal termination point. If any type of exception occurs, the program breaks at the point where the exception is called.

## **Update Now**

Forces an incremental update and saves all files. (Professional edition only)

## **Restart Method**

Acts like a “Pop” command. It takes you back to the method that called the currently active method. You should not think of it like an undo command, because it cannot undo some edits, such as variable edits. It does not undo side effects of code that was run; an example could be if part of a program runs two times and causes an exit. (Professional edition only)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The following option is available from the Visual Cafe Window Variables menu.

**Add Watch**

Takes the selected item in the source window, or current token if there is no selection and adds an entry to the Watch window.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The breakpoint menu is available when your applet or application is running. These commands are also available from the Breakpoint window's pop-up menu.

**Clear/Clear all**

Clears the selected breakpoint or all breakpoints.

**Enable/Enable all**

Enables the selected breakpoints or all breakpoints.

**Disable/Disable all**

Disables the selected breakpoints or all breakpoints.

**Go to Source**

Opens the Source window with focus on the line of code where the breakpoint is set.

**Clear/Clear all**

Clears the selected breakpoint or all breakpoints.

**Enable/Enable all**

Enables the selected breakpoints or all breakpoints.

**Disable/Disable all**

Disables the selected breakpoints or all breakpoints.

## **Go to Source**

Opens the Source window with focus on the line of code where the breakpoint is set.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following options are available from the Visual Cafe Window Threads menu.

**Suspend**

Suspend the execution of the selected thread.

**Resume**

Resumes execution of the selected thread from the previous suspend point.

**Suspend others**

Suspends all other threads in the debugger except for the selected thread.

**Resume others**

Resumes all other threads in the debugger except for the currently selected thread.

**Set Focus**

Sets the focus of the debugger to the selected thread. The Call Stack, Variables, and Source windows are updated.

## **Suspend thread**

Suspend the execution of the selected thread.

**Resume thread**

Resumes execution of the selected thread from the previous suspend point.

## **Suspend other threads**

Suspends all other threads in the debugger except for the selected thread.

## **Resume other threads**

Resumes all other threads in the debugger except for the currently selected thread.

## **Set Focus**

Sets the focus of the debugger to the selected thread. The Call Stack, Variables, and Source windows are updated.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The following options are available from the Visual Cafe Window Call menu.

**View parameter values**

Toggles the display of parameter values in the Method column of the Calls window.

**View parameter types**

Toggles the display of parameter types in the Method column.

**Go to source**

Opens the Source window for the selected call.

**Go to Variables**

Updates and displays the Variables Window with a list of variables in the selected method call.

**Set Focus**

Set the focus of the debugger to the selected method call. This option also updates the Variables to show the active variables in the method and Source windows to show the method call so that you can step in if you want.

## **View parameter values**

Toggles the display of parameter values in the Method column of the Calls window.

## **View parameter types**

Toggles the display of parameter types in the Method column.

**Go to source**

Opens the Source window for the selected call.

## **Go to Variables**

Updates and displays the Variables Window with a list of variables in the selected method call.

## **Set Focus**

Set the focus of the debugger to the selected method call. This option also updates the Variables to show the active variables in the method and Source windows to show the method call so that you can step in if you want.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The following commands are available on the Hierarchy menu. Use the Hierarchy Editor's pop-up menu to delete a class's inheritance.

**Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

**Go to Source**

Opens the Source window containing the code for the selected package or class.

**View Imports**

In addition to the local classes, the Hierarchy Editor displays the classes that are imported into the project.

## **Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

## **Go to Source**

Opens the Source window containing the code for the selected package or class.

## **View Imports**

In addition to the local classes, the Hierarchy Editor displays the classes that are imported into the project.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The following options are available from the Visual Cafe Classes menu. This menu is available when the Class Browser is active. Use the Insert menu or pop-up menus in the Class Browser panes to add new classes and members.

**Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

**Member Attribute**

Opens the Member Attributes dialog box, where you can change the member's [access type](#) for the selected method or data element.

**Delete Member**

Deletes the selected member from its class. If the selected member is a variable, like "Button1", this command removes the associated visual object.

**Go to Source**

Opens the class' Java file in the Source window.

**Options**

Opens the Class Options dialog box, where you can expand or refine the elements in the class listing and define how classes or members are sorted.

## **Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

## Member Attribute

Opens the Member Attributes dialog box, where you can change the member's [access type](#) for the selected method or data element.

## **Delete Member**

Deletes the selected member from its class. If the selected member is a variable, like "Button1", this command removes the associated visual object.

## **Go to Source**

Opens the class' Java file in the Source window.

## Options

Opens the Class Options dialog box, where you can expand or refine the elements in the class listing and define how classes or members are sorted.

## Popup Menus

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click a component in the Form Designer, Visual Cafe displays this menu:

**Cut/Copy/Paste**

These commands perform standard Windows functions.

**Align Left Edges**

**Align Top Edges**

**Align to Grid**

Aligns the selected object(s) as indicated. Use the Form Designer's pop-up menu to align object to their relative right and bottom edges. See the Layout menu topic for information about aligning to a grid.

These commands are disabled if the selected controls reside inside a container that is using a [layout](#).

**Edit Source**

Opens the Source window for the component.

**Add Interaction**

Opens the Interaction Wizard. The selected component is used as the interaction's trigger component.

**Add to Library**

Opens the Add to Library dialog box, where you can add the selected component to the Component Library making it reusable across projects.

**Properties**

Opens the Property List for the selected object.

## **Cut/Copy/Paste**

These commands perform standard Windows functions.

## **Edit Source or Open**

Opens the Source window for the component.

## **Add Interaction**

Opens the Interaction Wizard. The selected component is used as the interaction's trigger component.

## **Add to Library**

Opens the Add to Library dialog box, where you can add the selected component to the Component Library making it reusable across projects.

## Properties

Opens the Property List for the selected object.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click an object in the Project window, Visual Cafe displays a pop-up menu with the following commands:

**Go to Definition**

Opens the selected component's parent class in the Class Browser and shows the object declaration in the source pane.

**Insert *object***

This Insert command reflects the selected object. This is a quick way of creating a new object of the selected type.

**Insert Component**

Opens the Insert Component dialog box, where you can select a component from the Component Library for insertion into the current project.

**Edit *object***

This edit command reflects the selected object. The appropriate editor is opened for the selection.

**Edit Source**

Opens the selected object's source code in the Source window.

**Add Interaction**

Starts the Interaction Wizard so that you can define an interaction relationship between components. The selected object is the trigger component for the new interaction.

**Properties**

Opens the Property List for the selected object.

**Docking View**

Enable or disable docking of the window. (MDI development environment)

**Hide or Close**

Hide or close the window.

## **Go to Definition**

Opens the selected component's parent class in the Class Browser and shows the object declaration in the source pane.

## **Insert *object***

This Insert command reflects the selected object. This is a quick way of creating a new object of the selected type.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click an object in the Project window, Visual Cafe displays a pop-up menu with the following commands:

**Insert/Remove Files**

Opens the Project Files dialog box, where you can add and remove files to and from the current project.

**Edit Source or Open**

Opens the selected source file in the Source window.

**Docking View**

Enable or disable docking of the window. (MDI development environment)

**Hide or Close**

Hide or close the window.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click an object in the Project window, Visual Cafe displays a pop-up menu with the following commands:

**Compile**

Compiles and runs the current project with no debugging. Applets are run in the Symantec Applet viewer by default. You set the default viewer in the Project tab of the Project Options dialog box.

**Parse**

Parses the selected file.

**Insert/Remove Files**

Opens the Project Files dialog box, where you can add and remove files to and from the current project.

**Stop or Start RAD**

Disable or enable RAD for a file. The RAD feature makes objects appear in the Objects view, lets you design forms in the Form Designer, lets you design menus in the Menu Designer, lets you create interactions with the Interaction Wizard, and generates code according to your form design.

**Edit Source or Open**

Opens the selected source file in the Source window.

**Docking View**

Enable or disable docking of the window. (MDI development environment)

**Hide or Close**

Hide or close the window.

## **Insert/Remove Files**

Opens the Project Files dialog box, where you can add and remove files to and from the current project.

## Hide or Close

Hide or close the window.

## **Stop or Start RAD**

Disable or enable RAD for a file. The RAD feature makes objects appear in the Objects view, lets you design forms in the Form Designer, and generates code according to your form design.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click a tab in the Project window, Visual Cafe displays a pop-up menu with the following commands:

**Objects**

Enable or disable the display of the Objects tab.

**Packages**

Enable or disable the display of the Packages tab.

**Files**

Enable or disable the display of the Packages tab.

**Make *Tab* Default**

Enable or disable the tab as your default tab when a project first displays.

## Objects

Enable or disable the display of the Objects tab.

## Packages

Enable or disable the display of the Packages tab.

## Files

Enable or disable the display of the Packages tab.

## **Make *Tab* Default**

Enable or disable the tab as your default tab when a project first displays.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



When you right mouse click an object in the Class Browser's Classes pane, Visual Cafe displays a pop-up menu with these commands:

**Insert Class**

Opens the Insert Class Wizard, where you can add and define a new class or interface for the project.

**Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

**Go to Source**

Opens the selected class Java source file in the Source window.

**Options**

Opens the Class Options dialog box, where you can expand or refine the elements in the class listing, and define how classes or members are sorted.

## **Insert Class**

Opens the Insert Class Wizard, where you can add and define a new class or interface for the project.

## Options

Opens the Class Options dialog box, where you can expand or refine the elements in the class listing, and define how classes or members are sorted.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



When you right mouse click an object in the Class Browser's Member pane, Visual Cafe displays a pop-up menu with these commands:

**Insert Member**

Opens the Insert Member dialog box, where you can add a new method or variable to the selected class.

**Delete Member**

Deletes the selected method or data member. The member's declaration is removed from the source files.

**Member Attributes**

Opens the Member Attributes dialog box, where you can change a member's access type.

**Go to Source**

Opens the Source window with focus on the selected member.

**Options**

Opens the Class Options dialog box, where you can expand or refine the elements in the member listing, and define how members and classes are sorted and grouped.

## **Insert Member**

Opens the Insert Member dialog box, where you can add a new method or variable to the selected class.

## Delete Member

Deletes the selected method or data member. The member's declaration is removed from the source files.

## **Member Attributes**

Opens the Member Attributes dialog box, where you can change a member's access type.

## **Go to Source**

Opens the Source window with focus on the selected member.

## Options

Opens the Class Options dialog box, where you can expand or refine the elements in the member listing, and define how members and classes are sorted and grouped.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



When you right mouse click an object in the Project window, Visual Cafe displays a pop-up menu with these commands:

**Insert Class**

Opens the Insert Class Wizard, where you can add and define a new class or interface for the project.

**Edit Class**

Opens the Edit Class Wizard, where you can change the definition of a class or interface.

**Remove Inheritance**

This command is enabled when you select the line connecting two classes in the window.

The line represents a connection between two classes. When you delete the connection, the inheritance changes such that the object now extends `java.lang.object`.

**Go to Source**

Opens the Source window with focus on the selected package or class.

**View Imports**

In addition to the local classes, the Hierarchy Editor displays the classes that are imported into the project.

## **Remove Inheritance**

This command is enabled when you select the line connecting two classes in the window.

The line represents a connection between two classes. When you delete the connection, the inheritance changes such that the object now extends `java.lang.Object`.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



When you right mouse click an object in the Component Library, Visual Cafe displays a pop-up menu with these commands:

**Insert Group**

Creates a new group item.

**Add to Palette**

Adds the selected object(s) to the active Palette tab at the end of the object list.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The following commands are available from the pop-up menu when editing source code in the Source window, as well as the Class Browser's editing pane.

**Continue to Cursor**

Continues running a paused program until the programmer reaches the selected cursor position.

**Cut/Copy/Paste**

These edit commands perform standard Windows functions.

**Go to Definition**

Opens the selected method or data member in the Class Browser. The associated code block displays in the editing pane. If there are multiple occurrences of the selection in the source file, then the Members window displays for a selection.

**Evaluate Expression**

Lets you evaluate variables and expressions from a dialog box, modify their values, and add an entry to the Watch window.

**Set Breakpoint**

Sets a breakpoint at the current cursor location.

**Format Options**

Opens the Format Options dialog box, where you can set editing options for the current edit buffers.

## Toolbars

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe provides an extensive set of toolbars. Toolbars can be docked in the Visual Cafe Window or floated on the screen.

These toolbars are available in the Visual Cafe window:

- Standard
- Layout
- Views
- Debug
- Workspace

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Standard toolbar provides icons for tasks that you perform when using Visual Cafe. The icons execute standard Windows tasks.

From the standard toolbar, you can:

- create a new source file
- open an existing document
- save the active document
- cut/copy/paste
- print source code
- display information about Visual Cafe
- activate context sensitive help

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Layout toolbar provides quick access to the common commands that you use when using the Form Designer.

The icons on the Layout toolbar perform standard Windows graphic object layout tasks.

From the Form Designer, you can:

- Align objects by their left/right edges or top/bottom edges
- Center objects vertically or horizontally in the form
- Space objects vertically and horizontally
- Size objects equally vertically, horizontally, or both
- Show the grid on or off
- Show the display of [invisible objects](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Views toolbar gives you quick access to different views into your project and debugging tasks.

From the Views toolbar, you can access these windows:

- Class Browser
- Hierarchy Editor
- Property List
- Breakpoints
- Variables
- Watch
- Threads
- Call Stack
- Messages

#### Tasks

[Watching a variable or expression](#)

[Setting a breakpoint](#)

[Debugging threads](#)

[Viewing the current variable values](#)

[Viewing the call stack](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The Workspace toolbar allows you to select an existing workspace.

By default, there are two workspaces defined. Your workspaces may display differently than defined below because the workspace's state is saved as you close Visual Cafe tools.

**Edit**

Displays the Project window, Form Designer, and Property List windows.

**Debug**

Displays the Project, Variables, Call Stack, and Breakpoints windows.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The Debug toolbar provides easy access to the commands that you use most often when debugging your program.

**Tasks**

[Running a program in the Debugger](#)

[Pausing a program](#)

[Stopping a program](#)

[Stepping into a method](#)

[Stepping over a method](#)

[Stepping out of a Method](#)

[Toggling a breakpoint](#)

[Watching a variable](#)

## Tools and Window

## Dialog Boxes

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## **Object Add to Library**

The Add to Library dialog box allows you to add objects to the Component Library.

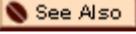
1. Select the group that the object will belong to.
2. Enter a name for the object in the Name field and a Description string.

The Description string is used when the object displays on the Palette or is listed in the Component Library window.

3. Click OK.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



**Project window, Packages tab**  **Insert/Remove files**

The Insert/Remove dialog box contains a list of files in the project. When a file is added, the file name is highlighted in the Project window.

**To remove a file**

1. Select the file to remove from the file pane at the bottom of the dialog box.
2. Click Remove or press the DELETE key.

**To add a file**

1. Locate and select the files to be added.
2. Click Add to add the selected files or click Add All to add all files in the current directory.

The file is added to the file pane at the bottom of the dialog box.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

**Project**  **Create Project Template**

The Create Project Template dialog box allows you to add a custom project templates to the Component Library.

#### **Tasks**

[Creating a template](#)

[Deleting Templates from the Component Library](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Source Set Conditional Breakpoint

You can set [conditional breakpoints](#) in any source window or pane. Breakpoints are marked in the editing window by a red dot in the left margin.

1. Specify the condition under which the breakpoint will be evaluated.

Option	Description
Always break	Always stop program execution at this breakpoint.
If expression is true	Use this option when you want the debugger to evaluate the associated expression every time this breakpoint is hit. A breakpoint is considered to be hit only if the expression is evaluated to true. This allows you to insert a statement in your code that you forgot, or to first test (in the debugger) that a certain modification does fix the problem before editing and compiling again.

2. Specify the location of the breakpoint.

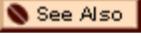
Option	Description
Method name	Name of the method that the breakpoint is set on.
Line number	The physical line number in the source code where the break should occur.

3. Click Ok.

The source editor determines if the breakpoint location is valid and then inserts the breakpoint.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



**Source**  **Evaluate Expression**

Use this dialog box to evaluate variables and expressions, and add an entry to the Watch window.

1. If needed, type a variable or expression in the Expression field.
2. To get the value of a variable or expression, click Recalculate or press RETURN. To change a value, type the new value in the Result list. To add a variable or expression to the Watch window, and close the dialog box, click Add Watch.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



**Property List**  **URL property/ImageList property**

Use this dialog box to enter a series of .gif and .jpg image files that are to be used with the associated component. The order of the images in this list is the order in which they display.

Reordering of the images can be done with the up and down arrows.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Insert Applet

The Insert Applet dialog box allows you to create the new applets from an existing [template](#) into an existing project.

The current default template is marked with an asterisk. If there are no templates in the default library, then a basic applet project is created.

To create an applet, select a template to be used as the applet base and click OK.

You can add applets to use as templates by adding an existing applet to the Project Template folder in the Component Library.

Standard view buttons are provided on the dialog box to change the display of the templates:



Shows the templates as large icons.



Shows the templates as small icons.



Shows the templates in a list.



Shows the templates in a single-column

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## **Insert Form**

This dialog box lets you to create a new form from one of the templates in the Component Library.

1. Select a template for the new form.
2. Click OK.

The form is added to the project and opened for editing in the Form Designer.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



## Insert Member

Use this window to add a new member to the selected class.

1. Specify the member declaration.

For data items, enter the type and member name (for example, `int nCats`). A trailing semicolon is optional. The member declaration is placed into the class declaration.



**Note** The Source File field is not editable. It displays the name of the source file into which the member definition will be placed.

2. Indicate the [base class access type](#).

This dialog box is also available with the Insert Member command on the Class Browser, Members pane pop-up menu.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Insert Component

The Insert Object dialog box allows you to add appropriate objects directly from the Component Library. The object list is filtered based on the current selection when the dialog box is opened.

Objects are filtered based on the current selection. For example, if a form is selected in the Project window when you open the Insert Object dialog box, then only the components that are valid within a form display.

You can also access the Insert Object dialog box from the Project window by choosing Insert Component from the pop-up menu.



**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

### Tasks

[Adding objects to a form](#)

[Adding objects to a project](#)

[Adding objects to the Palette](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## File New Project

The New Project dialog allows you to create a new object from an existing [template](#).

The current default template is marked with an asterisk. If there are no templates in the default library, then a basic applet project is created.

Standard view buttons are provided on the dialog box to change the display of the templates:



Shows the templates as large icons.



Shows the templates as small icons.



Shows the templates in a list.



Shows the templates in a single-column

### Tasks

[Creating a new project](#)

[Defining a new default template](#)

[Removing templates from the Component Library](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

**File**  **Open**

The Open File dialog box allows you to open Visual Cafe project file.

When you select a Visual Cafe project file (**.vep**), the project opens using the last saved window configuration.

When you select a Java source file (**.java**) or HTML file (**.html**), the Source window opens for source code editing.

 See Also


**Note** When you open a file, Visual Cafe does not automatically add the file to the current project.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



**Search**  **Go to Definition**

This window displays as a result of a successful search with the Go to Definition command.


1. In the Source window, select the definition that you want to search for within the file.  
For example, you can highlight the `init` keyword.
2. Choose Search  Go to Definition.  
The Members window lists all class methods in the source file that have the selected definition.
3. Double-click the appropriate method to open the Class Browser for the selected member's class.  
The selected method source code displays in the Class Browser editing pane.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp} **Using the ScriptMaker dialog box**



Use the ScriptMaker dialog box to copy and rename macros.

Use the File  Macro

 Record macro command to record your editing keystrokes. The file is saved as the <DEFAULT> macro. You can then use this dialog box to duplicate the macro and name the duplicate for saving. You cannot rename the default script; you must duplicate it first. If Default.java is deleted from disk, Visual Cafe creates a stub file when it starts up.


### **Macros**

Lists existing Java macros that are stored in \Bin\Macs\Src and have a run method. ScriptMaker macro files have the extension .java. The default macro is highlighted by default. Click a macro to select it. Double-click to open a macro in an editor.

### **Properties**

The display name and file name of the selected macro.

### **Display in Menu**

Select this option if you want the current (highlighted) macro to appear by name at the bottom of the Tools  Macro menu.

### **Reorder Commands**

Allows you to change the order of the macros that appear in the Macro menu. Click the up arrow to move the selected macro up in the menu; click the down arrow to move it down in the menu. The default macro always stays at the top of the list.

### **Done Button**

Click Done to save all changes to the macros.

### **Edit Button**

Click Edit to open a macro in an editor.

### **Rename Button**

Click Rename to rename the current macro. The Rename/Duplicate Macro dialog box appears. Rename is not active when the default macro is highlighted.

### **Duplicate Button**

Click Duplicate to create a copy of the current macro. The Rename/Duplicate Macro dialog box appears.

### **Delete Button**

Click Delete to delete the current macro. Delete is not active when the default macro is highlighted.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



**Property List**  **URL property or ImageURL property**

Use this dialog box to select a single .gif or .jpg image file to be used by the selected component. The image format must be compatible with the browser that will be running the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



**Window**  **Windows**

Use this dialog box to select a recently accessed tool or Project window. On this dialog box you can activate or close the selected window.

You can SHIFT-click multiple window names.

## Class Options Dialog box



## F1 help for standard Win95 windows

All of the help topics in this section use the direct helpid context string.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Use this dialog box to open the appropriate file for your current task.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Use this dialog box to save the active file or object.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Provides information about your Visual Cafe release.

Obsolete

## Database Edition F1 Help

Author note: They couldn't figure out a way to get this to work from the database help file, so you have to put it here.

## dbNAVIGATOR

Display the dbNAVIGATOR window. After connecting to a dbANYWHERE Server, you can use the dbNAVIGATOR as an easy way to view a list of dbANYWHERE servers and [data sources](#) that are currently connected, and to add database functionality to a project.

## Add Table

Start the Add Table wizard, which helps you add a table to your project. If your project already has a [QueryNavigator](#) or [RelationViewPlus](#), you can create a [master/detail relationship](#). If you are using the [JDBC API](#), you can create [joins](#) across [forms](#). Before choosing this command, in the Project window, select the [container](#) that you want to add the table to.



When you right-click in the dbNAVIGATOR window, a pop-up menu displays with the following commands:

**Insert Server**

Opens the dbANYWHERE Server dialog box that lets you connect to a dbANYWHERE Server. This command is available only if a dbANYWHERE Server is selected.

**Delete Server**

Deletes the selected dbANYWHERE Server from the dbNAVIGATOR. This command is available only if a dbANYWHERE Server is selected.

**Connect**

Opens the Logon dialog box that lets you logon to a [data source](#). This command is available only if a Data Source is selected.

**Disconnect**

Disconnects the selected database.

**Refresh**

Refreshes the selected database's dbNAVIGATOR tables and columns information. This command is useful for multiple concurrent users. This command is available only if a dbANYWHERE Server or Data Source is selected.

Obsolete

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Class are added to a project a derived class.

1. Select a Base Class for the new class. The base class is the class from which the new class is derived.
2. Enter a name for the new class.

As you enter a class name, the Source File path is generated for you. The default path works best in most conditions.

End of F1 Edit file

**Component procedures and conceptual topics, including creating custom components (component reference is in other files)**

- See Also
- See Also
- See Also
- See Also
- Example
- See Also
- See Also
- See Also
- See Also
- See Also
- See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

The Java Abstract Window ToolKit (AWT) is a standard and portable GUI library that lets you create visual front ends. 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 components defined in the AWT by expanding java.awt in the Packages view of a Project window.

The AWT supplies many [classes](#) for you to develop your Java programs. It is the crucial link between your Java program and the native GUI of the operating system. The AWT performs a very high level of abstraction because there has to be a common denominator in order for your Java program to be portable across platforms. It is a Java package that can be used in any Java program by importing java.awt.\* using the **import** keyword. Visual Cafe does this automatically for you.

In the structure of the AWT, [components](#) are added to and then arranged by [layout](#) managers in [containers](#). There are a variety of [event](#) handling, menu, fonts, and graphics classes in addition to components and containers. The AWT also works well in conjunction with the networking and [threads](#) classes.

Components are what allow you to interact with your program. Components display windows and buttons, lists inside of windows, and allow you to enter text in a field. Components consist of the visible objects such as Scrollbars, Buttons, and TextAreas, just to name a few, that have been added to a Container.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe includes support for Swing components. This includes:

- support for JFC 1.0.1
- proper code generation for changes made in the Visual Cafe visual environment
- JAR tool support for pluggable look and feel
- on-the-fly viewing of different looks and feels in the Form Designer
- Interaction Wizard support for Swing events and actions
- custom Property Editors for some Swing properties
- new project templates (JFC Applet and JFC Application)
- JavaBean Wizard support for JFC
- support for borders, icons, models, and actions in the Component Library and Project window
- Swing Menu Designer
- 

Look in the JFC directory in your Visual Cafe directory for items such as samples and API documentation provided in HTML files. The Sun Microsystems Web site also contains information about Swing components.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

In the Component Library, dialogs appear in the Forms group. Some of the dialogs are [top-level components](#) and some must be contained by a top-level component.

1. Drag a Dialog component from the Component Palette or Component Library to the project.

 See Also

[Details on this step](#)

2. Connect the dialog to a trigger component by using the Interaction Wizard.

 See Also

[Details on this step](#)

## JavaBeans and Component Library

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}


 See Also

 See Also

The Component Library is a repository for storing, organizing, and displaying [components](#) and [project templates](#). When you first install Visual Cafe, all of the standard Java components plus many additional Symantec components and project templates are displayed in the Component Library.

You can customize the Component Library by adding third-party components and your own custom components and project templates. For example, if there is a certain type of [form](#) you use often, you can design it then add it to the library. The form, including the components it contains and its properties, are added. You can also select a library component and modify its properties in the Property List.


You can think of the items in the Component Library as “reusable.” Once you design them, you can use them in your projects as needed. For example, you can start a new project with a project template you designed, and you can drag components from the library into the Project window or Form Designer to use them in a project.

To add components to the library, choose File  Add Component to Library and insert a [jar](#) or [class](#) file. Or drag and drop a [jar](#) or [class](#) file from a file system window, such as the Explorer. After you add a component, you should not move its corresponding [jar](#) or [class](#) file, because Visual Cafe looks for it in that location. If you move the location of a [jar](#) or [class](#) file for a component you already added to the library, you should re-add the component.

 See Also

**Note** Some components are made of more than one [class](#) file, such as 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.

You can also 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 do not 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 project template, design your project and while the Project window is active, choose Project  Create Project Template. The project files should all be in one directory; they are copied, so you do not have to leave the files in a particular location.

At the top of the main Visual Cafe window is the [Component Palette](#), which you can also undock. 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. To customize the Component Palette, you can add components from the Component Library or Project window, delete components, and group components in different arrangements.

#### To display the Component Library

- Choose View  Component Library.

 See Also

**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

#### Tasks

[Adding components to a form](#)

[Adding components to a project](#)

[Adding a component to the Component Library](#)

[Adding a bean to the Component Library](#)

[Adding a project template to the Component Library](#)

[Adding a group to the Component Library](#)

[Viewing and changing component properties](#)

[Moving components in the Component Library](#)

[Deleting components from the Component Library](#)

[Adding components to the Component Palette](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also


A Java Archive (JAR) file is a compressed archive file that complies with the [JavaBeans](#) standard. It is the primary method for delivering [JavaBeans components](#). For example, a JAR file can contain one or more related beans, and any support files, including classes, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. You can also deploy applets and applications from a JAR file. 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 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.

 See Also

#### 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 cannot add ZIP files to the Component Library.

To create a JAR file from Visual Cafe, choose Project  See Also AutoJAR or Deploy. See [Creating a JAR file by using AutoJAR](#) and [Deploying Java programs](#) for more information. To expand a JAR, see [Expanding a JAR file](#). To view the contents of a JAR file, see [Viewing a JAR file](#).


To specify a JAR file in an HTML file, add the variable `ARCHIVE="name.jar"` to the applet tags. You can specify multiple JAR files by delimiting them with a comma (,).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Components are stored in groups, represented by a folder icon, so you can organize your components.

1. Choose Insert  Group, or right-click the Component Library and choose Insert Group.
2. Type the group name.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

To add a [component](#) to the [Component Library](#), you can insert a [class](#) or [jar](#) file, or drag a component from a file system window into the Component Library. The component must be a [JavaBeans component](#) that complies with the [JavaBeans](#) standard for it to be added.

 See Also

**IMPORTANT** 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, such as 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.

You can also 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 do not 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. You can also add a bean to the Component Library using the deployment and AutoJAR features of Visual Cafe.

 See Also

**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.

#### To insert a component

You can insert a **class** or **jar** file.

 See Also

[Details on inserting a component](#)

#### To add a component from a file system window

- Drag a **class** or **jar** file from a file system window (such as the Explorer) to the location you want it in the Component Library.

#### To add a component template from the Project window

- Drag a component from the Project window (Objects tab) to the location you want it in the Component Library.

 See Also

**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

To use a [JavaBeans component](#), 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 will appear in the Component Library.
- If an icon was specified in the BeanInfo, 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 custom 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 ... button.
- Visual Cafe derives the interactions displayed in the Interaction Wizard. It determines the interactions through Java introspection:
  - The interactions are specified in the BeanInfo class.
  - If the interactions are not specified in the BeanInfo class, Visual Cafe uses reflection to look at the public methods and derives interactions from them.

 See Also

**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.

To add a JavaBeans component to the Component Library, follow these steps:

1. Make sure the component **class** or **jar** file is in the location you want to store it.

 See Also

**IMPORTANT** For class files, this location must be in your class path. For example, \\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 File  Add Component to Library.

This menu item is available only when a project is open.

An Open dialog box displays.

3. Select the **class** or **jar** file, then click Open.

For a **class** file, an Add to Library dialog box displays.

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 the bean.

4. For a **class** file, select a group, then click OK.

The component appears in the Component Library. You should verify that it is there. (You can display the Component

Library by choosing View  Component Library.)

 See Also

**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

#### Note

- If a bean is not added to the Component Library, it most likely does not comply with the JavaBeans standard. You can use the Jar Viewer to check if the class files are present and if your JAR has a manifest.mf file. Also make sure that JAVA-BEANS=TRUE. If the bean is a JavaBean, it can be instantiated through a public null constructor.
- 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 you re-add it.
- Some components are made of more than one **class** file, such as 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.
- You can also add a bean to the Component Library using the deployment and AutoJAR features of Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 [See Also](#)

 [See Also](#)

Visual Cafe provides many tools that make creating [JavaBeans components](#) easier. For example, with a single command, you can update components used in the same project or in different projects.

1. Create a new project for developing one or more [beans](#).

Get started quickly with the JavaBean Wizard.

 [See Also](#)

[Details on using the JavaBean Wizard](#)

2. Create the bean according to the JavaBeans standard.

After using the JavaBean Wizard, you can enhance Java source code as needed.

 [See Also](#)

[Details on adding code to a Java source file](#)

For information on Visual Cafe bean support, see [Support of features in the JavaBean specification](#).

Add to your project any support files, such as property editors, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. The JavaBean Wizard adds BeanInfo for you. If you add support files to the project, they are automatically included when you create a JAR file from the project with a Visual Cafe JAR utility.


 [See Also](#)

[Details on adding files to a project](#)

3. Optionally add to the BeanInfo some information for better integrating into the Visual Cafe environment.


 [See Also](#)

[Details on this step](#)

4. To test the bean(s), you can add one or more test objects (such as an applet) to the project, then drag the bean onto an appropriate object to create an instance of the bean. To update the local project, choose Project  [Update Project Beans](#) the first time you add an instance and every time you make a change to the bean.

 [See Also](#)

[Details on this step](#)

Alternatively, to test your bean in another project, choose Project  [AutoJAR](#), which compiles and adds the bean to the Component Library. You can then drag your bean from the library into another project. Whenever you make changes to the bean, use the AutoJAR command to update the bean in the Component Library and open projects. (If the project is not open, the update occurs the next time it opens.)

 [See Also](#)


[Details on using AutoJAR](#)

5. Set deployment project options.

 [See Also](#)

[Details on this step](#)

6. Deploy the bean.

You can choose Project  [Deploy](#), which lets you create a variety of archive types, or use the AutoJAR command.

 [See Also](#)

[Details on this step](#)

7. To use the bean within the Visual Cafe environment, add it to the Component Library.

 [See Also](#)

[Details on this step](#)

 [See Also](#)

**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.

If you want to modify the bean after using it for developing other projects, Visual Cafe makes it easy to do so. If you manually replace a JAR file that was to the Component Library or use AutoJAR, all beans in the JAR that are used in open projects are automatically updated. When a project is opened that uses the bean, it is automatically updated then.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe supports the following features in the JavaBean specification:

- Invisible Beans — They are represented at design time by icons on the form.
- Internationalization — we support 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 JavaBean event binding is provided, with the exception of 6.4.1 (event methods with arbitrary argument lists) and 6.5.2 (unicast event methods).
- Properties — Full support for JavaBean 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a wizard that helps you create a basic bean. The wizard creates two files: *beanclass*BeanInfo.java and *beanclass*.java where *beanclass* is the name of your bean. New in this release is direct support of a JFC Swing bean type.


The JavaBean Wizard consists of approximately seven steps:

- Choose a bean name
- Choose a bean type
- Choose a bean weight or base class, where applicable
- Select override methods (optional)
- Add properties (optional)
- Add icons (optional)
- Review your choices

Each step contains instructions.

To learn more about JavaBeans, refer to the Sun Microsystems JavaBeans API specification. For information on Visual Cafe bean support, see [Support of features in the JavaBean specification](#).

#### To start the JavaBean Wizard and create a new project

1. Choose File  New Project.
2. Select JavaBean and click OK.

#### To add a bean to a project

- Choose Insert  JavaBean.

#### To complete the JavaBean Wizard

1. After starting the JavaBean Wizard, you may get the Introduction page. Click Next.

 See Also [Details on this step](#)

2. Now you enter a bean name and optionally a package name. Then click Next.

 See Also [Details on this step](#)

3. Select a bean type. Then click Next.

 See Also [Details on this step](#)

4. If you receive the set base class page, type a base class. Then click Next.

 See Also [Details on this step](#)

5. If you receive the bean weight page, select a bean weight. Then click Next.

 See Also [Details on this step](#)

6. Select the methods you want to override. Then click Next.

 See Also [Details on this step](#)

7. Specify the properties you want to create. Then click Next.

 See Also [Details on this step](#)

8. Specify icons. Then click Next.

 See Also [Details on this step](#)

9. Check your selections. Then click Finish.

 See Also [Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Introduction page of the JavaBean Wizard gives you an overview of what you can do in the wizard to create a JavaBeans component.

1. If you do not want to access this page again, select the Skip This Screen In the Future option.
2. Click Next to access the next page.

**Tip** If you ever want to view the Introduction page again, simply click Back from the name and package page.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



First, you need to type a bean name and optionally the package name.

1. Type a name in the Bean Name field.

The name can contain Unicode letters and digits, starting with a letter, and should not be the same as a Java keyword. There is no length restriction.

2. Optionally type a package in the Package Name field.

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 are 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 is not in uppercase letters, it is 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 are in different packages, there is no naming conflict.

2. Click Next to access the next page.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The JavaBean Wizard can create several types of beans. For more information on the classes mentioned in this page, click the API button on this help page to access the API reference.

1. Select a bean type.

The Component class is the abstract superclass of the non-menu-related AWT components. The class Component can also be extended directly to create a lightweight component. A lightweight component is a component that is not associated with a native opaque window.

A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.

A generic AWT container object is a component that can contain other AWT components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container.

Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

The class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Most Swing components are subclasses of the Swing JComponent class (which is a subclass of java.awt.Container). Some standard features of Swing components are borders; icons; tool tips; keystroke handling; auto-scrolling; double buffering; slow-motion rendering; minimum, maximum, and preferred sizes; accessibility support; and actions. Swing components are mainly lightweight components, 100% pure Java, and have a pluggable look-and-feel, which means the component appearance can change based on the computer platform or some other criteria, such as user preference, or have a platform-independent design.

2. Click Next to access the next page.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



For Singular, Composite, and Container bean types, the JavaBean Wizard lets you choose the component weight. The weight for Swing components is lightweight.

1. Select Lightweight or Heavyweight.

Many AWT components are heavyweight, meaning they own their own opaque native window. Other components are lightweight, meaning they extend directly from the Java Component or Container class or extend from another lightweight class.

When overlapping components on a form, a heavyweight component is always on top of a lightweight component, no matter how you arrange it in the Form Designer or Project window. In the Project window, the order of the components is the order overlapping components are displayed, except when lightweight and heavyweight components overlap each other. So there are two lists: the order heavyweight components overlap and the order that lightweight components overlap.

2. Click Next to access the next page. Click Finish to complete the wizard; no code is generated for information you could specify in the rest of the bean pages.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Although common cases are listed in the bean type page, the JavaBean Wizard lets you create beans based on any class.

1. Type the full qualified name of the class you want the bean to extend from. This includes the [package](#) specification.
2. Click Next to access the next page. Click Finish to complete the wizard; no code is generated for information you could specify in the rest of the bean pages.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The JavaBean Wizard lets you override methods it inherits from its superclass. When the code is created, "to do: enter code here" tags tell you where you need to enter code for each overridden method.

1. In the list, double-click a method to select it. When selected, the angle bracket (>) appears.  
Click the heading buttons to sort the list.
2. Click Next to access the next page. Click Finish to complete the wizard; no code is generated for information you could specify in the rest of the bean pages.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The JavaBean Wizard lets you add bean properties. For example, when using a bean in the Visual Cafe environment, you can modify properties in the Property List.

1. For each property you want to add, click Add property, specify the property information in the dialog box, then click OK. To edit an existing property, double-click it; editable properties do not have an (s).

Type a property name and choose a type. If you do not specify a valid type, your compile will fail.

If you select Read, a getter method (*getProperty*) for the property is generated, so users and programmers can get the value of the property. If you select Write, a setter method (*setProperty*) is generated so users and programmers can set the property value.

If you select Storage, the wizard creates a variable for use in the class.

You can select bound, constrained, or both. 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 validation on a new property value before accepting or vetoing 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.

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.

2. Click Next to access the next page. Click Finish to complete the wizard; no code is generated for information you could specify in the rest of the bean pages.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can specify up to four icons to associate with your bean. A color icon can appear in a development environment, such as the Visual Cafe Component Library and Palette. The icon can be 16 by 16 pixels or 32 by 32 pixels, can be saved in GIF or JPG format, and can be color or monochrome. The size Visual Cafe prefers is 16 by 16 pixels. After specifying an icon, the JavaBean Wizard adds information about it to the BeanInfo file and adds a getIcon method to the bean source, and the icons are copied into your project directory.



**Note** The icons also get added to your project, in the Files tab. This allows them to be added to the JAR when building your project.

1. Type the full path to the icon you want to use or click Browse to locate it from a dialog box. Click Reset to clear a field.
2. Click Next to access the next page. Click Finish to complete the wizard; no code is generated for information you could specify in the rest of the bean pages.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you have completed your selections, you can create the bean.

1. In the review page of the wizard, look over your selections. If there is something you want to change, click Back until you reach the page where you can change it. Then click Next until you reach the Review page again (or click Finish earlier).
2. Click Finish.

The wizard creates the bean.

**Note** If there were errors that prevented your bean from being created, they appear in the Messages window. Choose View  Messages to open this window. For example, you receive an error if you specified an incorrect or missing base class.

{ewc\_HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a tool you can use within its environment to automatically create [JAR files](#) from a project, which is particularly useful for packaging [beans](#). AutoJAR performs the following actions:

- saves source files
- compiles source files into class files
- creates a JAR file with a [manifest file](#)
- adds beans in the JAR to the Component Library
- updates any open projects that use the component(s)

The manifest file generated by AutoJAR automatically includes the following attributes for class files that follow this pattern:

- If a class is public, can be instantiated (it is not an abstract class or interface), and has a public constructor with no arguments, 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 implements one or more of the BeanInfo, Customizer, or PropertyEditor interfaces, Visual Cafe automatically marks the file as Design Time Only. It will have the attribute and value of Design-Time-Only:True. For example, if MybeanBeanInfo.class implements the java.beans.BeanInfo interface, the file will be marked as Design-Time-Only:True.

 See Also

**Note** If you want non-Java files to be included in your JAR, you must add them to the project. These files must be on the classpath, or Visual Cafe will not include them in your JAR and may give you an error message.

To use AutoJAR:

1. Optionally set your deployment options.

 See Also

[Details on this step](#)

2. Choose Project  AutoJAR.

The JAR file is by default placed in the output directory and has the same name as the project.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a tool you can use within its environment to quickly create [JAR](#) files. To expand JAR files, use the jar.exe utility in the java\bin subdirectory. For example, to expand the file Amazing.jar, enter the following at a DOS prompt:

**jar -xf Amazing.jar**

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a JAR Viewer tool that you can use within its environment to quickly view [JAR](#) files.

#### To look at the contents of a JAR file

- Choose Tools  JAR Viewer.

Within the JAR Viewer you can:

- Sort the JAR contents by clicking the title of the column you want to determine the sort order. 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 choosing View.
- View the manifest of multiple files by selecting the files and clicking View.
- Hide the manifest display by clicking Hide Manifest Display. Click Show Manifest Display to reveal the manifest display.
- Hide the zip/jar display by clicking Hide Zip Display. Click Show Zip Display to reveal the zip display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



JavaBeans has *introspection*, the ability to read JavaBeans classes directly with the Core 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 two Visual Cafe classes: `symantec.itools.beans.SymantecBeanDescriptor` and `symantec.itools.beans.ConnectionDescriptor`.

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 in the Visual Cafe environment. The `SymantecBeanDescriptor` class extends from `java.beans.BeanDescriptor` and is the main way to supply Visual Cafe integration information. The `SymantecBeanDescriptor` methods let you tell the Visual Cafe environment about the following:

- 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 not allow users to drop other components into this bean (if this bean derives from `java.awt.Container`)
- Visual Cafe flags, such as `INVISIBLE` for specifying [invisible components](#)
- Visual Cafe connections (used by the Interaction Wizard) that are not tied to a specific method

You can find the constants for these attributes in `com.symantec.itools.vcafe.BeanDescriptorAttributes`.

### ConnectionDescriptor

The `ConnectionDescriptor` class extends from `java.beans.FeatureDescriptor`. A `ConnectionDescriptor` encapsulates a Visual Cafe connection, which is used by the Interaction Wizard. A Visual Cafe connection defines an interaction, or connection, for a bean. It implies a relationship between objects (or between an object and itself) involving either event notification or data transmission. The Interaction Wizard allows users to graphically build these relationships between objects, and Visual Cafe is able to generate the code for the specified relationship based on the underlying connection information encapsulated in the `ConnectionDescriptor`.

The connection is made of five pieces:

- **form** — Determines whether the value of a connection is `INPUT` or `OUTPUT`. An input connection defines an interaction that sets data or initiates the execution of a method; an output connection defines an interaction that returns data.
- **type** — Sets the Java type of the input or output value of the connection.
- **expression** — Defines the code that is generated to create the connection. Properties and the following replacement variables are allowed in the code string:
  - `%name%` — name of the class/component
  - `%class%` — full class name of the class/component
  - `%arg%` — method argument used for output connection data
- **initialization** — Defines any initialization code that needs to be present prior to the code generated from the connection expression.
- **description** — Supplies an English description of the connection. This string appears in the Interaction Wizard. `%class%` is allowed.

A Visual Cafe connection looks like a method call. However, a connection can be more than that. It can be a “meta-method.” Any valid code expression can be used to generate the connection, for example, the connection “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 class). A connection does not have to be tied to a method, for example, the connection “Point the button arrow LEFT” might have the code expression `%name%.LEFT`.

The `ConnectionDescriptor` methods allow you to set the form, type, expression, initialization, and description of a particular connection. If a connection is tied to a specific method, its `ConnectionDescriptor` is associated with that method’s `MethodDescriptor`. Currently, the association uses the `MethodDescriptor`’s inherited method `setValue`, passing in a `Vector` of all `ConnectionDescriptor` objects to be associated with that method. All connections that are not tied to a specific method have their `ConnectionDescriptor` objects associated with the bean’s `SymantecBeanDescriptor` object.

### Code Samples

Following are three code samples that show three different ways of implementing the BeanInfo information specific to Visual Cafe. Here is the first sample:

```
public BeanDescriptor getBeanDescriptor() {

    SymantecBeanDescriptor bd = new SymantecBeanDescriptor(beanClass);

    bd.setCanAddChild(false);
    bd.setFolder("Additional");
    bd.setToolBar("Additional");

    bd.addConnectionDescriptor(new ConnectionDescriptor("output", "int", "",
        "%name%.BORDER_REGULAR",
        "BORDER_REGULAR"));

    bd.addConnectionDescriptor(new ConnectionDescriptor("output", "int", "",
        "%name%.BORDER_NONE",
        "BORDER_NONE"));

    return (BeanDescriptor) bd;
}
```

Here is a second style:

```
ConnectionDescriptor cd = new ConnectionDescriptor( );
    cd.setForm(ConnectionDescriptor.OUTPUT);
    cd.setType("int");
    cd.setExpr("%name%.BORDER_NONE" );
    cd.setShortDescription("BORDER_NONE");

bd.addConnectionDescriptor(cd);
```

Here is a third style:

```
ConnectionDescriptor cd = new ConnectionDescriptor(ConnectionDescriptor.OUTPUT );
    cd.setType("int");
    cd.setExpr("%name%.BORDER_NONE" );
    cd.setShortDescription("BORDER_NONE");

bd.addConnectionDescriptor(cd);
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe provides a utility for converting Visual Cafe description files so you can implement the [JavaBeans](#) standard with your custom components.

1. Run the Description File Converter.

To do so, double-click the \Bin\DescToBeanInfo.bat batch file, which is in the DescFileConverter directory of the main Visual Cafe directory.

The Description File Converter dialog box appears. You can view the Java version by choosing Help



2. Click the Location tab.
3. In the Description File Directory field, specify the full path to the directory containing 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.

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, 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 is not preserved.

7. Click the Selection tab and select the files you want to convert. Shift-click to select multiple files. Select **select all listed files** to select all files.

8. Choose File  Convert.

A dialog box appears when the conversion is complete. The files appear 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 32 by 32 **gif** file is created. For the latter, the 16 by 16 image is expanded to a 32 by 32 size.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The format of a description file (.desc) is documented in this topic.

Each description file line has the following basic syntax:

```
KEY={parameter}[, {parameter}[, {parameter} [...]]]
```

A line beginning with a semicolon (;) denotes a comment line (the remainder of the line is ignored), for example:

```
; this line is a comment
```

### Description Keys

The description file uses seven keys:

Key	Defines
BASECLASS	The base class that component “appears” to derive from.
CLASS	The class being described.
SMALLICON	The icon relating to this component.
TOOLBAR	The toolbar tab for this component.
FOLDER	The component library folder.
DEFPROPERTY	A default property for this class.
PROPERTY	A property for this class.
AWTEVENT	An AWT event for this class.
CONNECTION	An interaction for this class.
WINHELP	The Windows Help ID number.

#### Baseclass

The **BASECLASS** key defines the fully qualified class name of the base class used to define the default properties and interactions for the next class to be defined. This key is useful if you have a component that derives from `java.awt.Panel`, but you don't want to allow users to drop other components into the component.

Format:

```
BASECLASS={class name}
```

Examples:

```
BASECLASS=java.awt.Component
BASECLASS=java.awt.Panel
```

#### Class

The **CLASS** key defines the fully qualified class name of the component to be brought in. Each **CLASS** definition starts a new class/component description. Both component and abstract classes may be defined within the description file. If a description file contains multiple **CLASS** definitions, subsequent definitions describe classes and/or components that are derived from preceding classes.

Format:

```
CLASS={class name}
```

Examples:

```
CLASS=symantec.itools.awt.ImageButton
CLASS=symantec.itools.awt.spinner.Spinner
```

#### Smallicon

The **SMALLICON** key defines the "small" icon for the component Used throughout the product. The icon can either specify a DLL image containing the icon resource, or may specify an icon file that contains the icon. This key only appears once per class description, and should only be used if the class description defines a component class. The file path parameter is relative to the

BIN subdirectory of the Visual Cafe installation directory.

Formats:

```
SMALLICON={icon file path}, "0"  
SMALLICON={DLL file path}, "{resource name | '#' resource id}"
```

Examples:

```
SMALLICON=components\ImageButton.ico,"0"  
SMALLICON=components\vpojava.dll,"ICON_FOO"  
SMALLICON=components\vpojava.dll,"#10"
```

### Toolbar

The **TOOLBAR** key defines the component toolbar tab name that contains this component. This key only appears once per class description, and should only be used if the class description defines a component class.

Format:

```
TOOLBAR={toolbar tab name}
```

Examples:

```
TOOLBAR=Multimedia  
TOOLBAR=Additional
```

### Folder

The **FOLDER** key defines the component library folder name that contains this component. This key only appears once per class description, and should only be used if the class description defines a component class.

Format:

```
FOLDER={folder name}
```

Examples:

```
FOLDER=Database  
FOLDER=Multimedia
```

### Property

The **PROPERTY** key defines a property for this class. This key may appear multiple times per class description, and may define properties for both abstract and component parent classes as well as base component classes.

Format:

```
PROPERTY={type}, "{property description}", "{property method}", "{default value}"
```

where:

{type} is the property type, which currently is one of the following:

Boolean	Color
Font	Integer
String	String[]
URL	URL[]

{property description} is the description that appears in the leftmost column in the Property List.

{property method} is the internal name is the internal base name of the property. For example, if the property method is "Color", then `getColor` is considered to be the internal method to get the property and `setColor` is the method to set the property.

{default value} is the default value of the property, which may be empty ("").

Examples:

```
PROPERTY=Integer, "Initial Value", "StartValue", "0"  
PROPERTY=URL[], "Image List", "Images", ""  
PROPERTY=Boolean, "Show Border", "ShowBorder", "false"
```

### Defproperty

The `DEFPROPERTY` key defines a “default” property for this class. This key may appear only once per component description, and may define properties for both abstract and component parent classes as well as base component classes. This key is identical to the `PROPERTY` key with the difference that focus will be given to this property field in the Visual Cafe property inspector window when it first receives focus.

Format:

```
DEFPROPERTY={type},”{property description}”,”{property method}”, “{default value}”
```

where:

{type} is the property type, which currently is one of the following:

Boolean	Color
Font	Integer
String	String[]
URL	URL[]
Enum	

{property description} is the description that appears in the leftmost column of the property inspector

{property method} is the internal name is the internal base name of the property. For example, if the property method is “Color”, then `getColor` is considered to be the internal method to get the property and `setColor` is the method to set the property.

{default value} is the default value of the property, which may be empty (“”).

Examples:

```
DEFPROPERTY=Integer, “Initial Value”, “StartValue”, “0”
DEFPROPERTY=URL[], “Image List”, “Images”, “”
DEFPROPERTY=Boolean, “Show Border”, “ShowBorder”, “false”
DEFPROPERTY=Enum, “Direction”, “Direction”, “SW”, “NE=0, NW=1, SE=2, SW=3”
```

## Awtevent

The `AWTEVENT` key defines an AWT event handling property for this class. This key may appear multiple times per class description, and may define properties for both abstract and component parent classes as well as base component classes.

Format:

```
AWTEVENT={event type}
```

where:

{event type} is the AWT event type, which currently is one of the following:

```
ACTION
FOCUS
KEYBOARD
MOUSE
WINDOW
```

Here is a list of the events covered by these event types:

Event Type	Events
AWTEVENT	Java AWT event handled/processed
ACTION	ACTION_EVENT
FOCUS	GOT_FOCUS LOST_FOCUS
KEYBOARD	KEY_RELEASE KEY_ACTION KEY_ACTION_RELEASE KEY_PRESS
MOUSE	MOUSE_MOVE MOUSE_UP MOUSE_DOWN MOUSE_DRAG MOUSE_ENTER MOUSE_EXIT

WINDOW

WINDOW\_MOVED  
WINDOW\_DESTROY  
WINDOW\_ICONIFY  
WINDOW\_DEICONIFY

### Connection

The `CONNECTION` key defines an interaction, or connection, for this class. This key may appear multiple times per class description, and may define properties for both abstract and component parent classes as well as base component classes.

Format:

```
CONNECTION={form},"{type}","{init}","{expr}","{description}"
```

where:

`{form}` is either `input` or `output`. An `output` connection defines an interaction that returns data; an `input` connection defines an interaction that sets data, or initiates execution of a method that doesn't take data.

`{type}` is the Java type of the parameter or return value to the method tied to the interaction. Any Java type is valid. The most common types are `int`, `boolean`, `string`, and `void`.

`{init}` defines any initialization code that needs to be present prior to the code generated from the interaction expression (`{expr}`, see the next item). This string is usually blank.

`{expr}` defines the code generated that is used to create the connection. Properties and the following replacement variables are allowed in the code string:

<code>%name%</code>	defines the name of the class/component
<code>%class%</code>	full classname of the class/component
<code>%arg%</code>	method argument used for <code>output</code> connection data

`{description}` is an English description of the connection. This string appears in the Interaction wizard when the connection is being created. The `{expr}` replacement variables above are allowed in the `{description}` as well. `%class%` is the only replacement variable allowed in `{description}`.

Examples:

```
CONNECTION=input,"boolean","", "%name%.show(%arg%);","Show the %class% on condition"
CONNECTION=output,"boolean","", "%name%.isEnabled()","Is enabled?"
```

### Winhelp

The `WINHELP` key defines the Windows help system ID number. The number can be decimal (345607) or hex (0x54607).

Format:

```
WINHELP=helpid[, helpfile]
```

where `helpfile` is the name of the file where the help resides. If not specified, "VCAFE" is assumed.

Examples:

```
WINHELP=0x54607
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The following text is an example of the description file (.desc) for a combo box component in Visual Cafe.

```
; symantec.itools.awt.ComboBox component description

CLASS=symantec.itools.awt.ComboBox

SMALLICON=components\ComboBox.ico, "0"
ICONSUITE=Icons, "135"

FOLDER=Additional
TOOLBAR=Additional

PROPERTY=Boolean, "Editable", "Editable", "false"
PROPERTY=Boolean, "Searchable", "Searchable", "false"
PROPERTY=Boolean, "Case-sensitive", "CaseSensitive", "false"
PROPERTY=Font, "Font", "Font", ""
PROPERTY=Font, "Edit field font", "EditFieldFont", ""
PROPERTY=Font, "Drop-down list font", "DropDownFont", ""

AWTEVENT=ACTION
AWTEVENT=MOUSE

CONNECTION=input, "String", "", "%name%.addItem(%arg%);", "Add item to the list"
CONNECTION=output, "int", "", "%name%.countItems()", "Get the number of items in the list"
CONNECTION=input, "int", "", "%name%.enable(%arg%);", "Enable the item in the list"
CONNECTION=input, "int", "", "%name%.disable(%arg%);", "Disable the item in the list"
CONNECTION=input, "int", "", "%name%.delItem(%arg%);", "Delete the item from the list"
CONNECTION=input, "void", "", "%name%.delSelectedItem();", "Delete the selected item"
CONNECTION=output, "String", "", "%name%.getText()", "Get the text in the edit field"
CONNECTION=output, "String", "", "%name%.getSelectedItem()", "Get the text of the selected item"
CONNECTION=output, "int", "", "%name%.getSelectedIndex()", "Get the index of the selected item"
CONNECTION=input, "int", "", "%name%.select(%arg%);", "Select the item at the index"
CONNECTION=input, "int", "", "%name%.deselect(%arg%);", "Deselect the item at the index"
CONNECTION=input, "String", "", "%name%.select(%arg%);", "Select the item that matches the string"
CONNECTION=input, "boolean", "", "%name%.setEditable(%arg%);", "Set the editable mode"
CONNECTION=output, "boolean", "", "%name%.getEditable()", "Get the editable mode"
CONNECTION=input, "boolean", "", "%name%.setSearchable(%arg%);", "Set the searchable mode"
CONNECTION=output, "boolean", "", "%name%.getSearchable()", "Get the searchable mode"
CONNECTION=input, "boolean", "", "%name%.setCaseSensitive(%arg%);", "Set case-sensitivity"
CONNECTION=output, "boolean", "", "%name%.getCaseSensitive()", "Get case-sensitivity"
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can drag [components](#) within the Component Library to another group or location.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

As you are 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 Also

**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.

1. In the Component Library, select the component to be deleted.
2. Choose Edit  Cut, or press the DELETE key.

## Component Palette



{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



The [Component Palette](#) contains reusable [components](#). It is located at the top of the main Visual Cafe window or as a floating toolbar.

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 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.

You can drag components from the Component Palette to the Form Designer or Project window to add them. You can also click a component on the Component Palette, then drag to draw it on the Form Designer.

You can customize the Component Palette to best meet your needs:

- add components to the Component Palette
- delete components from the Component Palette
- group components in tabs
- remove a tab from the Component Palette

You can add components to the Component Palette from these areas:

- Component Library
- Project window
- Customize Palette tab in the Environment Options dialog box

You can add any component from the Component Library to the Component Palette and easily use the component in your applets and applications. If you add a component to the Component Palette from the Project window, it is added to the Component Library as well.

Also on the Component Palette are these buttons:



The Interaction Tool lets you create an interaction between two components by visually connecting them. After you complete an interaction, the Selection Tool is selected and the Interaction Tool is not selected.



The Selection Tool is enabled by default. Click this icon to return to component select mode. A component is selected only if it is completely surrounded by the selection rectangle.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The Component Palette contains a variety of [components](#) that you can add to [forms](#). It can contain visual and invisible components, program modules, and form templates.

You can control the Component Palette's position and visibility by docking, floating, resizing, and hiding.

You can customize the Palette to contain the components that you use most often.

#### To customize the Component Palette from the Component Palette tab

The Component Palette tab of the Environment Options dialog box provides an interface to help you quickly customize your Component Palette. To access it:

- Choose File  Environment Options

 See Also

Component Palette tab.

- Right-click the Component Palette, then choose Customize Palette.

 See Also

[Details on using the Component Palette tab](#)

#### To add components to the Component Palette 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 stored. 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 only available 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.

#### To add components to the Component Palette from the Project window

While working in the Project window, you can quickly add components to your custom Palette:

1. Select the component.
2. Drag-and-drop the component onto the tab of the Component Palette where you want the component to be stored.

 See Also

**Note** After you add a new component to a new tab, the group and component are added to the Component Library.

#### To create a tab on the Component Palette

- Drag a new group from the Component Library onto the Component Palette. All items in the Library group are automatically added to the Component Palette tab.

#### To move components within the same tab

- Drag-and-drop components within the same tab to reorganize their display.

#### To delete a component from the Component Palette

1. Right-click a component on the Component Palette.
2. Select Remove Component.

#### To delete a tab from the Component Palette

 See Also

**Note** Deleting a tab deletes the group and all components within the tab. However, it does not delete them from the Component Library.

1. Right-click a tab on the Component Palette.
2. Select Remove Tab.

**To float the Component Palette**

- Drag the toolbar from the top of the Visual Cafe window onto your desktop.
- Double-click somewhere in the toolbar background.

**To dock the Component Palette**

- Drag the toolbar to the top of the Visual Cafe window.
- Double-click somewhere in the toolbar background.

**To hide/show the Component Palette**

- Right-click at the top of the Visual Cafe window and select the toolbar name.
- Click the close box on the toolbar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Component Palette tab allows you to add and remove components from the Component Palette, create new tabs in the Component Palette, and reorganize components within groups.

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 current tabbed Palette toolbar.

Groups in the Palette pane represent tabs, and the components in each group display as icons on the tab.

#### To access the Component Palette tab

- Choose File Environment Options



Component Palette tab.

- Right-click the Component Palette, then choose Customize Palette.

#### To add a component or tab to the Component Palette

- Select a component or group from the Component Library pane and drag it onto a Palette Group or into the general Palette area.

Dragging a component adds it to the current Palette group. Dropping the component onto another component adds the selected component to the same group.

Or

1. Select a Group in the Palette pane.
2. Select a component in the Component Library pane.
3. Click the Add button.

The selected component is added as a member of the current group.

#### To move components within the same tab

- Drag-and-drop components within the same group to reorganize their display.
- Select a component and use the up and down arrow buttons to reorganize the display order.

#### To move components to another tab

You can move components on the Component Palette to another group:

1. Select a component and drag the component to a new location.
2. Use the up and down arrow buttons to reposition the component.

#### To delete a component or tab from the Component Palette

1. In the Palette list, select the component or group to delete.
2. Click Remove or press the DELETE key.

#### To create a tab on the Component Palette

1. Click New Group.
2. Edit the group item name.



**Note** After you add a new component to a new tab, the group and component is added to the Component Library.

#### To rename tabs

- Select the tab, then click the name again, and type a new name.

## Visual Cafe Components

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe provides an extensive collection of [components](#) to help you develop your Java programs faster. In the help, you can find step-by-step procedures for some components that show basic tasks. You can also find more detailed help in the components reference. See [Visual Cafe components and containers](#).

To get help on a component property, select the property in the Property List and press F1.

Remember that tasks common to many components are described in more general help topics.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

Most Visual Cafe components are heavyweight, meaning they own their own opaque native window. Other components are lightweight, meaning they extend directly from the Java Component or Container class or extend from another lightweight class.

When overlapping components, a heavyweight component is always on top of a lightweight component, no matter how you arrange it in the Form Designer or Project window. In the Project window, the order of the components is the order overlapping components are displayed, except when lightweight and heavyweight components overlap each other. So there are two lists: the order heavyweight components overlap and the order that lightweight components overlap.

The following Visual Cafe components are lightweight:

- ImageViewer
- InvisibleButton
- InvisibleHTMLLink
- Label3D
- RollOverButton
- SlideShow
- all shapes

Different Web browsers use a different “z-order” 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 is a good idea to “sandwich” the InvisibleHTMLLinks on top and beneath lightweight components they overlap. Use

Layout  Send to Back or Send to Front.

## Utility

Utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a subjective measure, meaning it varies from person to person and even from time to time for the same person.

Utility is often represented by a utility function, which is a mathematical expression that shows the relationship between the quantity of a good or service consumed and the utility derived from it.

Utility is a key concept in microeconomics, as it helps to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a key concept in microeconomics, and it is used to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a key concept in microeconomics, and it is used to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a key concept in microeconomics, and it is used to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a key concept in microeconomics, and it is used to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

Utility is a key concept in microeconomics, and it is used to explain consumer behavior and the allocation of resources.

Utility is also used to evaluate public policy and to make decisions about the distribution of resources.

Utility is a complex concept, and there are many different ways to measure it. However, the basic idea is that utility is a measure of the satisfaction or happiness derived from consuming a good or service.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The Timer [invisible component](#) lets you set a timer in your Java program. You can create an [interaction](#) between the Timer and other [components](#).

1. In the Component Palette, click Utility, then click Timer and drag it to the Form Designer.  
Alternatively, you can drag Timer from the Component Library.  
A new Timer appears on the [form](#) in the Form Designer.
2. Set properties in the Property List.  
You can enable and disable the Timer, cause it to cycle, and specify how long it takes before the Timer is activated.
3. Set an interaction to other components or write code to use the Timer.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The Calendar [component](#) lets you add a calendar to your [form](#). Through an [interaction](#), you can set the date the calendar displays.

1. In the Component Palette, click Utility, then click Calendar and draw it on the Form Designer.

Alternatively, you can drag Calendar from the Component Library or Palette.

A new Calendar appears in the Form Designer.

2. Set properties in the Property List.

You can set the day selection color and show or hide the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Use the TreeView [component](#) to create a hierarchical list of different text elements. The list can be expanded and contracted by users, similar to a directory structure of files. Follow these steps:

1. In the Component Palette, click Utility, then click TreeView and draw it on the Form Designer.  
Alternatively, you can drag TreeView from the Component Library or Palette.  
A new TreeView appears on the [form](#) in the Form Designer.
2. Resize the new TreeView component as needed.
3. In the Property List, click the Items property, then type your labels according to these guidelines:
  - Press CTRL+ENTER between labels, then ENTER when you are finished entering labels. (Each label should be on its own line.)
  - Add labels in order from top to bottom.
  - Create a hierarchy by adding spaces to the front of a label (spaces within a label are allowed). For example, no space is the root level, one space is the next level into the hierarchy, two spaces indicates one more level in, and so on. The label will be subordinate to the first label above it that is at a higher level in the hierarchy. For example:

```
level 1
 level 2
  level 3
level 1
 level 2
level 1
level 1
```

4. Look through the hierarchy in the Form Designer, then make adjustments to the Items property as needed.  
In the Form Designer, you can click a + to expand the hierarchy and a - to collapse it.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The ProgressBar component can display the status of a process, such as printing.

1. In the Component Palette, click Utility, then click ProgressBar and draw it on the Form Designer.

Alternatively, you can drag ProgressBar from the Component Library or Palette.

A new ProgressBar appears on the [form](#) in the Form Designer.

2. Set properties in the Property List.

You can set several properties that affect the ProgressBar appearance, including the line style, color, bar type, initial bar value, and whether a percentage is displayed.

3. Set [interactions](#) or write code to make the ProgressBar show status.

You can set interactions that display the ProgressBar status and change its appearance.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The StatusScroller [invisible component](#) lets you set a scrolling message in the status bar of a browser where your applet is displayed. The scrolling area is as wide as the text. If you want to test the component in the Applet Viewer, resize the Applet Viewer window so it is longer on the bottom.

1. In the Component Palette, click Utility, then click StatusScroller and drag it to the Form Designer.

Alternatively, you can drag StatusScroller from the Component Library.

A new StatusScroller appears on the [form](#) in the Form Designer.

2. Set properties in the Property List.

You can type the string, automatically start the StatusScroller, set the scroll direction, cause it to cycle, scroll clean (meaning the entire message scrolls before the message appears again), and set a delay.

3. Optionally set an [interaction](#) to other [components](#) or write code to use the StatusScroller.

## Multimedia

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the SlideShow [component](#) to display a series of images.

 See Also

#### Note

- For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the graphics files can continue to display. A relative URL specifies the graphics file relative to the Java program. For example, if Applet1 is in a Project directory and the graphics files are in an Images subdirectory, specify just Images/*file*.
  - SlideShow is a lightweight component. If you want to overlap components, see [Using lightweight components](#) for important considerations.
1. In the Component Palette, click Multimedia, then click SlideShow and draw it on the Form Designer.  
Alternatively, you can drag SlideShow from the Component Library or Palette.  
A new SlideShow appears on the [form](#) in the Form Designer.
  2. In the Property List, click the URL List property, click the ... button, then specify the images in the dialog box. You can optionally add a description for an image.

 See Also

**Tip** If the files are named in a sequence, you can add the first file and a dialog box asks you if you want to add the other files in the sequence.

3. Create [interactions](#) from external components, such as a Previous and a Next button, to the SlideShow to control the display of images.
4. To display a description associated with an image, create an interaction from the SlideShow to a component that will display the text you entered in the URL dialog box.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the Emblaze20 [component](#) to display an Emblaze animation.

 See Also

**Note** For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the animation files can continue to display. A relative URL specifies the animation file relative to the Java program. For example, if Applet1 is in a Project directory and the animation files are in an Animation subdirectory, specify just Animation/*file*.

1. In the Component Palette, click Multimedia, then click Emblaze20 and draw it on the Form Designer.

Alternatively, you can drag Emblaze20 from the Component Library or Palette.

A new Emblaze20 appears on the [form](#) in the Form Designer.

2. In the Property List, click the URL property, click the ... button, then specify the animation file (**blz**).
3. Create an [interaction](#) or write code to start the animation.

For example, you could create an interaction from a Start button to the Emblaze20 component. You can also pause an animation.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the `ImageViewer` [component](#) to display a graphics file, such as **jpg** or **gif** files.

 See Also

#### Note

- For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the graphics files can continue to display. A relative URL specifies the graphics file relative to the Java program. For example, if `Applet1` is in a `Project` directory and the graphics files are in an `Images` subdirectory, specify just `Images/file`.
  - `ImageViewer` is a lightweight component. If you want to overlap components, see [Using lightweight components](#) for important considerations.
1. In the Component Palette, click Multimedia, then click `ImageViewer` and draw it on the Form Designer.  
Alternatively, you can drag `ImageViewer` from the Component Library or Palette.  
A new `ImageViewer` appears on the [form](#) in the Form Designer.
  2. In the Property List, click the URL property, click the ... button, then specify the graphics file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The SoundPlayer [invisible component](#) lets you play a Sun AU format sound file.

1. In the Component Palette, click Multimedia, then click SoundPlayer and drag it to the Form Designer.  
Alternatively, you can drag SoundPlayer from the Component Library.  
A new SoundPlayer appears on the [form](#) in the Form Designer.
2. In the Property List, click the URL List property, click the ... button, then specify the sound files in the dialog box.  
You can also set the other properties in the list as needed to specify where the files are played all at once and how many times the playing repeats.
3. Optionally set an [interaction](#) to other [components](#) or write code to use the SoundPlayer.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The ScrollingText [component](#) lets you set a scrolling message within a rectangular area. You can also specify a URL to link to.

1. In the Component Palette, click Multimedia, then click ScrollingText and drag it to the Form Designer.

Alternatively, you can drag ScrollingText from the Component Library.

A new ScrollingText appears on the [form](#) in the Form Designer.

2. Set properties in the Property List.

You can type messages and URLs (press CTRL+ENTER between lines), set the scroll direction, set a delay, and the amount of pixels moved in each scroll step.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Use the Plasma [component](#) to display a rectangle containing moving colored shapes. If you do not want to view the shapes on the Form Designer, turn off the preview property.

1. In the Component Palette, click Multimedia, then click Plasma and draw it on the Form Designer.  
Alternatively, you can drag Plasma from the Component Library or Palette.  
A new Plasma appears on the [form](#) in the Form Designer.
2. Optionally set an [interaction](#) with other [components](#), such as a Start button.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Use the Firework [component](#) to display a moving fireworks graphic.

1. In the Component Palette, click Multimedia, then click Firework and draw it on the Form Designer.  
Alternatively, you can drag Firework from the Component Library or Palette.  
A new Firework appears on the [form](#) in the Form Designer.
2. Optionally set an [interaction](#) with other [components](#), such as a Start button.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Use the NervousText [component](#) to display a moving text string.

1. In the Component Palette, click Multimedia, then click NervousText and draw it on the Form Designer.  
Alternatively, you can drag NervousText from the Component Library or Palette.  
A new NervousText appears on the [form](#) in the Form Designer.
2. In the Property List, type the text string in Text field.
3. Optionally set an [interaction](#) with other [components](#), such as a Start button.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the Animator [component](#) to display a series of graphics files, such as **jpg** or **gif** files, to create an animation.

 See Also

**Note** For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the graphics files can continue to display. A relative URL specifies the graphics file relative to the Java program. For example, if Applet1 is in a Project directory and the graphics files are in an Images subdirectory, specify just *Images/file*.

1. In the Component Palette, click Multimedia, then click Animator and draw it on the Form Designer.  
Alternatively, you can drag Animator from the Component Library or Palette.  
A new Animator appears on the [form](#) in the Form Designer.
2. In the Property List, click the URL List property, click the ... button, then specify the images in the dialog box.

 See Also

**Tip** If the files are named in a sequence, you can add the first file and a dialog box asks you if you want to add the other files in the sequence.

3. Set other properties in the Property List.  
You can have the animation run a certain number of times, have it run indefinitely, and preview it in the Form Designer.
4. Optionally create [interactions](#) from external components, such as a Start and a Stop button, to the Animator to control the animation display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the MovingAnimation [component](#) to display a series of graphics files, such as **jpg** or **gif** files, to create an animation. Unlike the Animator component, the animation moves within the component boundaries by setting the shift offset property.

 See Also

**Note** For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the graphics files can continue to display. A relative URL specifies the graphics file relative to the Java program. For example, if Applet1 is in a Project directory and the graphics files are in an Images subdirectory, specify just Images/*file*.

1. In the Component Palette, click Multimedia, then click MovingAnimation and draw it on the Form Designer.  
Alternatively, you can drag MovingAnimation from the Component Library or Palette.  
A new MovingAnimation appears on the [form](#) in the Form Designer.
2. In the Property List, click the URL List property, click the ... button, then specify the images in the dialog box.

 See Also

**Tip** If the files are named in a sequence, you can add the first file and a dialog box asks you if you want to add the other files in the sequence.

3. Set other properties in the Property List.  
You can have the animation run a certain number of times, have it run indefinitely, have a certain shift offset, and preview it in the Form Designer.
4. Optionally create [interactions](#) from external components, such as a Start and a Stop button, to the MovingAnimation to control the animation display.

## Additional

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



An InvisibleHTMLLink [component](#) lets you set up a jump to a URL. Because it is invisible, InvisibleHTMLLink is ideal for creating a clickable area over an image that lets you jump to a new location in an HTML file.

1. In the Component Palette, click Additional, then click InvisibleHTMLLink and click and drag to draw a rectangle on the Form Designer.

Alternatively, you can drag InvisibleHTMLLink from the Component Library or Palette, and resize and reposition it.

A new InvisibleHTMLLink appears on the [form](#) in the Form Designer.

2. In the Property List, double-click the HTML Link URL property.
3. In the HTML Link URL dialog box, type the URL.

Anchors (#) are supported if you want to jump to a specific location within an HTML file.

#### Important

- InvisibleHTMLLink is a lightweight component. If you want to overlap components, see [Using lightweight components](#) for important considerations.
- Different browsers handle anchors (#) differently. For example, while Netscape Navigator requires one # symbol, some versions of Internet Explorer require two (##). If you are using relative URLs, the Visual Cafe component will handle both cases for you; it is sufficient to type one #. However, if you are using absolute URLs, the Visual Cafe component cannot handle the difference for you.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A MultiList [component](#) lets you create a table with rows and columns. Follow these steps:

1. In the Component Palette, click Additional, then click MultiList and draw it on the Form Designer.  
Alternatively, you can drag MultiList from the Component Library or Palette, then resize it.  
A new MultiList appears on the [form](#) in the Form Designer.
2. Resize the new MultiList component as needed.
3. In the Property List, click the Column Headings property, then type your column names. Press CTRL+ENTER between labels, then ENTER after the last label.
4. Click the List Items property, then type the list items according to these guidelines:
  - Press CTRL+ENTER between rows, then ENTER after the last row. (Each row should be on its own line.)
  - Type a semicolon (;) to indicate a new column in a row, for example, john; cheryl; tim would specify text for three columns in one row.
  - Add rows in order from top to bottom.
5. Optionally set the Heading and Cell properties to customize the look of your table.

## Panels

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

A TabPanel component can contain multiple panel components. You can access a particular panel by clicking its tab, which can appear on the top or bottom of the TabPanel. To set up a TabPanel container, follow these steps:

1. In the Component Palette, click Panels, then click TabPanel and draw it on the Form Designer.

Alternatively, you can drag TabPanel from the Component Library or Palette.

A new TabPanel appears on the [form](#) in the Form Designer.

2. Resize the new TabPanel component as needed.
3. If the background color is white, you could set the Background property of TabPanel to a color, such as lightGray, to see the tabs better.
4. Drag panel components onto the TabPanel.

A tab is automatically added for each panel, from right to left. You can change the tab order by changing the order of the panels listed in the Project window.

Be sure to drag panels onto the TabPanel, and not a panel it contains. For example, you can drop a panel next to one of the tabs.

 See Also

**Tip** To make a tab the default active tab, enter its number in the TabPanel Active Tab property. The tabs are numbered starting from 0 at the left.

5. In the Property List, choose the TabPanel component from the menu, then click the Tab Labels property.
6. Type the tab labels. Press CTRL+ENTER between labels, then ENTER when you are finished entering labels.  
The labels are added in order. The top label is the rightmost label, and the bottom label is the leftmost label.
7. If you want the tabs to appear on the bottom rather than the top, set the Tabs On Bottom property to true.
8. To add components to a panel contained by a TabPanel, click a tab in the TabPanel, then drag components onto this panel.

After clicking a tab, you can click the panel associated with that tab to better see its dimensions. All panels contained by TabPanel are the same size. You must resize the TabPanel to change the size of all the panels it contains.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A `ToolBarPanel` [container](#) can contain multiple [components](#), and is ideal for creating toolbars. Components are arranged in a `FlowLayout`. To add spaces between components on the panel, separate them with `ToolBarSpacer` components.

1. In the Component Palette, click Panels, then click `ToolBarPanel` and draw it on the Form Designer.  
Alternatively, you can drag `ToolBarPanel` from the Component Library or Palette, and resize the component as needed.  
A new `ToolBarPanel` appears on the [form](#) in the Form Designer.

2. Drag components onto the `ToolBarPanel` and arrange them in order.
3. Optionally drag `ToolBarSpacer` components onto the form to add space between the components.
4. Set other properties in the Property List, as needed.

You can change the appearance of the `ToolBarPanel`, including adding a title and changing title alignment, padding, border color, and the border style. You can also change the amount of space the `ToolBarSpacers` use.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



You can allow users to tab between fields on a [form](#) by placing the fields in a `KeyPressManagerPanel` [container](#). The tab order of the [components](#) contained by the panel is listed in the Project window. To change the tab order, change the order of components in the Project window list.

To tab out of the panel to the next traversable component in the Java program, press CTRL+TAB.

1. In the Component Palette, click Panels, then click `KeyPressManagerPanel` and draw it on the Form Designer.

Alternatively, you can drag `KeyPressManagerPanel` from the Component Library or Palette, and resize the component as needed.

A new `KeyPressManagerPanel` appears in the Form Designer.

2. Drag components onto the `KeyPressManagerPanel` and arrange them.
3. Set other properties in the Property List, as needed.

You can change the appearance of the `KeyPressManagerPanel`, including adding a title and changing title alignment, padding, border color, the border style, and the layout manager.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The `BorderPanel` [container](#) lets you organize [components](#) in a panel that has a border and can have a title.

1. In the Component Palette, click Panels, then click `BorderPanel` and draw it on the Form Designer.  
Alternatively, you can drag `BorderPanel` from the Component Library or Palette, and resize the component as needed.  
A new `BorderPanel` appears on the [form](#) in the Form Designer.

2. Drag components onto the `BorderPanel` and arrange them.

3. Set other properties in the Property List, as needed.

You can change the appearance of the `BorderPanel`, including adding a title and changing title alignment, padding, border color, border style, and the layout manager.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A ScrollingPanel [container](#) has scroll bars and can contain one [component](#) or panel, which can contain multiple components.

1. In the Component Palette, click Panels, then click ScrollingPanel and draw it on the Form Designer.  
Alternatively, you can drag ScrollingPanel from the Component Library or Palette, then resize it as needed.  
A new ScrollingPanel appears on the [form](#) in the Form Designer.
2. Design another panel, if you want multiple components in the ScrollingPanel.
3. Drag a component or the other panel onto the ScrollingPanel.  
The panel now appears within the scrolling region. You can test it in the Applet Viewer.
4. Set other properties in the Property List, as needed.  
You can change the display of the scroll bars and set a minimum size.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Use the RadioButtonGroupPanel [container](#) to contain RadioButton [components](#), which act as a group within the panel – only one radio button can be selected at a time.

1. In the Component Palette, click Panels, then click RadioButtonGroupPanel and draw it on the Form Designer.  
Alternatively, you can drag RadioButtonGroupPanel from the Component Library or Palette, then resize it as needed.  
A new RadioButtonGroupPanel appears on the [form](#) in the Form Designer.
2. Drag RadioButton components onto the panel.
3. In the Property List, set the properties of each RadioButton, including the label. Set one of the RadioButton's states to true; this is the default selection.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

Use the ImagePanel [container](#) to display a graphics file, such as a **jpg** or **gif** file. You can drop other [components](#) on top of the image. You can choose different image display options, such as tiled, centered, or scaled.

 See Also

**Note** For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and the graphics files can continue to display. A relative URL specifies the graphics file relative to the Java program. For example, if Applet1 is in a Project directory and the graphics files are in an Images subdirectory, specify just Images/*file*.

1. In the Component Palette, click Panels, then click ImagePanel and draw it on the Form Designer.  
Alternatively, you can drag ImagePanel from the Component Library or Palette, then resize it as needed.  
A new ImagePanel appears on the [form](#) in the Form Designer.
2. In the Property List, click the URL property, click the ... button, then specify the graphics file.
3. In the Property List, specify the image style.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

A `SplitterPanel` [container](#) can be divided into subpanels, which hold visual [components](#) and other panels. You must set up subpanels programmatically. See [SplitterPanel component](#) for more information. You can drop components into subpanels from the Component Library or Palette. Subpanels can be resized at run time with the mouse.

## Obsolete Custom Components Topics

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



<<Needs to be updated.>>

Creating and integrating custom components into Visual Cafe is relatively straightforward. Any Java component can be used in a Visual Cafe program by referencing the component class in the Java source code. If you want to use the custom component in Visual Cafe views, such as the Component Library, Form Designer, and Class Browser, you must integrate the component into Visual Cafe.

An integrated custom component is made up of two main pieces: the component's class or classes and the component's description file.

Visual Cafe components are basic Java classes that usually derive from `java.awt.Canvas` or `java.awt.Panel`. Component properties are exposed using the Sun Java Beans method property definitions. Panel components can contain other components (including other Panel components), whereas Canvas components cannot.

Component descriptions are files that describe the interface of components to Visual Cafe. They include definitions of properties, component interactions, event handling, and component inheritance.

If you have modified a Visual Cafe component and want to place it in the Component Library so you can reuse it, all you have to do is drag the component from the Project window, Objects view, into the Component Library.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

<<Needs to be updated.>>

If you have modified a Visual Cafe component and want to place it in the Component Library, all you have to do is drag the component from the Project window, Objects view, into the Component Library. However, if you want to add custom components that are not based on the Visual Cafe components, you need to follow these steps.

 See Also

**Note** You only need to add 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.

1. Create the component class Java file (**.class**).
2. Create a description file for the component (**.desc**).

 See Also

[Details on the description file](#)

3. Create a small icon for the component (**.ico**).


Create a 32 by 32 standard Windows icon, but only the 16 by 16 upper left region of the icon will be used and displayed by Visual Cafe.

4. Add the icon and description files to your Visual Cafe environment.

The description file and component icon must be copied to the Visual Cafe bin\component directory.

5. Make sure the class file is in the location you want to store it.

This location must be 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.

6. Display the Component Library by choosing View  Component Library.

7. Choose File  Add Component to Library.

An Open dialog box displays.

8. Select the class file, then click Open.

An Add to Library dialog box displays.

9. Select a group, then click OK.

The component is added to the Component Library. You can verify that it is there.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Custom Palette dialog box provides an interface to help you quickly customize your Palette. Access the dialog box by choosing:

- File  Environment Options



-  Component Palette tab
- Customize Palette from the Component Palette's pop-up menu

1. Select an object or group from the Component Library pane and drag it onto a Palette Group or into the general Palette area.

Dragging an object adds it to the current Palette group. Dropping the object onto another object adds the selected object as a sibling of the same group.

**Or**

1. Select a Group in the Palette pane.
2. Select an object in the Component Library pane.
3. Click the Add button.

The selected object is added as a member of the current group.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



To add objects to the Palette from the Component Library, use one of these methods:

- drag the object from the Component Library and drop it on the tab of the Palette that you want the icon stored. Visual Cafe displays a message if the addition duplicates an existing object on the Palette. Duplicates are not allowed in the same tab.
- select the objects and use the right mouse pop-up menu command Add to Palette. This command is only available if the selected objects are not currently on the active tab of Palette's tab.



**Note** Dragging and dropping a group from the Component Library creates a tab on the Palette. The tab contains all components in the group.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




While working in the Project window, you can quickly add objects to your custom Palette:

1. Select the object.
2. Drag/drop the object onto the tab of the Palette where you want the object to be stored.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Add tabs to your custom Palette to help organize components.

1. Choose File  Environment Options

 Palette tab.

2. Click New Group.
3. Edit the group item name.

A tab is also created when you drag a new group from the Component Library onto the Palette. All items in the Library group are automatically added to the Palette tab.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



As you customize your Palette, you may need to remove objects that you no longer need on a frequent basis.

**Caution** Deleting a tab deletes the group and all objects within the tab.


#### **From the Palette**

1. Right-mouse click an object on the Palette.
2. Select Remove Component.

#### **From the Environment Options dialog box**

1. Choose File  Environment Options




2.  Palette tab.
2. Select the object to delete.
3. Click Remove or press the DELETE key.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Palette components can be moved within their group or to another group.

1. Choose File  Environment Options

 See Also Component Palette tab.

2. Select a Palette object and drag the object to a new location.
3. Use the up and down arrow buttons to reposition the object.

Within the Palette toolbar, you can drag and drop components within the same tab to reorganize their display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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.

Palette customization includes:

- creating object groups
- adding objects to the Palette
- deleting objects from the Palette
- removing a tab from the Palette

You can add and remove objects on the Palette from the Environment Options dialog box Component Palette tab.

When groups are added to the Palette, they display on the Palette as tabs. Palette tabs can display horizontally only.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

Visual Cafe provides the JAR Packager, a tool you can use within its environment to quickly create [JAR files](#). While creating a JAR, Remove Files allows you to remove files that you do not want to include from the JAR.

#### **To remove files from a JAR**

1. Select the file you want to remove.

 See Also

**Note** You can select multiple files by pressing Shift while clicking the files.

2. Click Exclude Files.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

You can create [JavaBeans components](#) within the Visual Cafe environment. Visual Cafe provides tools to make your job easier.

For information on Visual Cafe bean support, see [Support of features in the JavaBean specification](#).

1. Create a new project for developing one or more [JavaBeans](#) components.

Get started quickly with the JavaBean Wizard.

 See Also

[Details on using the JavaBean Wizard](#)

2. If you are converting a custom component that has a description file, convert the description file.

 See Also

[Details on this step](#)

3. Create the [bean](#) according to the JavaBeans standard.

This includes adding to the project any support files, such as classes, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. If you add support files to the project, they are automatically included when you create a JAR file from the project with the Visual Cafe JAR utility.

4. Optionally add to the BeanInfo some information for better integrating into the Visual Cafe environment.

 See Also

[Details on this step](#)

5. To package the bean(s) in a [JAR file](#), choose Project  JAR. <<Set project options, then deploy or AutoJAR>>

 See Also

[Details on this step](#)

 See Also

**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.

6. To use and test the bean within the Visual Cafe environment, add it to the Component Library.

 See Also

[Details on this step](#)

If you use JavaBeans components you created in a new project, you can update your component project file and have the changes applied to any project that uses the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

<<Remove>> Visual Cafe enhances the development of [JavaBeans components](#) by providing a single command that immediately shows updates made to components. After modifying the form, source code, or events in your JavaBeans components, use AutoJAR to update the [JAR file](#). You will quickly see the results of those modifications in real time as any open projects using those components are automatically updated.

The following steps describe preparing a JavaBeans component for dynamic development.

1. Create a bean using the JavaBean Wizard.

 See Also

[Details on this step](#)

 See Also

**Tip** If you choose to use already created source code by adding a source file to the project, you need to include a corresponding *BeanNameBeanInfo.java* file, where *BeanName* is the name of your bean. See [Using AutoJAR with JavaBeans components](#) for a description of the file pattern. <<Now optional?>>


2. Choose Project  AutoJAR.

 See Also

[Details on this step](#)

3. Open a different project and add your new JavaBeans component(s).

The following steps describe the process of dynamic development with JavaBeans components.

1. Make any modifications to the form, source code, or events of the project containing the bean.
2. Choose Project  AutoJAR again to update the JavaBeans component(s) in the Component Library and all instances of the component in any open projects.

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.

 See Also

**Note** If you manually replace a JAR file added to the Component Library, all instances of components in the JAR that are in any open projects are automatically updated.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a JAR Packager tool you can use within its environment to quickly create [JAR files](#). The JAR Packager sets up much of the JAR for you. Initially, the display includes all of the classes in your project, any classes they depend on (depending on your JAR options), and other files you have added to the project. You can also create JAR files with the AutoJAR tool, as described in [Creating a JAR file using AutoJAR](#). AutoJAR creates the JAR faster but does not give you as much control over the JAR output.

Remember that 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.

Also, you cannot add to an existing JAR file, but you can create a new JAR file from a project.

 See Also


**Tip** If you want non-Java files to be automatically included in your JAR, you should add them to the project. Ideally, these files should be in the project directory, project output directory, or subdirectories of these directories; otherwise, Visual Cafe might not be able to determine the correct package name.

To create a JAR:

1. If needed, set up the Bundler.properties file so it contains the JAR options you want. <<now where?>>

 See Also

[Details on this step](#)

2. Make sure your project options are set to the release type you want, either Debug or Final.  
By default, when the release type is Debug, all dependent files that are in archive files (JAR/ZIP) are not included in the JAR. By default, these files are included when the release type is Final.
3. Activate the project you want to work with, then choose Project  JAR.  
Visual Cafe saves the source files and compiles the files in the project.
4. If you use Swing components in the project, you may be prompted to specify the look and feel you want to use. Select the items you want, then click OK.  
The JAR Packager appears. The panel on the left shows the files in your JAR. The panel on the right is where you specify information about the files listed on the left. You can resize the JAR tool by clicking and dragging any of the edges of the dialog box.
5. In the JAR File field, type the name and full path that you want the JAR file to have. Click the ... button to choose a path from a dialog box.
6. Add or remove files as needed.
  - To add files to the JAR, select a file and click Add File.

 See Also

[Details on adding files to a JAR](#)

- To remove files from the JAR, select one or more files and click Exclude Files.
- To add or remove files, packages, and archives that were automatically added to the JAR when you started the JAR tool, click Manage Files.

 See Also

[Details on managing files](#)

7. To specify information about a file, select a file in the left panel and specify the option in the right panel:
  - To specify a package, type it in the Package field.

Visual Cafe tries to specify the correct package for you. However, if a file is not in the project directory, project output directory, or subdirectories of these directories, Visual Cafe might not be able to determine the correct package name.

- To identify a class file as a [bean](#), select Java Bean.

If a *name.class* file has a corresponding *nameBeanInfo.java* file, Visual Cafe will automatically set that class file to be a bean. For example, if there is a *MybeanBeanInfo.java* file, *Mybean.class* will be marked as a Java Bean.

- Select Design Time Only if a class file is needed only at design time, such as a *BeanInfo* file, which is used when you add a JAR to the Component Library.

If a class file has a corresponding *nameBeanInfo.java*, *nameCustomizer.java*, or *nameEditor.java* file, Visual Cafe automatically marks the file as Design Time Only. For example, if the class file *MybeanBeanInfo.class* has a corresponding *MybeanBeanInfo.java* file, the file will be marked as Design Time Only.

- To specify dependencies within the JAR, click Add and select a file that the class depends on, such as a graphics file or another support file. (This command is the same as the depends flag in the manifest file.)



**Note** The file must be listed in the left panel before you can specify dependencies with it; clicking Add does not let you add files.

8. To create a compressed JAR file, select Compressed.



**Tip** You probably do not want to add a compressed JAR file to the Component Library because it takes more time for the development environment to access the files in the JAR. Also, some browsers may not support compressed JAR files.

9. To automatically add the beans in the JAR to the Component Library, click Add to Library.
10. Click OK to create the JAR.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

While creating a [JAR file](#) with the JAR Packager tool, you can add additional files to the JAR that may not be in your current project, such as a GIF or an HTML file, as well as additional class files.

 See Also

**Tip** If you want non-Java files to be automatically included in your JAR, you should add them to the project. Ideally, these files should be in the project directory, project output directory, or subdirectories of these directories; otherwise, Visual Cafe might not be able to determine the correct package name.

To add files to a JAR:

1. Click Add File.  
The Select File dialog box is displayed.
2. Select a file to add and click OK.
3. If needed, change the package in the Package field.
4. Repeat steps 1 and 2 to add additional files.

 See Also

**Note** The Select File dialog box does not allow selection of multiple files.



{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides the JAR Packager, a tool you can use within its environment to quickly create [JAR files](#). If you have excluded a package or archive while creating a JAR, but need a file contained in that package or archive in the JAR, Manage Files allows you to add the file back in. You can also remove all files at once from a package or archive.

To manage your files, packages, and archives:

1. Click Manage Files.  
The Manage Files dialog box is displayed.
2. Select the Files, Packages, or Archives tab to manage what items are excluded or added.
3. Add any items you have excluded back into the JAR by double-clicking it or using the Add button .
4. Remove any items you no longer want in the JAR by double-clicking it or using the Remove button .
5. Click OK to save the changes or Cancel to remove the changes.

 See Also

**Note** You can select multiple files by pressing Shift while clicking the files.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Visual Cafe provides JAR Packager and AutoJAR tools you can use within its environment to quickly create [JAR files](#). These tools set up much of the JAR for you. You can customize how you want your JAR files to be set up by configuring a Bundler.properties file located in the Visual Cafe \java\lib directory. A default file is provided.

#### **To preserve your JAR Packager settings**

If you make changes in the JAR Packager, the changes in the bottom panel are always preserved. If you also want to preserve changes you make in the upper two panels, set PersistData to true:

```
PersistData=true
```

The default is false.

#### **To change what archive files are included in JARs for Debug and Final release types**

By default, when the release type is Debug, all dependent files that are in archive files (JAR/ZIP) are not included in the JAR. By default, these files are included when the release type is Final.

To include these dependent archive files in Debug builds, set IgnoreArchivesInDebug to false:

```
IgnoreArchivesInDebug=false
```

To not include these dependent archive files in Final builds, set IgnoreArchivesInFinal to true:

```
IgnoreArchivesInRelease=true
```

#### **To create a JAR even if there are compiler warnings**

By default, a JAR tool will not create a JAR if there are compiler warnings. To create a JAR despite compiler warnings, set TestSJOutput to false:

```
TestSJOutput=false
```

#### **To include or exclude Swing classes**

By default, if you use Swing components in your project, Swing classes are included in your JAR. However, including these classes can make your JAR quite large and your users may already have the Swing classes on their computers. In this case, you could exclude Swing classes by setting Swing.include to false:

```
Swing.include=false
```

If the IgnoreArchivesInDebug (default true) or IgnoreArchivesInRelease (default false) is set to true, the Swing classes are not included even if Swing.include is set to true.

#### **To prompt for Swing look and feels**

By default, the JAR Packager prompts for Swing look and feels:

```
Swing.promptForLAFs=true
```

Before it prompts, however, the Swing classes need to be included in the JAR. That means the IgnoreArchivesInDebug or IgnoreArchivesInFinal property needs to be set to true for the corresponding release type you are using when generating the JAR, and Swing.include must be true. If either of these properties are false, the Swing classes are not included, so you are not prompted.

You can also specify the default look and feels, described next.

#### **To specify the default Swing look and feels**

With the Swing.defaultLAFs property, you can set default Swing look and feels to include when you create a JAR with the JAR or Auto JAR commands.

```
Swing.defaultLAFs=metal,motif
```

If Swing.promptForLAFs=true, these defaults appear in the dialog box when you create a JAR with the JAR Packager.

The values must be listed in the Swing.LAFs section. Here are the defaults:

```
# - These Swing look and feels are required. If the requirements change,
# - you can specify the change here.
Swing.mandatory.LAFs=plaf,basic
Swing.mandatory.LAF.plaf=swingall.jar
Swing.mandatory.LAF.plaf.base=com/sun/java/swing/plaf
Swing.mandatory.LAF.basic=swingall.jar
Swing.mandatory.LAF.basic.base=com/sun/java/swing/plaf/basic

# - These Swing look and feels are optional. You can remove the settings you
# - don't want to use or specify additional values.
Swing.LAFs=metal,motif,windows
Swing.LAF.metal=swingall.jar
Swing.LAF.metal.base=com/sun/java/swing/plaf/metal
Swing.LAF.motif=swingall.jar
Swing.LAF.motif.base=com/sun/java/swing/plaf/motif
Swing.LAF.windows=swingall.jar
Swing.LAF.windows.base=com/sun/java/swing/plaf/windows
```

#### **To specify where to look for Swing JAR files**

By default, Visual Cafe looks in the Visual Cafe Jfc directory for the Swing JAR files. You can use the Swing.home property to change this location if you have your JFC files somewhere else. For example, to set the directory to c:\jars:

```
Swing.home=c:/jars
```

#### **To display the JAR Packager version number**

By default, the JAR Packager version number appears in the JAR Packager title bar. To disable the display, set the DisplayVersion property to false:

```
DisplayVersion=false
```

#### **To gather data for Symantec Technical Support to use for troubleshooting**

During normal operation, the Debug and KeepFiles properties should be set to false. This is the default. If you need to gather troubleshooting information, you can set them to true:

```
Debug=true
KeepFiles=true
```

When Debug is true, debug logging starts. When the KeepFiles property is true, Visual Cafe keeps the intermediate files (bundler.dar and bundler.rsp) that the JAR Packager generates so Technical Support can examine them.

By default, the debug log is kept in c:\Debug.log. You can change this location and name with the DebugLog property. For example:

```
DebugLog=C:/MyDebug.log
```

#### **Sample file**

Here is a sample Bundler.properties file that shows the current default values.

```
# - Example Bundler.properties file (3.1.0).
# - All of the entries show the default values (except for Swing.home,
# - which is commented out).

# - Turn off the debug logging.
# - (True is useful for Symantec Technical Support when troubleshooting.)
Debug=false

# - Write to this debug file when Debug=true.
DebugLog=C:/Debug.log
```

```

# - Fail when a build gives warnings.
TestSJOutput=true

# - Do not keep the intermediate files that the JAR Packager generates.
# - This includes bundler.dar and bundler.rsp.
# - (True is useful for Symantec Technical Support when troubleshooting.)
KeepFiles=false

# - Do not save the included/excluded class lists and other information
# - specified in the upper two panels of the JAR Packager.
PersistData=false

# - Display the JAR Packager version number in the title bar.
DisplayVersion=true

# - For Debug builds, exclude files that come from JAR/ZIP files.
IgnoreArchivesInDebug=true

# - For Final builds, include files that come from JAR/ZIP files.
IgnoreArchivesInRelease=false

# - Include Swing classes (this effects all Swing options).
# - If an IgnoreArchivesIn property is true, then Swing
#   is excluded regardless.
Swing.include=true

# - Look here for the Swing JAR files.
# - You only need to set this if your JFC files are not
#   located in the <VisualCafeInstall>\jfc directory.
# Swing.home=c:/jars

# - When choosing Project>JAR, display a dialog box of LAFs to select from.
# - Auto JAR does not use this.
Swing.promptForLAFs=true

# - Here are the default Swing LAFs to include when you create a JAR with
# - the JAR or Auto JAR commands. If Swing.promptForLAFs=true, these defaults
# - are displayed in the dialog box. The values must be listed in the SwingLAFs
# - section below.
Swing.defaultLAFs=metal,motif

# - These Swing look and feels are required. If the requirements change,
# - you can specify the change here.
Swing.mandatory.LAFs=plaf,basic
Swing.mandatory.LAF.plaf=swingall.jar
Swing.mandatory.LAF.plaf.base=com/sun/java/swing/plaf
Swing.mandatory.LAF.basic=swingall.jar
Swing.mandatory.LAF.basic.base=com/sun/java/swing/plaf/basic

# - These Swing look and feels are optional. You can remove the settings you
# - don't want to use or specify additional values.
Swing.LAFs=metal,motif,windows
Swing.LAF.metal=swingall.jar
Swing.LAF.metal.base=com/sun/java/swing/plaf/metal

```

```
Swing.LAF.motif=swingall.jar
Swing.LAF.motif.base=com/sun/java/swing/plaf/motif
Swing.LAF.windows=swingall.jar
Swing.LAF.windows.base=com/sun/java/swing/plaf/windows
```

## Debugging


 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also


After you write and execute your program, you might encounter errors, such as compile errors (code construction, syntax errors), run-time errors that occur after you start the program (dividing by zero or writing to a file that does not exist), 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 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 that program [events](#) occur

 See Also

**Note** Your program must successfully compile into a [class](#) file before the debugger can open it.

When you run the debugger by choosing Project  Run in Debugger, Visual Cafe switches to [debug mode](#). Unless you have Visual Cafe Professional edition, while you are in debug mode, you cannot compile your source files or rebuild your project. To try out your code changes, you must first stop the debugger and recompile your program. Once the program has been recompiled, you can restart the debugger. For more information on run-time editing with the Professional edition, see [Incremental debugging features](#). With run-time editing, when you find run-time errors you can modify the source files to correct the behavior of your program.

While debugging your program, you can use the [Source window](#), [Class Browser](#), [Breakpoints window](#), [Variables window](#), [Calls window](#), [Threads window](#), and [Messages window](#) to help you isolate problems and resolve them. The Source window, Class Browser, Messages window, and Breakpoints window are available at all times; the other windows show information while you are debugging.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



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 a misbehaving program.

There are three basic techniques that can be used to step through method calls in your program code: stepping into, stepping over, and stepping out. Each technique has an associated icon in the Debug toolbar.

### Step Into

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 the 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 is not usually the best way to debug a program. For example, let's say your program calls the `println` method. We know the `println` method works, because it is 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 known working code by stepping over method calls. However, if you stepped in where you should have stepped over, you could step out.

### Step Over

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.

By using the Step Over method, you can bypass unnecessarily opening other files, and stepping into every line of code each time a method is called.

### Step Out

Step out returns to the calling method at the point after the current method was called.

### Editing Breakpoints

Watching an entire program execute from beginning to end helps you understand the program flow. However, it is more likely that there are only specific parts of the program which you want to observe. It makes sense to step through the few specific lines of code which have behavior you want to observe rather than the entire program. In these cases you can use breakpoints. A breakpoint is a flag placed in the source code that tells the debugger to pause execution of the program.

When you work with breakpoints the debugger executes 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.

To set breakpoints in your program, position your cursor on the line where you want to pause execution and click the Set Breakpoint icon in the Debug Toolbar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



The Java language uses exceptions to process errors in its programs. 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

Some types of errors force the Java run-time system to throw exceptions — from simple programming errors to invalid data. When such an error occurs within a Java [method](#), the method creates an exception [component](#) and passes it to the Java run-time system by throwing it. 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 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 the 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 as well as the Java program terminate.

### Setting exceptions in Visual Cafe

You can view and set the exceptions options by choosing Project  Options



Debug tab and choosing Exceptions from the drop-down list. To stop whenever a particular exception occurs, check the box to the left of the exception name. When this option is set, the program stops whenever the exception is encountered regardless of whether your program handles the error or not. The default behavior for exceptions is for the debugger only to stop if a particular exception is not handled in the source code. For more information on setting exception options, see [Defining how exceptions are handled by the debugger](#).


{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}


 See Also

 See Also

With Visual Cafe Professional Edition, you can use several [incremental debugging](#) features that make debugging your Java programs more efficient. While your program is executing or paused in the Visual Cafe debugger, you can edit your Java program and immediately see the effects of the code change. This is also called *run-time editing*. The code is compiled and saved automatically.

You enable run-time editing in the Debugging tab of the Environment Options dialog box. See [Setting debugging options for the environment](#) for more information.

Debug  Update Now forces an incremental update and saves all files.

Debug  Restart Method is like a “Pop” command. It takes you back to the method that called the currently active method and allows you to rerun the current method from the beginning. You should not think of it like an undo command, because it does not undo side effects of code that was run.

Another incremental debugging feature is if you change a portion of code that is active (anywhere on the call stack), the code is compiled and saved, and you get a dialog box asking 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.

 See Also

**Note** When debugging native code, you can add new methods, while with bytecode you cannot.

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



If you have the Professional edition, you can debug native Win32 applications and DLLs in Visual Cafe the same as you would bytecode programs. For DLLs, you need to specify the calling program you want to use to run and debug your DLL. See [Specifying a program for running and debugging a DLL](#) 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 debugging native code while [incremental debugging](#) is enabled, you can add new methods, while with bytecode you cannot.
- You cannot perform remote debugging.

## Running in Debugger

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

Debugging involves finding and fixing bugs in your code. After building your [project](#) you can enter debugging mode by choosing Project  Run in the Debugger.

The Debug [workspace](#) is automatically loaded. By default, the Debug workspace includes the [Calls](#), [Variables](#), and [Breakpoints](#) windows, which are opened automatically in the same size and position as they were when the project was last run. In addition, the debugger also has [Watch](#) and [Threads](#) windows.

These windows help you isolate and resolve problems. The [Source window](#) and [Class Browser](#) are also available during development and debugging.

The Visual Cafe debugger:

- provides a Windows graphical user interface
- provides a graphical representation of data structures
- supports breakpoints
- lets you drag-and-drop to execute commands

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



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 the debugger, you are executing your program as a finished program and you cannot break into debug mode.

In Visual Cafe, you can choose to run your program in one of four ways. You can run your program:

- until it terminates normally or reaches a breakpoint or an exception.
- until execution reaches the cursor location in the Source window.
- so it pauses as it is about to execute the first line.
- from its current debug point until it terminates, ignoring any breakpoints that are set.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

After creating your program, you will want to run your program until it terminates normally or encounters a breakpoint.

- Choose Project  Run in Debugger

**Note**

- Before running in the debugger, make sure your project options are set to debug. See [Specifying whether builds are debug or final](#) for more information.
- You can run your program outside of the debugger by choosing Project  Execute.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

Visual Cafe lets you run your program and pause just before the first line executes. This puts you in [debug mode](#) before anything has been done to your program's state.

- Choose Project  Step Into.


This loads your Java program into the debugger and prepares to execute the first line. At this point, you can examine the state of your program before it begins to execute.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can run your program from its current point until the end, ignoring any breakpoints that are set. This allows you to check for any suspected [exception](#) conditions.

- Choose Debug  Continue to End.

Your Java program will continue to run until it exits normally or you cause it to exit by closing it as you would outside of the debugger.

 See Also

**Note** If any kind of exception occurs, the program will break at the point of the violation, unless otherwise specified. For more information on setting exception options, see [Defining how exceptions are handled by the debugger](#). You have to restart the debug session.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can run your program until execution reaches the cursor location, after which the program pauses. This is a useful way to continue from a breakpoint to a line you are inspecting in the Source window.

- Choose Debug  Continue to Cursor.

 See Also

**Note** If the selected line does not get executed before the end of the program, your program will not pause.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



In Visual Cafe, you can pause an executing program and switch to [debug mode](#). The effect on your program is the same as if a breakpoint is hit.

- Choose Debug  Pause.



**Note** When an unhandled [exception](#) is encountered, the program pauses automatically at the line where the error occurred.


Often when you choose to pause your Java program, it is executing in a portion of code inside the Java AWT. If so, you are prompted to find that source file.

While your program is paused, you can examine its state by using the [Calls](#) and [Variables](#) windows, as well as setting breakpoints.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can resume execution when your program is paused because of a breakpoint, an exception, or because you manually paused execution. When you resume your program, execution is continued from the current location.


- Choose Debug  Continue.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Restarting a program is when you start execution over again from the beginning. This is different from continuing a program where you restart execution after a breakpoint.

- Select Debug  Restart.

When Restart is selected while a program is executing, it is the same as choosing Debug  Stop then Project

 Run in Debugger. Otherwise, it is the same as choosing Debug


 Stop then Project

 Step Into.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can completely stop a program (as opposed to pausing it temporarily). Stopping a program terminates it completely. Execution of a stopped program cannot be continued. For information on run-time editing with the Professional edition, see [Incremental debugging features](#).

- Choose Debug  Stop.

The Java program also is terminated.

## Debugger Windows

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



This window displays all Visual Cafe messages. For example, if an error is encountered during parsing, the error displays in this window. You can modify what errors display in your project options.

Anything you write to `System.out` can also appear in the Messages window when you are running the in the debugger. For example:

```
System.out.println("my message");
```

Double-click any error message to open the file in the [Source window](#) with focus on the selected error.

#### **To open the Messages window**

- Choose View  Messages.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



This window shows the variables that are active in the current context. You can modify these variables directly in this window.

For each variable, the window can display the variable name, type, and value.

#### To open the Variables window

- Choose View  Variables.

#### Tasks

[Viewing and modifying variables](#)

[Watching variables and expressions](#)

[Modifying a variable in the Variables window](#)

[Viewing the value of a variable](#)

[Viewing type information for a variable](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



This window shows all the method calls that have started, but not completed execution.

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 Calls window. The currently executing method appears at the top of the stack and older function calls 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 Calls window, you can change the context of the Variables window to display the variables for that method, and its source.

#### To open the Calls window

- Choose View  Call Stack.

#### Tasks

[Viewing Parameters for a Method on the Call Stack](#)

[Viewing Source for a Method on the Call Stack](#)

[Viewing the Call Stack for a Thread](#)

[Viewing Variables for a Method on the Call Stack](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The Breakpoints window displays a list of all breakpoints you've defined in a project. Use this window to add, remove, or modify breakpoints.

The location and condition of each breakpoint is displayed.

#### To open the Breakpoints window

- Choose View  Breakpoints.

#### Tasks

[Setting a Breakpoint](#)

[Clearing a Breakpoint](#)

[Enabling or Disabling a Breakpoint](#)

[Ignoring Breakpoints](#)

[Managing Breakpoints](#)

[Modifying a Conditional Breakpoint](#)

[Setting a Breakpoint on a Line Number](#)

[Setting a Breakpoint on a Method Name](#)

[Setting a Conditional Breakpoint](#)

[Stepping Through Code When the Program is Paused](#)

[Toggling a Breakpoint](#)

[Viewing Source for a Breakpoint](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



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. Variables in the Watch window are only updated 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 a watch expressions that rely on calls to other components.

#### To open the Watch window

- Choose View  Watch.

#### Tasks

[Deleting a variable or expression from the Watch window](#)

[Modifying a variable or expression in the Watch window](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Threads window presents at a glance all the currently existent [threads](#) that your program has created, together with their states. The Thread window provides no benefits when debugging a single-thread application. It is only when debugging a multi-threaded application that the Thread window provides indispensable services. This window lets you:

- easily switch between threads to debug
- update the Source window with a thread's current location
- update the Call window with a thread's call chain

The current threads are listed one per row. The currently selected thread is highlighted. You can change the selection by clicking a different row, or by using the 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 an application) is created. The active thread — the one from which the debugger regained control — is identified by a normal arrow in the left margin. The columns of the Thread window have the following significance:

Column	Meaning
ID	Thread id number. Thread ids are unique within a process.
Status	Status. Possible values of this field are:  Frz - the thread is "frozen" (suspended) Thw - the thread is "thawed" (resumed or not suspended)



**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

#### To open the Threads window

- Choose View Threads.

#### Tasks

[Debugging Threads](#)

[Debugging a Single Thread](#)

[Resuming Other Suspended Threads](#)

[Suspending a Thread](#)

[Suspending Other Threads](#)

[Viewing Source for a Selected Thread](#)

[Viewing the Active Variables in a Thread](#)


[Viewing the Call Stack for a Thread](#)

## Changing Source Code

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



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 directly from the Messages window.

1. Choose View  Messages to bring the Messages window to the front.
2. Double-click any error message to go to that error.

The file containing the error opens at the offending line within a Source window. Once the file opens, you can work on your source code. If you are using [incremental debugging](#), you can change your code and compile-time errors are reported in the Messages window the same as always.

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



The [Source window](#) is the primary debugging window; it is 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 provides 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.

## Breakpoints

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```

 See Also

 See Also

You can set breakpoints anywhere in your code to stop execution at a particular line and regain control after starting your program. When your program breaks at a breakpoint you can examine variable values, single step your program, or examine the state of your program in any way you want.

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

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



Visual Cafe provides many ways to manage breakpoints, allowing you to stop execution of a program at a specific location and optionally on a predetermined condition, such the value of a variable.

Breakpoints are manipulated either from the [Source window](#) or from within the [Breakpoints window](#), which shows all breakpoints currently defined in your program. You can view or modify the different parameters of any breakpoint and enable or disable the currently defined breakpoints. Use the Breakpoints window to examine the state of breakpoints in your program.


Breakpoints are saved with the source and are made visible each time you open the [project](#). The state of each breakpoint — enabled or disabled — is also saved with the project.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You set a breakpoint directly in a line of source code in the [Source window](#).

1. Click the line where your breakpoint is to be set.
2. Choose Source  Set Breakpoint, or right-click and choose Set Breakpoint.

The breakpoint is set, indicated by a diamond symbol in the left margin next to the line of code.


 See Also

**Tip** You can also set and remove breakpoints on the current line by pressing the F9 function key.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```




You can have your program break every time a specific line within your current source file is about to be executed.

1. Choose Source  Set Conditional Breakpoint.
  2. Select Line number.
  3. Enter a line number in the text box.
  4. Click OK.
- The breakpoint is added to the list in the [Breakpoints window](#).

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can have your program break every time a specific method is executed by setting a breakpoint at a method name.


1. Choose Source  Set Conditional Breakpoint.
  2. Select Method name.
  3. Enter a method name in the text box.
  4. Click OK.
- The breakpoint is added to the list in the [Breakpoints window](#).

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

A conditional breakpoint pauses the execution of your program when a specified condition is met. In Visual Cafe, when you set a conditional breakpoint, it is added to the breakpoint list in the Breakpoints window along with the condition you specify.

1. In the Source window, click the line where the breakpoint should occur.
2. Choose Source  Set Conditional Breakpoint.
3. Select If expression is true.
4. Type your breakpoint condition into the text box.

This condition must be an expression that evaluates to true when you want a breakpoint to trigger. When the expression is true, the breakpoint occurs.

 See Also

**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](#).

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```


 See Also

 See Also



Clearing a breakpoint deletes it from the set of breakpoints you have defined. This is distinct from disabling a breakpoint, which retains it in your breakpoint list in a temporarily disabled state.

You clear a breakpoint either at a source line (in the [Source window](#) or from the [Class Browser](#), or from the list maintained in the [Breakpoints window](#).

#### To clear a breakpoint at a line of source code

1. Select the breakpoint by clicking in the line where your breakpoint is set (the breakpoint is denoted with a diamond symbol in the left margin).
2. Choose Source  Clear Breakpoint, or right-click and choose Clear Breakpoint.

#### To clear a breakpoint from the Breakpoints window

- Select one or more breakpoints in the list and press DELETE.
- Select one or more breakpoints, then choose Breakpoints  Clear, or right-click and choose Clear.
- To clear all breakpoints, then choose Breakpoints  Clear All, 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 item is removed from the Breakpoints window.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The checkbox preceding each breakpoint in the [Breakpoints window](#) indicates whether or not that breakpoint is enabled. When checked, the breakpoint is enabled. In the [Source window](#), disabled breakpoints appear as empty red diamonds.

You can enable or disable a breakpoint by clicking the breakpoint's checkbox.

**To enable a breakpoint**

- Find the breakpoint in the list and click the empty checkbox corresponding to that breakpoint. When the box is checked, the breakpoint is enabled.

**To disable a breakpoint**

- Find the breakpoint in the 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 Breakpoints window.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

Toggling a breakpoint is a quick way to reverse the state of a breakpoint on any line of source code in the Source window.

1. Click the line where you want to toggle the breakpoint.
2. Click  (Toggle Breakpoint) in the Debug toolbar.

If the breakpoint is set, a diamond symbol appears in the left margin next to the line of code; if it is reset, the diamond symbol is cleared.


 See Also

**Tip** You can also use the F9 function key to toggle breakpoints.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You modify a conditional breakpoint by changing the conditions of the breakpoint in the [Breakpoints window](#).

1. Choose View  Breakpoints to display the Breakpoints window.
2. Click 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

There is an easy way in Visual Cafe to ignore all breakpoints you have set without changing the state of any of the breakpoints in the Breakpoints window. You can do this two ways.

#### **To run to program end**

Ignores breakpoints and runs the program to termination.

- Choose Debug  Continue to End.

#### **To run to the cursor location**



Ignores breakpoints until execution hits the current line, where execution pauses.

- Choose Debug  Continue to Cursor.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In Visual Cafe, you can select any of the breakpoints listed in the [Breakpoints window](#) and view the source code associated with that breakpoint.

1. Choose View  Breakpoints to display the Breakpoints window.
2. Click the breakpoint for which you want to see the source code.
3. Choose the Breakpoints  Go to Source, or right-click and choose Go to Source.  
The focus in the Source window will be set to the specified breakpoint.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



After your program has hit a breakpoint, you can step through your lines of code one at a time using three techniques.

### Step In

Steps in to the next line of source code. 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 In steps inside the method. If the method is in another source file, that file is opened to the next line of source code.

### Step Over

If the current statement is a method call, the program executes to the next statement following the call. Step Over executes the program to the next statement (unless a breakpoint or an exception is encountered before execution reaches that point).

### Step Out

This command executes to the current method's return address (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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also


 See Also

After your code has hit a breakpoint, you can choose to single step by stepping into any method call contained in the next line to be executed. This causes the single step to land at the first line of the called method.

- Choose Debug  Step Into.

 See Also

**Note** If the next line to be executed does not contain a method call, Step Into performs a single step of the line.

To start debugging and pause at the first line, choose Project  Step Into. When used on an applet, this command takes you to the applet constructor.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can step 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.

- Choose Debug  Step Over.

 See Also

**Note** If the next line to be executed does not contain a method call, Step Over performs a single step of the line.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



If you hit a breakpoint in a method and want to execute the rest of that method returning to the caller, you can choose to step out of the currently executing method.

- Choose Debug  Step Out.

## Debugging Variables, Expressions, Methods

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



In Visual Cafe, you can examine any of the variables in your program while in [debug mode](#).

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.

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 the [Variables window](#), the [Watch window](#), or the Evaluate Expressions dialog box.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can view a variable by selecting its name from the [Variables window](#).

1. Choose View  Variables.

The Variables window opens, displaying all the variables in the current context of the program.

2. To expand an object to see its contents, click the plus sign to the left of the object you want to expand.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can view a variable's type from the [Variables window](#).

1. Choose View  Variables.

The Variables window opens, displaying all the variables in the current context of the program.

2. Click the variable whose type you want to view.

The variable's type is shown in the Type column.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



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 only available 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 a watch expressions that rely on calls to other components.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can watch a variable or expression while debugging by adding the variable name to the [Watch window](#).


There are two ways to do this: directly from the Watch window, or from the [Variables window](#).

**Caution** Since the program is paused, do not enter a watch expressions that rely on calls to other components.

#### **To add it by dragging and dropping**

- Drag an item from the Variables window to the Watch window.

#### **To add it by typing the name**

1. Choose View  Watch to display the Watch window.
2. Type a variable name or an expression directly into the Watch field of the Watch window.

#### **To add it from a dialog box**

1. Right-click the variable you want to watch.
2. Choose Evaluate Expression from the pop-up menu.
3. In the dialog box, click Add Watch.

The Watch window evaluates and immediately displays the value of the variable or expression.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can add a variable directly in the [Watch window](#). The Watch window can also be used for modifying and delete variables in place while your program is in [debug mode](#).

1. Choose View  Watch.

All the variables and expressions you have chosen to watch are displayed in the Watch window.

2. Start typing the name of the variable in the window.

As you begin to type, the window changes to edit mode for the column of the selected row.

3. Press ENTER to save or ESCAPE to leave the item unchanged.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can change the variable name or expression you elected to watch at run time directly from the [Watch window](#). The Watch window can also be used for modifying variables in place while your program is in [debug mode](#).

#### To locate the variable you want to modify

1. Choose View Watch.

All the variables and expressions you have chosen to watch are displayed in the Watch window.

2. Click the variable or expression you want to change.

#### To change the value of a variable

1. Click in the Value field.
2. Type the new value in the Value box.

#### To change the variable or expression to watch

1. Click in the Watch field.
2. Edit the variable or expression.
3. 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.

**Caution** Since the program is paused, do not enter a watch expressions that rely on calls to other components.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you have finished watching a run-time variable or expression, you can delete it directly from the Watch window.

1. Choose View  Watch.

All the variables and expressions you have chosen to watch are displayed in the Watch window.

2. Click the variable or expression 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can modify a variable by selecting its name from the [Variables window](#) and typing a value into the value column.

1. Choose View  Variables.

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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

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 open the [Calls window](#), choose View  Call Stack.

#### To view parameter types

- Choose Calls  View Parameter Types.

This toggles the display of parameter types in the Procedure column of the Calls window.

#### To view parameter values



- Choose Calls  View Parameter Values.

This toggles the display of parameter values in the Procedure column of the Calls window.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```





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.

1. Choose View  Call Stack to display the Calls window.
  2. Click the Method whose variables you want to view.
  3. Choose Calls  Go to Variables.
- This opens the [Variables window](#) that lists the variables in the selected call.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Visual Cafe allows you to choose any method entered on the call stack and view the source code for that method.

1. Choose View  Call Stack to display the Calls window.
2. Click the Method whose code you want to view.
3. Choose Calls  Go to Source.  
This opens the [Source window](#) for the selected call.

## Threads

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

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.

However, 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 are 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.

While debugging, the Visual Cafe Threads window lists the active threads — one per row. The 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 an application) is created. The active thread — the one that was active when you entered break mode — is identified by an arrow in the left margin.

When [debugging](#) a multithreaded program, you can select a single thread using the Threads window. The Threads window shows all the extant threads your program has created, and the state of each thread.

You can use the Threads window to:

- Switch easily between threads to debug.
- View the source for a selected thread.
- Update the Calls window with a thread's call chain.
- Update the Variables window to show the variables within the current tread.

To open the Threads window, choose View  Threads.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

When debugging a multithreaded program in Visual Cafe, you can work with a single thread in exclusion of others. All the extant threads your program has created display in the Thread window, along with the state of each thread.

You can switch between threads to debug or view the source for a selected thread from the [Threads window](#). You can also update the [Calls window](#) with a single thread's call chain and update the [Variables window](#) to show only the variables within the current thread.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.



{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



If you want to focus your debugging efforts on a specific thread, you can set the focus to that thread and work on it alone.



**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread you want to work on.
3. Choose Threads  Set Focus.  
This sets the focus of the debugger to the selected thread.
4. Continue debugging that thread in the [Calls window](#), [Variables window](#), or [Source window](#).

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



 See Also

 See Also

If you are working on a multithreaded program, you can suspend any specific thread if you suspect it of causing unwanted side effects while your program is running or while you are debugging.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread you want to suspend.
3. Choose the Thread  Suspend.  
The selected thread is suspended.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



 See Also

 See Also

If you have 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.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread you want to resume.
3. Choose Threads  Resume.  
The selected thread is resumed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

If you want to focus your debugging attention on a single thread, Visual Cafe allows you to suspend other threads to narrow down the behavior of your program.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Thread window.
2. Click the thread you want to work on.

 See Also

**Warning** You are choosing all threads other than this one to be suspended.

3. Choose Threads  Suspend Others.

All other threads in the debugger except the one that is selected are suspended.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

If you have chosen to suspend any threads in a multithreaded program, you can resume all other threads at any time.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread you do not want to resume.

 See Also

**Warning** You are allowing threads other than this one to resume.

3. Choose Thread  Resume Others.



All other threads in the debugger except the one that is selected are resumed.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

If you want to look at the source code for a specific thread, you can do so from the Threads window.

1. Choose View  Threads to display the Threads window.
2. Click the thread you want to focus on.
3. Choose Threads  Set Focus.  
Set Focus updates the [Source window](#).
4. Look in the Source window.

The thread's source code is visible in the Source window.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




 See Also

 See Also

You can view the values of variables for any thread currently executing.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread you want to focus on.
3. Choose Threads  Set Focus.  
Set Focus updates the [Variables window](#).
4. Choose View  Variables.  
The Variables window opens displaying the chosen thread's variables.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




 See Also

 See Also

If you're working on a specific thread and want to see the call stack for that thread alone, Visual Cafe allows you to do so.

 See Also

**Note** The Threads window does not show any user-created threads until a breakpoint is hit.

1. Choose View  Threads to display the Threads window.
2. Click the thread whose call stack you want to view.
3. Choose Threads  Set Focus.  
Set Focus updates the [Calls window](#).
4. Choose View  Call Stack.  
The Calls window opens on the thread you selected.
5. Click the Method whose code you want to view.

## Remote Debugging

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

With Visual Cafe Professional Edition, you can debug an applet or application remotely on another computer. To do this, the class files and HTML files need to be located on the remote computer, while the project and the source files are located on the local computer. Both the local and the remote computer need to have TCP/IP installed and working properly.

 See Also

**IMPORTANT** The class files you are debugging reside on the remote computer and not on the computer where you are running the debugger. So, to have a good debug session, you need to be sure that the remote computer has the debug build of the classes you need, and so on. Also, the remote virtual machine needs to have the classpath set in the environment before running debugvm.exe; it should be set the same as when you run class files using a Java virtual machine. Make sure the classpath is set correctly in autoexec.bat for Windows 95 or in the Control Panel for Windows NT; debugvm.exe can also read classpath information from the sc.ini file in your Bin directory.

You can debug to a computer that is not running Windows 95/98/NT. In addition, you can use the remote debugging feature to debug to a different virtual machine or JDK than the Visual Cafe environment uses. For more information, look at the information here, and see [Tips for debugging different virtual machines and JDKs](#) and [Debugging with JDK 1.2](#).

You cannot debug native applications or DLLs remotely.

#### To start remote debugging

1. If you are debugging to a remote Windows 95/98/NT computer, install the same version of Visual Cafe on the remote computer as you have on the local computer.

 See Also

**Note** If you do not install Visual Cafe and use the Sun JDK instead (for example), it does not have Symantec classes that make remote debugging work more reliably. As a result, you can experience errors and abnormal terminations. This applies if you are debugging to a computer other than Windows 95/98/NT, for example.

2. Execute the VM you want to connect to.
  - If you installed Visual Cafe on the remote computer, open a console session on the remote computer by moving to the directory containing the class and HTML files of the program you intend to debug. Then execute debugvm, which is in the Visual Cafe directory. For example, after opening an MS-DOS window and moving to the directory where your project files reside, you could type the following if debugvm is on the path:

#### debugvm

Debugvm displays some version information, and then a password.

- If you want to debug to a computer other than Windows 95/98/NT, to get a password to use for remote debugging, you can enter a command like this (EmptyApp is included in the JDK):

**java\_g -debug sun.tools.debug.EmptyApp**

- If you want to debug in a VM other than the VM the Visual Cafe environment uses, open a console session on the local or remote computer by moving to the directory containing the class and HTML files of the program you intend to debug. Then type a command like the following (see the Sun JDK documentation for more information):

**java\_g -debug sun.tools.debug.EmptyApp**

**java -debug sun.tools.debug.EmptyApp**

3. On the local computer, open the project corresponding to the class files on the computer where you executed the VM.
4. If you are remotely debugging an applet, from the Project tab, specify an HTML file including the full path (as the remote computer sees it). Then click OK.

For example, if the HTML file is in d:\Files on the remote computer, the path you would specify is d:\Files.

5. Choose Project  Debug in Waiting VM.

6. In the Remote Debug dialog box, type information in the fields:


Field	Description
Host	The host name or IP address of the remote computer. Type <b>127.0.0.1</b> or <b>localhost</b> if the process is on the local computer.

Password                      The remote computer's password, which is returned by debugvm.exe or java.exe run on the remote computer.

7. Click Go To Breakpoint or Step Into.

The Go To Breakpoint command acts the same as the Run in Debugger command.

#### To end remote debugging

- Choose Debug  Stop.
- Run the Java program to completion (cause it to exit on its own).

You might have to select the console window and press CONTROL+C.

To debug again, you need to follow the steps in the previous procedure, including obtaining a new password.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Here are some tips if you want to use Visual Cafe to debug Java programs using JDK 1.2. For general information on debugging in a waiting VM, see [Debugging remotely](#).

1. Set up a pluggable VM for JDK1.2 (RC1). Be sure to put *JDK-1.2-Install\jre\lib\rt.jar* on the front of your classpath in the Virtual Machines tab of your Environment Options.

To execute and debug applets, you need to set permission in the security file *JDK-1.2-Install\jre\lib\security\java.policy*. Inside the grant{} block, add these two lines:

```
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
```

2. Unzip the classes in *VisualCafe\java\lib\Wrappers\generic-jdk-1.2.zip*. Insert them into the JAR file *JDK-1.2-Install\lib\tools.jar*.
3. In a DOS session, set up the environment variables to refer to the necessary classes and executables.



For example, you might use commands like the following so you can run java.exe from the directory where your main classes are:

```
C:\>set CLASSPATH=.;<vcafeinstall>\bin\components\symbeans.jar;mybeans.jar;%CLASSPATH%
C:\>set PATH=<jdk1.2install>\bin;%PATH%
```

4. Navigate to the directory containing your classes and enter the java.exe execution command line. For example:

```
C:\MyProject>java -Xdebug -Djava.compiler=NONE -Xbootclasspath/a:<jdk1.2install>\lib\tools.jar
symantec.tools.debug.EmptyApp
```

The VM password appears, and the VM waits.

5. In Visual Cafe on the local computer, activate the project you want to debug. Then choose Project  Debug in Waiting VM.
6. In the Remote Debug dialog box, type **127.0.0.1** or **localhost** in the Host field and the agent password (returned in step 4) in the Password field.
7. Click Goto Breakpoint or Step Into to begin debugging.
8. To stop debugging, choose Debug  Stop, or run the program to completion.
9. To start debugging again, start from step 3 or 4.

When you are finished debugging, you might have to activate the DOS window and press CONTROL+C.

**Obsolete**

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



You can debug an applet or application remotely on another computer. To do this, Visual Cafe and the class files and HTML files need to be located on the remote computer.

Both the local and the remote computer need to have TCP/IP installed and working properly.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

On the remote computer, select the debugvm console window and press CONTROL+C. This terminates the debug session currently in process.

To continue, you need to re-execute debugvm to get a new agent password.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

## Common errors

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 multiple times. The result is multiple compiler errors from a single typing error.

Sometimes the compiler error message will indicate exactly what caused the problem. You may have to inspect the troublesome line of code character-by-character to locate the source of the problem.

## Compiler errors

A compiler error occurs when the compiler does not understand a portion of your source code. Even in the simplest program, it is 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. Fixing the logic of your program to get it to run the way you want is called debugging your program.

The Messages window displays all of the compiler errors encountered each time you build or compile a program.

## 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. Double-click the error message to jump to the line containing the error. The editor highlights the error. The integration between the Messages window and the Source window saves you from searching through every line of your code to find an error.

## 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 is 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 sensitive name as the class you are creating.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The debugging views consists of these windows:

### Variables

The Variables window allows you to 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. To watch all the variables accessible to a method, simply drag the method from the [Calls window](#) into the [Variables window](#).

You can add a variable name or an expression to the Variables window by clicking in a cell and typing the name. Variables can be modified by double-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 debug session.

### Calls

The Calls window lists the method calls a program has made since it began running. This list is known as the [call stack](#). The [methods](#) in the call stack are displayed in reverse order from the last (most recent) call on top to the first (initial) call on the bottom. Each entry lists the name of the method followed by the name of the class which contains the method.

### Threads

The Thread window displays the [threads](#) that your program has created. A thread is a process that is controlled by your program. It is possible to create programs that have more than one process (thread), running at the same time.



**Note** If you are using multiple threads in your program, each thread has its own call chain. Drag a thread from the thread window and drop it on the call window to see the call chain for individual threads.

### Breakpoints

The Breakpoints window is used to display a list of breakpoints currently set in the source code. Breakpoints are flags you can insert into the code at specific points that cause the program execution to pause. When the debugger goes through the instructions, it stops whenever it encounters a breakpoint. You can check on the value of the data and other program conditions.


### Source window

The Source window is the primary debugging view: it is where you add, edit, and view code. When debugging an application, Visual Cafe automatically opens the Source window containing the current line of code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

When running your program, you can choose to have all exceptions break into the debugger, or to have only unhandled exceptions break into the debugger.

#### **To prepare for handling exceptions**

1. Choose Project  Options.
2. In the dialog box that appears, click the Debugger tab.
3. Choose how you want to handle exceptions, and add exceptions to the list.

#### **To have all exceptions break into the debugger**

1. Select the exception in the list.
2. Click in the Action column.
3. Click the down-arrow button and choose Stop Always.

#### **To have only unhandled exceptions break into the debugger**


- Select the exception in the list.

You check a checkbox next to an item that you want *always* to pause the debugger. The default behavior is for unchecked items to pause the debugger if they are not handled in source code.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can have your program break whenever a variable reaches a specific value or an expression of your choice becomes true.

A breakpoint symbol in the left margin of the Source window indicates a breakpoint on the adjacent line.

1. Choose Source  Set Conditional Breakpoint.
2. Select If expression is true.
3. Type a Boolean expression into the text box.

This condition 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 true. 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.

4. Click OK.

The breakpoint is added to the list in the [Breakpoints window](#).

# Glossary

This is the glossary for Visual Cafe, Visual Cafe Database Development Edition, and dbANYWHERE:

- Definitions
- URLs (not pop-ups because they're too large)
- dbawdsn.ini fields (for dbANYWHERE)
- Fields in Data Source configuration files (for dbANYWHERE)

## How it was created

1. Started with defin.doc (the glossary for Visual Cafe).
2. Put entries in alphabetical order.
3. Added definitions from the Pro glossary (progloss.doc) and the dbaw glossary (db\_pop.doc).
4. Verified that all definitions in the Visual Cafe User's Guide glossary (defin.doc) are already in here.
5. Created and attached glossary.dot for the styles (glossary.dot is a revised version of robotf.dot).
6. Combined duplicate entries.
7. Modified descriptions (so that they are clearer).
8. Added descriptions: JDK, Java API, Hierarchy Editor, dbNAVIGATOR, Project window, Form Designer, server, foreign key, optimistic concurrency, primary key, stored procedure.
9. Create the main glossary topic which can be added to the .cnt files. For the bitmaps, I used the samples provided on the CD that came with *Developing Online Help* by Boggan, Farkas, and Welinske. You may want to use different bitmaps because I don't know if it's legal to use these and because someone can probably create more attractive ones.

## Ramifications

1. Files for projects: Each help project needs to:
  - Delete the current glossary/definitions file (defin.doc, progloss.doc, or db\_pop.doc).
  - Add this file (gloss.doc).
  - Make whatever modifications are necessary to access the definitions in this file.
2. Duplicate entries:
  - debug mode: Kept the Debug\_mode\_glossary topic ID. Deleted the Break\_mode\_glossary topic ID.
  - Java Virtual Machine: Kept the Java\_Virtual\_Machine\_glossary topic ID. Deleted the Java\_VM\_Java\_Virtual\_Machine topic ID.
  - database view, global view, and two views: Deleted database view and global view (this information was in the other view definitions). Combined the two view definitions. Kept the View\_glossary topic ID. Deleted the global\_view\_glossary and view topic IDs (there were two of these).
  - URLs: Deleted the URL entry because it has been superseded by the dbANYWHERE Server machine URL entry. Deleted the URL topic ID.
  - class method, instance method, method: Deleted the class method and instance method entries. Kept the Method\_glossary topic ID. Deleted the Class\_method\_glossary and Instance\_method\_glossary topic IDs.
  - component and visual object: Deleted the visual object entry and its Visual\_Object\_glossary topic ID. Kept the Component\_glossary topic ID.
3. Changed entry names:
  - Changed class variable and instance variable to variable. Kept the Class\_variable\_glossary topic ID. Deleted the instance\_variable topic ID.
  - Changed class member to member. Kept the same topic ID.
  - Changed invisible component and non-visible object to hidden component. Kept the Invisible\_object\_glossary topic ID. Deleted the Non\_visual\_object\_glossary topic ID.
4. Deleted entries:

- control (topic ID = Control\_glossary).
5. Since the main glossary topic uses mid-topic jumps, you must redefine the visual window so that it does not auto-size the height.

**To do**

1. action: Make sure all the valid actions are listed.
2. Data Source: what spreadsheets are supported? should this information go in the glossary definition or in the "Installing dbANYWHERE Server" manual?

Main glossary topic

a  
b  
c  
d  
e  
f  
g  
h  
i  
i  
k  
l  
m

n  
o  
p  
q  
r  
s  
t  
u  
v  
w  
x  
y  
z

## A

[abstract class](#)

[access type](#)

[action](#)

[active window](#)

[adapter](#)

[anchor component](#)

[applet](#)

[applet tag](#)

[application](#)

## B

[bean](#)

[boolean expression](#)

[Breakpoints window](#)

[bytecode](#)

## **C**

[call stack](#)

[Callswindow](#)

[class](#)

[Class Browser](#)

[client machine](#)

[client software](#)

[component](#)

[Component Library](#)

[Component Palette](#)

[conditional breakpoint](#)

[connection statistics](#)

[ConnectionInfo](#)

[ConnectionManager](#)

[container](#)

[context menu](#)

[custom code](#)

## **D**

[data binding](#)

[datasource](#)

[databound](#)

[databound component](#)

[dbANYWHERE](#)

[dbANYWHERE API](#)

[dbAW subprotocol](#)

[dbAWARE component](#)

[dbNAVIGATOR](#)

[debug mode](#)

[Debug toolbar](#)

[Debugging window](#)

[declaration](#)

[definition](#)

[design time](#)

[development machine](#)

[dialog box](#)

[DML](#)

## **E**

[event](#)

[event adapter](#)

[event binding](#)

[event handler](#)

[event listener](#)

[exception](#)

## **F**

[foreign key](#)

[form](#)

[Form Designer](#)

[frame](#)

## **G**

[garbage collection](#)

## **H**

[Hierarchy Editor](#)

## **I**

[identifier](#)

[incremental debugging](#)

[inheritance](#)

[inner class](#)

[instance](#)

[interaction](#)

[interface](#)

[introspection](#)

[invisible component](#)

## **J**

[Java API](#)

[Java file](#)

[Java Archive \(JAR\) file](#)

[Java Virtual Machine](#)

[JavaBeans](#)

[JavaBeans component](#)

[Javadoc](#)

[JDBC API](#)

[JDBC-ODBC bridge](#)

[JdbcConnection](#)

[JDK: Java Developer's Kit](#)

[join](#)

## **K**

(none)

## **L**

[layout](#)

[list binding](#)

[listener](#)

## **M**

[macro](#)

[manifest file](#)

[master/detail relationship](#)

[mediator](#)

[member](#)

[method](#)

[Messages window](#)

[metadata](#)

[mnemonic](#)

[modal](#)

[modeless](#)

## **N**

[nested class](#)

## **O**

[ODBC](#)

[ODBC subprotocol](#)

[object](#)

[Objects view](#)

[optimistic concurrency](#)

[overloading](#)

[overriding](#)

## **P**

[package](#)

[Packages view](#)

[panel](#)

[ping: Packet INternet Groper](#)

[primary key](#)

[project](#)

[Project template](#)

[Project window](#)

[projection binding](#)

[property](#)

[Property List](#)

## **Q**

[QueryNavigator](#)

## **R**

[RecordDefinition](#)

[refresh](#)

[result set](#)

[regular expression syntax](#)

[RelationViewPlus](#)

[run mode](#)

[run time](#)

## **S**

[scope](#)

[Session](#)

[server](#)

[servlet](#)

[Source pane](#)

[Source window](#)

[SQL](#)

[SQL statement](#)

[stored procedure](#)

## **T**

[thread](#)

[Threads window](#)

[top-level component](#)

[transaction isolation levels](#)

## **U**

(none)

## **V**

[ValueTip](#)

[variable](#)

[Variables window](#)

[view](#)

[Visual Cafe window](#)

## **W**

[Watch window](#)

[window](#)

[workspace](#)

## **X**

(none)

## **Y**

(none)

## **Z**

(none)

## Definitions

A

**abstract class**

A class that contains an abstract method and is incompletely defined; that is, at least one method is not complete. You cannot instantiate an abstract class.

## **access type**

Scope 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' 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.

## **action**

1. A commonly used procedure and its bound properties, including any images, bundled into a single class. An example of an action is the Cut menu item and toolbar item in the Edit menu.

2. Action performed on a client-side result set:

- Delete record: marks the current record for deletion.
- First record: displays the first record.
- New record: inserts a new record in the client record set.
- Next record: displays the contents of the next database record
- Previous record: displays the contents of the previous database record
- Undo record: reverts current record back to it's original state
- Query By Example (QBE): creates query from your form using data in a field as an example.
- Retrieve records: reloads a fresh record set from the database
- Save record: commits changes made to the current record
- Save All records: commits changes to all existing records, including updates, inserts, and deletes.

**active window**

Window which receives keyboard input. Only one window can be active at any time.

## **adapter**

Classes that implement an interface. The `java.awt.event` package provides, as a convenience, a series of listener classes that can be implemented by classes.

## **anchor component**

Component which acts as the reference point when modifying a group of components. When you select multiple components, the last component selected is the anchor component. It has distinct, colored selection handles.

## **applet**

### **Program**

Special type of Java program that you can add to a Web page. To run an applet:

- Add the applet to a Web page and display the Web page in a Java-enabled Web browser.

-or-

- Run the applet from the Applet Viewer. (The Applet Viewer is a tool that Sun provides with the JDK. The Applet Viewer is also included in Visual Cafe and Visual Cafe Database Development Edition.)

### **Class**

Class in the Java API.

## **applet tag**

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 width.

*pixh* is the number of pixels for the height.


Consult an HTML book for more information on the applet tag.


## application

### Program

Java program you can run from a computer that has the Java Virtual Machine.

### Template

Template provided by Visual Cafe when you choose File  New Project

 JFC or AWT Application. This template creates a class that is an extension of the JFrame or Frame class. This class is a good base for a main application window.

B

## **bean**


A component complying with the JavaBeans standard. Also called a JavaBean or a JavaBeans component. A bean is a reusable component that can be visually manipulated in a builder tool. The minimum requirements are that it can be instantiated (it is not an abstract class or interface) and has a public class constructor method that takes zero parameters (it has a null constructor). It can also have properties and events, implement the serializable or externalizable interface, and follow method signature rules so it can be introspected.

## **boolean expression**

Expression that evaluates to true or false.

## Breakpoints window

Window that contains a list of all breakpoints in a project. Use this window to add, remove, or modify breakpoints. You can set simple breakpoints that stop execution at a certain line or method, or conditional breakpoints based on an expression.

To see the Breakpoints window, choose View  Breakpoints.

**bytecode**

Machine-independent code generated by the Java compiler and executed by the Java interpreter.

C

**call stack**

An area reserved in memory by the compiler to keep track of all method calls that are made.

## Calls window

Debugger window that shows all the active method calls leading to the current process. 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 previous methods called are below it. This window is useful for following the flow of your code, for example.

To see the Calls window, View  Call Stack.

## **class**

Collection of variables and methods that you can use to define an object. The variables define the class structure. The methods define the class behaviors.

When compiled as a bytecode program (versus a native Win32 executable), a Java source file becomes one or more class files.

## Class Browser

A three-pane window that lists the Java classes in your project and the methods and data members contained within each class:

- Classes pane — classes
- Members pane — methods and data in the selected class
- Source pane — source code of the selected member

Both the Classes and Members panes support keyboard incremental searches. As you type the name of a class or member, the list of matching items is refined until the class or member you want is automatically selected. You can display classes and members in a variety of views and filter the items that are displayed. The Source pane provides the same editing features as the Source window, while ensuring that you do not unintentionally change code outside of the object's scope.

To see the Class Browser, choose View  Class Browser.

**client machine**

Machine that communicates with a dbANYWHERE Server machine to access a Data Source.

## **client software**

Software that communicates with a database engine such as:

- Oracle Server client software
- Microsoft SQL Server client software
- ODBC32 Administrator


## component

A JavaBeans component.

- A *visual component* is a user interface element, such as a window, menu, button, and so on, that is visible at run time and appears in the Visual Cafe Project window and Form Designer. It extends from the Java Component class and has a screen position, a size, and a foreground and background color.
- An *invisible component* 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. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not effect the form layout.
- Some components can contain other components, such as an application window containing a button; these components are called *containers*. In the Objects view of the Project window, 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 (called Visual Cafe *forms*), while the components in the containers are Java code within the container Java file.
- To create your user interface, you can display a form in the Form Designer, then drag onto the form a variety of components from the Component Library or Palette; you assign the component properties in a separate Property List window, and add interactions between components with the Interaction Wizard. Visual Cafe automatically creates the Java code for you during this design process.
- Components are like controls in C++.

## Component Library

Collection of components that you can add to your applet or application. When you create a project template or add a component to the Component Palette, Visual Cafe adds the component to the Component Library. To see the Component Library:

View  Component Library

## **Component Palette**

Palette that provides easy access to components. You can place any component in the Component Library on the Component Palette.

**conditional breakpoint**

Breakpoint that lets you stop program execution when a specified expression evaluates to true.

## **connection statistics**

DC messages which occur when a client disconnects. The format for a DC message is:

DC, <client IP address>, <initial connect time>, <connect time duration>, <number of requests processed>

where:

- <initial connect time> is in second since 1970
- <connect time duration> is in seconds

## ConnectionInfo

When you use a database wizard or drag a data source item to a project, Visual Cafe Database Edition creates a ConnectionInfo object, showing the database you are connected to. It includes the default user name and password, which you can remove. The object supports multiple queries and updates simultaneously, reducing database resource requirements.

## **ConnectionManager**

A bean, used with the JDBC API, as a logical container for JdbcConnection beans. Dragging a Data Source item from dbNAVIGATOR to a project creates ConnectionManager and JdbcConnection objects, which manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source. You can also drag ConnectionManager objects to the Component Library.

## **container**

Component that can contain other components. For example, Applet and Panel are containers.

A top-level container, also called a Visual Cafe form, 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.

**context menu**

Menu that appears when you click the secondary mouse button in a Visual Cafe window. A context menu is often called a pop-up menu.

## **custom code**

Java code that Visual Cafe has not automatically created for you.

D

## **data binding**

The mechanism by which data is automatically passed between components and the data source. The data binding is the same whether your data is being obtained from the JDBC API or the dbANYWHERE API. The components adhere to a standard interface for the binding. If the component is not aware of the data-binding interfaces, the binding can be controlled by a mediator bean.

## **databound**

also known as DataBound, Data Bound, or data bound. Components and properties that do not need additional coding to work with databases. Formerly called dbAWARE. Databound components contain properties that allow you to bind the component to a database row, column, table, or result set. You may also bind to a Stored Procedure, Calculation, or Validation Adapter.

## **databound component**

Formerly called dbAWARE component. Visual Cafe component that has additional properties for binding the component to a database row, column, table, or result set.

## **datasource**

Also known as data source. Contains the information for creating a database connection, like the name of the database, its server, and its network location. A datasource is what identifies a database to an ODBC- or JDBC-compliant database application. Before you can use your database in a Visual Cafe project, you must create its data source. Sometimes a datasource is called a DSN (Data Source Name). dbANYWHERE also supports datasources, see the dbANYWHERE Help or User's Guidel for more information.

## **dbANYWHERE (naming conventions)**

dbANYWHERE Server, dbANYWHERE	Middleware database software that supports three-tier architecture for database access. Symantec dbANYWHERE is a database connectivity server that provides an implementation of the JavaSoft JDBC API as well as its own powerful java database connectivity API, called the dbANYWHERE API.
dbANYWHERE Server machine	Hardware on which dbANYWHERE software is running

## **dbANYWHERE API**

An API developed by Symantec for accessing databases using the dbANYWHERE Server. It is an SQL-level database protocol that makes use of the extensions provided by dbANYWHERE. Symantec dbANYWHERE is a database connectivity server that provides an implementation of the JavaSoft JDBC API as well as its own powerful Java database connectivity API, called the dbANYWHERE API.

## **dbAW subprotocol**


A JDBC subprotocol supported by dbANYWHERE, which is a JDBC driver. Another JDBC subprotocol supported by Visual Cafe Database Edition is ODBC, which is used by the free JDBC-ODBC bridge (another JDBC driver). The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax:

***jdbc:dbaw://host-name-or-IP-address:port-number/data-source-name***

When attempting to connect to a JDBC driver, the Java program makes a connect call, passing the URL to the driver. If the driver recognizes the information in the URL, including the subprotocol, the connection can be made.

## dbNAVIGATOR

Browser window which shows the servers, data sources, and contents of the data sources that are connected to those servers. You can also use the dbNAVIGATOR perform drag-and-drop operations on your forms and components. To see the dbNAVIGATOR:

View  dbNAVIGATOR

## **debug mode**

Temporary suspension of a running program. In debug mode, you can edit code, edit forms, set or clear breakpoints, debug threads, and view the calls on the call stack. After making any edits, restart program execution to see the effect of the changes.

## **Debug toolbar**

Toolbar that has buttons for debugging.

## **Debugging window**

Window for debugging an applet or application. The Debugging window is a special mode of the Source window: It is the Source window while you are debugging.

## **declaration**

Statement that establishes an identifier and associates attributes with it, without necessarily reserving its storage for data or providing the implementation for methods.

**definition**

Declaration that reserves storage for data or provides implementation for methods.

**design time**

Time during which you build an applet or application in the development environment.

**development machine**

Machine that combines a client to a database server and a Java development environment such as Visual Cafe Database Development Edition.

## **dialog box**

Window with a title bar that can receive and process input from a user.

- Applets cannot use dialog boxes.
- Dialog boxes are not suitable for an application's main window.
- Dialog boxes can contain components, but not menus.
- Use dialog boxes for temporary windows.

## **DML**

Data Modification Language, a subset of SQL.

E

**event**

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**

Event listener interface that is implemented as a class.

## **event binding**

Code that enables an object to receive and process an event. It is made 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.

## **event handler**

A method that is 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 from the Events/Methods pull-down menu of the Source window. An event binding is made 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. 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.

**exception**

An event that occurs during the execution of your program that interferes with, disrupts, or stops the normal flow of instructions.

F

**foreign key**

Column or columns in one database table whose values match the primary key in another table. Foreign key columns may or may not be defined as Unique or NOT NULL.

## **form**

A component that can only appear at the top level in the Objects view of the Project window. Applet, Frame, Window, and some dialog components are forms. In the Component Library, the Visual Cafe forms are all in the Forms group. When you open a form in the Form Designer, the form boundaries are the bounds of the window. You can position other components, such as text, buttons, and graphics, on the form; these components are contained by the form.

## **Form Designer**

Window that visually displays form components so you can design the user interface of the form.

**frame**

Window which can contain components and menus.

G


## **garbage collection**

Automatic detection and freeing of memory that is no longer in use. The Java run time system performs garbage collection so you don't have to explicitly free the memory associated with objects and other data.

H

## Hierarchy Editor

Window that displays the class hierarchy for your applet or application. To see the Hierarchy Editor:


View  Hierarchy Editor




**identifier**

Name of an item in a Java program.

## 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. You enable this feature by choosing Tools  Environment Options

 Debugging.

## **inheritance**

Concept wherein classes automatically contain the variables and methods defined in their superclasses.

## **inner class**

A class that is 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**

Data item 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 components on each instance can be set to different values.

## **interaction**

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)
- Trigger event
- 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**

The ability to read JavaBeans classes directly with the Core 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.

## invisible component

Also called a non-visual component. 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. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not effect the form layout. To toggle between displaying and hiding invisible components at design time:

Layout  Invisibles

J

## **Java API: Java Application Program Interface**

API provided by the JDK. For more information, see the Java Web page ([java.sun.com](http://java.sun.com)).

**Java file**

File that contains components and Java source code.

## **Java Archive (JAR) file**

Compressed archive file that complies with the JavaBeans standard. It is the primary method for delivering JavaBeans components. For example, a JAR file can contain one or more related beans, and any support files, including classes, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. You can deploy applets and applications from a JAR file. 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.

## **Java VM: Java Virtual Machine**

Virtual machine provided with the JDK (Java Development Kit). The machine contains:

- Bytecode translator that converts a downloaded binary Java file into instructions that the client machine can execute.
- Library routines that a Java applet calls.

For more information, see the Java Web page ([java.sun.com](http://java.sun.com)).

## **JavaBeans**

A standard for creating portable, cross-platform components.

## **JavaBeans component**

A component complying with the JavaBeans standard. Also called a bean or JavaBean. A bean is a reusable component that can be visually manipulated in a builder tool. The minimum requirements are that it can be instantiated (it is not an abstract class or interface) and has a class constructor method that takes zero parameters (it has a null constructor). It can also have properties and events, implement the serializable or externalizable interface, and follow method signature rules so it can be introspected.

## **Javadoc**

Javadoc is a way to generate HTML documentation directly from source code. It is made up of two parts:

- Special comments with embedded tags in Java source code that help describe how to use a class, and its methods and fields. The Javadoc comments are also used by the Java compiler to report to the user if a class or member is deprecated.
- A utility that scans this Java source code and generates the 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.

## **JDBC API**

An API developed by JavaSoft as a standard SQL-level database protocol. JDBC, based on ODBC, is a database extension to Java. The Visual Cafe Database Edition implementation of the JDBC API includes the Symantec JDBC components, in addition to the base JDBC. While it is not an acronym, JDBC is often thought of as the Java Database Connectivity protocol.

## JDBC-ODBC bridge

A free JDBC driver. It supports an ODBC subprotocol of the JDBC API. The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax. For example:

**jdbc:odbc:***//subname*

**jdbc:dbaw:***//host-name-or-IP-address:port-number/data-source-name*

When attempting to connect to a JDBC driver, the Java program makes a connect call, passing the URL to the driver. If the driver recognizes the information in the URL, including the subprotocol, the connection can be made.

## **JdbcConnection**

A bean, used with the JDBC API, that represents a connection to a database. It has properties for establishing connection parameters in accordance with JDBC. Dragging a Data Source item from dbNAVIGATOR to a project creates ConnectionManager and JdbcConnection objects, which manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source. You can drag either of these objects to the Component Library.

**JDK: Java Developer's Kit**

Tools for developing Java applets and applications. The JDK is included in Visual Cafe and Visual Cafe Database Development Edition. It is also available on the Java Web page ([java.sun.com](http://java.sun.com)).

## **join**

Any database query that produces data from more than one table. As its name suggests, a join brings together data from the specified tables.

K

L

## **layout**

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 resolutions.

## **list binding**

Data binding that lets you display data from multiple database columns. List binding is helpful for populating list boxes.

## **local variable**

Data item 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.

M

## **macro**

Sequence of keystrokes. The Source editor's macro facilities let you record, save, and edit macros.

## 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. It is made of sections separated by empty lines. Each section has one or more headers. Each header uses a *attribute:value* format where *attribute* specifies various attributes of the archive contents and *value* is the relative name of the file being described. Attributes used include Manifest-Version, Name, Java-Bean, Digest-Algorithms, and XXX-Digest (related to Digest-Algorithms). See Sun Microsystem's Manifest File specification for more information.

## **master/detail relationship**

A one to many relationship between two database tables. The join property of the detail QueryNavigator or RelationViewPlus component defines the column of the master view which is the common attribute in the one to many relationship between the tables. For instance, you might want to generate a result set which enumerates all customers who purchased a product called the Spectacular Widget. You create this master/detail relationship by joining the Spectacular Widget column of a Sales table (which also has columns for numbers of sales and price) in an SQL query that returns a list of all the customers in a Customer database table who purchased a Spectacular Widget.

## **mediator**

Bean that lets you make a component database-aware. It is a bridge between the component and the QueryNavigator or RelationViewPlus components.

At design time, you can add a Mediator component to a form in a project and set its properties for a particular component on the form. If you are creating a JavaBeans component or have access to the component Java code, you can add a mediator to the component code, thereby encompassing its functionality within your component.

## **member**

Variable or method defined in a class.

## **method**

Behavior that acts on an object. A method is defined in a class.

- Class method: Method invoked using a class name.
- Instance method: Method invoked using the name of an instance (object).

## Messages window

Window that displays all Visual Cafe 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. To see the Messages window:

View  Messages

## **metadata**

Data about data. In a database, the schema contains metadata that describes the data in the tables. In CORBA, the interface repository contains metadata that describes the objects that have been registered with the ORB.

## **mnemonic**

A character, word or string which is intended to be easier to remember than the thing it stands for. Mnemonics in Visual Cafe are the underlined character in a menu, menu item, or button. When used with the ALT key, the underlined character performs same action as if you clicked on the item with the mouse. For example, ALT+F opens the File menu. A mnemonic property is the property that allows you to create a mnemonic in a menu, menu item or button.

## **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 does not require you to take some action before the focus can switch to another window or dialog box.

N

## **nested class**

A class that is included within the body of another class, even within a method (called a local class). A nested class is also called an inner class. This feature is new for JDK 1.1 and is useful for creating adapter classes.

O

## **ODBC**

Open DataBase Connectivity, a standard interface for communicating with databases. The dbANYWHERE server uses this interface to communicate with many of the databases it supports. The dbANYWHERE server also communicates to other major database types through direct native calls to the client APIs (application programming interfaces) for those databases.

## ODBC subprotocol

A JDBC subprotocol supported by the JDBC-ODBC bridge, which is a free JDBC driver. (Each JDBC driver supports certain subprotocols.) Another subprotocol supported by Visual Cafe Database Edition is dbAW, which is used by Symantec dbANYWHERE. The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax:

**jdbc:odbc://subname**


When attempting to connect to a JDBC driver, the Java program makes a connect call, passing the URL to the driver. If the driver recognizes the information in the URL, including the subprotocol, the connection can be made.

## **object**

Instance of a class.

## Objects view

View of a project that displays only visible components. To toggle between displaying and hiding hidden components at design time:

Layout  Invisibles

**optimistic concurrency**

Locking technique that maximizes concurrency. The database management system (DBMS) locks data that is being modified but leaves alone data that is being viewed.

## **overloading**

Using one identifier to refer to multiple items in the same scope. In Java, you can overload methods but not variables or operators.

## **overriding**

Providing a different implementation of a method in a subclass of the class that originally defined the method. Overridden methods have the same name but take different parameters.


P

## **package**

Group of classes and interfaces. 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 are 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 is not in uppercase letters, it is 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 are in different packages, there is no naming conflict.

## Packages view

View of a project that displays both visible and hidden components. To toggle between displaying and hiding hidden components at design time:

Layout  Invisibles

## **panel**

Container that you can add to another container. A panel doesn't have a visible border. You can use a panel to group a window into logical regions.

**ping: Packet INternet Groper**

Network message that a computer transmits to check for the presence, connectedness, and responsiveness of another computer. The other computer responds by echoing the message back to the first computer.

**primary key**

Column or columns in a database table that uniquely identify a record. A value in a primary key column cannot be NULL and must be unique.

**project**

Set of components and code that comprise an applet or application.

**project template**

Collection of components that you can use as the foundation for an applet or application. When you choose a template for a new project, the new project inherits all of the template's components. For example, you can create project templates for Web sites, single documents, and applets.

## **Project window**


Window that displays a project's objects or packages depending on which tab you select.

## **projection binding**

Data binding to one row in a database column.


## property

Attribute of a component. Properties define component characteristics such as size, color, label, or the state of an object, such as enabled or disabled. The Property List displays the properties of a project's components. To see the Property List:

View  Property List

## Property List

Window that displays a component's properties. To see the Property List:

View  Property List

Q

## QueryNavigator

A bean, used with the JDBC API, that manages a set of records. Dragging a Data Table item from dbNAVIGATOR to a form in a project creates a top-level RecordDefinition object and a QueryNavigator object, which is contained by the form. You can think of the RecordDefinition component as the data, and the QueryNavigator component as a way of looking at the data. After you have a QueryNavigator and RecordDefinition component in a project, you can create a master/detail relationship. You can drag a QueryNavigator object to the Component Library.

R

## **RecordDefinition**

A bean, used with the JDBC API, that is used to define and access a row of data in a database table. Dragging a Data Table item from dbNAVIGATOR to a form in a project creates a top-level Record Definition object and a QueryNavigator object, which is contained by the form.

## **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. Two types of resource bundles are list type, which is a Java source file, and properties type, which is a text file.

**refresh**

Updates the dbNAVIGATOR window to reflect changes made to database tables and columns it is displaying.

## regular expression syntax

Syntax that lets you perform regular expression matching. Regular expressions are wildcard characters. The pattern you search for can be a text string or a regular expression.

### Wildcard characters

?	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.
[...]	Any of the characters listed between [ and ]. You can use a hyphen (-) to specify 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 [~ and ]. You can use a hyphen (-) to specify 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 wild card character. For example, you can use \* to search for an asterisk or \\ to search for a backslash character.
\t	Tab character.
\f	Formfeed character.

### RelationViewPlus

Database-aware component that shows a view of a database and is used with the dbANYWHERE API. It provides a simple, property-driven solution for defining and maintaining a result set. It is invisible at run time but contains methods used to handle navigation and data manipulation operations on the data in the result set. (The “Plus” was added when the data binding changed in later versions of Visual Cafe Database Development Edition.)

**result set**

Set of data which is returned by a query on a database(s). An example of a result set might be the data from all the rows in an employee database table which have 200 as their department identification number.

## **run mode**

Mode wherein your applet or application is running. In run mode:

- You can interact with your program as a user.
- Visual Cafe replaces the Project Menu with the Debug Menu.

## **run time**

Time when your applet or application is running. At run time:

- You can interact with your program as a user.
- You can pause the application and start debugging by pressing `CONTROL+BREAK`.

S

## **scope**

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' 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.

## **Session**

For the dbANYWHERE API, when you use the database wizard or drag a dbANYWHERE Server item to a project, Visual Cafe Database Edition creates a Session object, indicating the dbANYWHERE Server you are connected to through sockets.

**server**

- Database server: Computer that stores and manages a database.
- dbANYWHERE Server: Computer that runs dbANYWHERE, the middleware database software from Symantec which supports three-tier architecture for database access.
- File server: Computer that stores programs and data files.
- Web server: Computer that stores and sends out Web pages in response to HTTP requests from Web browsers.

## **servlet**



Java component that resides on a Java-enabled server and can dynamically extend server-side functionality. A servlet can provide services using a request-response paradigm; it has no 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. .

## **Source pane**

Pane in the Source window and Class Browser that lets you view and edit code.

## Source window

Window that displays and lets you edit a Java source file, an HTML file, or a text file. The Source window is also available when you are debugging. To see the Source window:

- Select a file in the Project window, then choose Object  Edit Source, or right-click and choose Edit Source.
- Double-click a file in the Packages or Files view of the Project window.
- Open a file by choosing File  Open.

## **SQL**

Standard query language which is used for retrieving information from a relational database. The acronym stands for Structured Query Language.

## **SQL statement**

SQL string used to query a database and thereby define a view. If the string is in the standard SELECT statement format as shown below, the Visual Cafe Database wizards and Join Definition dialog box can parse the string and edit it for you. If the string is not in the standard SELECT statement format, you need to use the SQL Statement dialog box to edit the string manually.

### **Standard format**

```
SELECT column[, column ...] FROM table WHERE condition
```

### **Example**

```
SELECT emp_fname, emp_lname, location_id FROM emp_info WHERE location_id BETWEEN 10 AND 20
```

## **stored procedure**

Collection of SQL statements that is named and precompiled.

Procedures and triggers store procedural SQL statements in a database for use by all applications.

Procedures and triggers can include control statements that allow repetition (LOOP\_statement) and conditional execution (IF\_statement and CASE\_statement) of SQL statements.

Procedures are invoked with a CALL\_statement, and use parameters to accept values and return values to the calling environment. Procedures can also return result sets to the caller. Procedures can call other procedures and fire triggers.

Triggers are associated with specific database tables. They are invoked automatically (fired) whenever rows of the associated table are inserted, updated or deleted. Triggers do not have parameters and cannot be invoked by a CALL statement. Triggers can call procedures and fire other triggers.

User-defined functions are one kind of stored procedure that returns a single value to the calling environment. User-defined functions do not modify parameters passed to them. They broaden the scope of functions available to queries and other SQL statements.

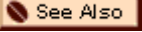
T

**thread**

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

Debugger window that displays all the existing threads your program has created and the state of each thread. You can pause individual threads, which causes their execution to cease temporarily while all other threads continue to execute, and 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. Double-clicking a thread in this window updates the Calls window, so it reflects the scoping level of the thread.

To see the Threads window, choose View  Threads.

## **top-level component**

A component that can only appear at the top level in the Objects view of the Project window. Also called a *form*. Applet, Frame, Window, and some dialog components are top-level components. In the Component Library, the Visual Cafe top-level components are all in the Forms group. When you open a top-level component in the Form Designer, the component boundaries are the bounds of the window. You can position other components, such as text, buttons, and graphics, on the top-level component in the Form Designer; these components are contained by the top-level component.

## transaction isolation levels

The *isolation level* defines degree to which the operations in one transaction are visible to the operations in a concurrent transaction. Isolation levels prevent some or all inconsistent behavior and can be different for each connection. All isolation levels guarantee that each transaction will execute completely or not at all, and that no updates will be lost. This ensures recoverability at all times, regardless of the isolation level.

There are three types of inconsistency that can occur during the execution of concurrent transactions:

**Dirty read** — Transaction A modifies a row. Transaction B then reads that row before transaction A performs a COMMIT. If transaction A then performs a ROLLBACK, transaction B will have read a row that was never committed.

**Non-repeatable read** — Transaction A reads a row. Transaction B then modifies or deletes the row and performs a COMMIT. If transaction A then attempts to read the same row again, the row will have been changed or deleted.

**Phantom row** — Transaction A reads a set of rows that satisfy some condition. Transaction B then executes an INSERT, or an UPDATE (that generates one or more rows that satisfy the condition used by transaction A) and then performs a COMMIT. Transaction A then repeats the initial read and obtains a different set of rows.

Here are the types of transaction isolation levels:

TRANSACTION\_NONE — Transactions are not supported.

TRANSACTION\_READ\_UNCOMMITTED — Lets dirty reads, non-repeatable reads, and phantom reads occur.

TRANSACTION\_READ\_COMMITTED — Prevents dirty reads, but not non-repeatable reads and phantom reads.

TRANSACTION\_REPEATABLE\_READ — Prevents dirty reads and non-repeatable reads, but not phantom reads.

TRANSACTION\_SERIALIZABLE — Prevents dirty reads, non-repeatable reads and phantom reads.

See the JDBC API for more information.

U

V

## **ValueTip**

Interface element in Visual Cafe that displays the value of the variable that is under the cursor in the Source window in debug mode.


## **variable**

Structure in memory which holds data that has been assigned to it. A variable that is defined in a class defines the class' structure.

- Class variable: Variable associated with a class and not with a particular instance of the class.
- Instance variable: Variable associated with an instance of a class (an object).

## Variables window

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.

To see the Variables window, choose View  Variables.

## **view**

### **In a development environment**

Window that shows a particular type of information.

- Global view: View that is accessible from the View menu. The global views are: Breakpoints window, Calls window, Class Browser, Component Library, Component Palette, Hierarchy Editor, Output window, Property List, Threads window, Variables window, Watch window.
- Local view: View that is local to a project. The local views are: Project window, Source window.

### **For Data Sources**

A QueryNavigator and a RecordDefinition component, or a RelationViewPlus component, represent a view. A view has three parts:

- It is a collection or set of objects that contain data. You can set focus to one member of the set at a time by scrolling forward and backward through the set.
  - A QueryNavigator component has next and previous methods.
  - A RelationViewPlus component has next and previous methods.
- There is a data model associated with the objects that contain data. Each object has a set of values that can be indexed numerically.
  - A QueryNavigator component has symantec.itools.db.beans.jdbc.RecordDefinition objects. These objects obtain their data model on their own. The Table property is a property of RecordDefinition.
  - A RelationViewPlus component has symantec.itools.db.pro.Record objects. These objects obtain their data model from the SQL statements that define the RelationViewPlus.
- There is a relationship between objects in the view.
  - RecordDefinition components contained by a QueryNavigator component are related in that they meet the select criteria imposed by the Filter property of the QueryNavigator.
  - Records contained by a RelationViewPlus component are related in that they all meet the select criteria in the defining SQL statement.

For detail views, the objects (RecordDefinition or RelationViewPlus' Record) are further related in that they meet the join criteria.



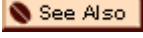
## **Visual Cafe window**

Main development environment window that contains the menu bar and toolbars.

W

## Watch window

Debugger window that displays the value of a variable or expression you enter. The values update when you pause execution or step through code. You can also examine the contents of a class member. To watch a variable accessible to a method, drag a variable from the Variables window to the Watch window. You can modify values directly in this window and continue debugging without having to stop and restart the debug session.

To see the Watch window, choose View  Watch.

## **window**

Area that has no borders and no menu bar.

## **workspace**

Saved window arrangement. Your workspace is saved automatically while you work.

X

Y

z

## **VISUAL Cafe 1.0 and 1.01 online help project**

### **Change history:**

3/10 - removed all windows except for visual and substep, updated files, replaced defin.doc with gloss.doc.

3/11 - added back compo.doc, events.doc, and prop.doc for the Pro release around 3/20.

---

**See note for building Visual Cafe Pro with this project, below.**

**Do not** add topics to this file. This is the base .doc file for help and is used For project information only.

This project consists of these files:

editors.doc	Tasks and light overview
F1_editor.doc	F1 help topic – menus, dialogs, window, etc
gloss.doc	Glossary of terms
debug.doc	Debugger tasks and overview
compo.doc	Components reference
events.doc	Events reference
props.doc	Properties reference

This project includes the following:

vcguide.hlp	To allow Alink searches
alias.txt	As an alias file. Maps engineering help ID strings to help topic
`	IDs for context-sensitive help
helpid.txt	A file provided by the engineers. Simply update the file exactly as you get it from the engineers. Contains the hex number that the software calls and the help ID that the software thinks that is should look for. Since our topic IDs don't match, that is why we have the alias.txt file.

New for 1.01

ctask.doc	Component tasks (generic components only) dbAWARE component tasks are in vcafe.pro.hlp
-----------	---

There is a summary topic for each reference doc. Refrain from linking to much to reference .doc files since they change often.

The other piece of the Visual Cafe online help and online document project is the Visual Cafe User's Guide. This project is vcguide.hpj. The online help (vcafe) links to the User's Guide through the User Guide button. The User Guide links to online help through the How To button. To enable this cross .hlp file linking, you must list both files in the Content Tab Composer>>Contents>>Setup...>> Link Files option. The Alinks of other .hlp files don't show up in the A word listing. You have to know the A word and type it in manually.

**WARNING: For this project the Topic ID and A Search Word fields in the Topic definition must**

be EXACTLY the same and only use ONE A Search Word.

Context sensitive help is done by including the engineer's helpid.txt file into the help compile. The mapping between the engineer's ID and the Topic ID is done in an included Alias file called alias.txt.

Other notes:

- since this is a multiple writer project. Everyone should use shared templates robohelp.doc and robortf.dot

### **Building the Visual Cafe Pro help system**

The VC Pro version of vcafe.hlp uses all of the standard .doc files from the VC version, BUT the vcafe.cnt and vcafe.hpj are customized..(Masters are in the Doc source directory | Pro | ComboHelp)

This is because the .cnt current displays the vcafe.pro.cnt. And, the .hpj has to be modified so that the help window header displays "Visual Cafe Pro Online Help"

To build the vcafe.hlp file for Visual Cafe Pro:

1. Create a directory.
2. Copy in the Visual Cafe vcafe help system files.
3. Copy in the current User Guide (vcguide.hlp and cnt), VC Pro (vcafepro.hlp and cnt)
4. From the source "combohelp" vcafe directory, copy in the vcafe.hpj and vcafe.cnt. Overriding the standard VC version.
5. Compile

\*\*\* We need to sit down and figure out a better way to do this. The problem is that Visual Cafe, and hence Pro, look in "vcafe.hlp" for all F1 topics. To override this the engineers have to hardcode in the name of the help file. Therefore, VC and VCPro have to use the same file name of all UI F1 (we do call vcafepro.hlp for all F1 that is not common to VC.) And the cnt files have to be the same name as the corresponding .hlp (I think, there may be a workaround.)

## Events.doc

8/6/97 Update/check for 1.1

See Also

See Also

See Also

See Also

See Also

See Also

See Also

See Also

See Also

Properties

Property Summary

Events

See Also

Components List

Event List

db Components

API (Standard)

API (databound)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



An event is an asynchronous signal that a source program element sends to target program element to notify the target that some specific behavior has occurred.

### The New 1.1 Event “Delegation” Model

In the Java event delegation model events are sent from a “Source” to “Listener.” A Source is an object that generates events, like an AWT Component. A Listener is any object that implements the appropriate listener interface so that it may receive events. 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 anytime 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 only handled by Frame and Dialog objects. Semantic events are higher-level events that usually indicate 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 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.

### Usage

In Visual Cafe, you can usually route an event to a target component and specify the action to be taken, via the Interaction Wizard. The Interaction Wizard automatically writes event-related code. It creates new listeners as needed, adds them to the appropriate components, and performs routine event-processing tasks.

### The Old 1.0 Event “Inheritance” Model

The old event inheritance model should not be used for new code. It has been replaced with the new event 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.

Java defines the new model events it supports in the `java.awt.Event` class, which extends the `Object` class.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Events fall under the following categories:

## SEMANTIC EVENTS

### Action Events (java.awt.event.ActionEvent)

[ACTION\\_PERFORMED](#)

### Adjustment Events (java.awt.event.AdjustmentEvent)

[ADJUSTMENT\\_VALUE\\_CHANGED](#)

### Item Events (java.awt.event.ItemEvent)

[ITEM\\_STATE\\_CHANGED](#)

### Text Events (java.awt.event.TextEvent)

[TEXT\\_VALUE\\_CHANGED](#)

## LOW-LEVEL INPUT EVENTS

### Ancestor Events (com.sun.java.swing.event.AncestorEvent)

[ANCESTOR\\_ADDED](#) [ANCESTOR\\_MOVED](#)  
[ANCESTOR\\_REMOVED](#)

### Caret Events (com.sun.java.swing.event.CaretEvent)

[CaretUpdate](#)

### Change Events (com.sun.java.swing.event.ChangeEvent)

[EditingCanceled](#) [EditingStopped](#)  
[StateChanged](#)

### Component Events (java.awt.event.ComponentEvent)

[COMPONENT\\_MOVED](#) [COMPONENT\\_RESIZED](#)  
[COMPONENT\\_SHOWN](#) [COMPONENT\\_HIDDEN](#)

### Container Events (java.awt.event.ContainerEvent)

[COMPONENT\\_ADDED](#)[COMPONENT\\_REMOVED](#)

### Focus Events (java.awt.event.FocusEvent)

[FOCUS\\_GAINED](#) [FOCUS\\_LOST](#)

### Key Events (java.awt.event.KeyEvent)

[KEY\\_PRESSED](#) [KEY\\_RELEASED](#)  
[KEY\\_TYPED](#)

### Mouse Events (java.awt.event.MouseEvent)

[MOUSE\\_CLICKED](#) [MOUSE\\_DRAGGED](#)  
[MOUSE\\_ENTERED](#) [MOUSE\\_EXITED](#)  
[MOUSE\\_MOVED](#) [MOUSE\\_PRESSED](#)  
[MOUSE\\_RELEASED](#)

### PopupMenu Events (com.sun.java.swing.event. PopupMenuEvent)

[PopupMenuCanceled](#) [PopupMenuWillBecomeVisible](#)  
[PopupMenuWillBecomeInvisible](#)

### Window Events (java.awt.event.WindowEvent)

[WINDOW\\_ACTIVATED](#) [WINDOW\\_CLOSED](#)  
[WINDOW\\_CLOSING](#) [WINDOW\\_DEACTIVATED](#)  
[WINDOW\\_DEICONIFIED](#) [WINDOW\\_ICONIFIED](#)

WINDOW\_OPENED

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A databound component will automatically update the database when certain triggering events occur. The table below notes which events trigger these updates for which components.

The entries marked with “Default” indicate the default event that the component uses to trigger database updates. The entries marked with “X” indicates events that the component can optionally use as a trigger.

	<u>ACTION</u>	<u>FOCUS_LOST</u>	<u>ITEM</u>	<u>KEY_PRESSED</u>	<u>TEXT</u>
<b>CheckBox</b>		X	Default	X	
<b>ComboBox</b>	X	X		Default	
<b>FormattedTextField</b>		X		X	Default
<b>ListPlus</b>	X	X	Default		
<b>TextArea</b>		X		X	Default
<b>TextField</b>	X	X		X	Default

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This semantic event typically indicates some action by the user on a component. They are general purpose signals whose interpretation changes depending on the source of the event.

The user action that causes this event is different for different objects. For example, a check box component's control the action event is triggered when the user clicks the check box, but for a list component, the action event is triggered when the user double-clicks an item in the list.

This table describes what makes common components generate Action Performed events. The Component column contains the component class name. The Command Name column contains the name of the command as retrieved by the `ActionEvent`'s `getActionCommand` method. Command names flagged with a "\*" can be overridden by the user by using that component's `setActionCommand` method. The I.W. Name column contains the name used by the Interaction Wizard for the command. And finally, the User Action column describes how the command typically gets generated.

Standard Java components:

Component	Command Name	I.W. Name	User Action
Button	* (label)	Clicked	User clicked in the Button.
List	(item text)	DblClicked	User double-clicked on a list item.
MenuItem	* (label)	Action	User picked a MenuItem.
TextField	(field text)	EnterHit	User pressed the ENTER key while the TextField had the focus.

Symantec components:

Component	Command Name	I.W. Name	User Action
Calendar	* (null)	Action	The selected date changed.
ComboBox	* (null)	Action	List item state changed.
DirectionButton	*	(null)	Action User pressed the button.
HorizontalSlider	*	HSliderMoved	Action Slider value changed.
ImageButton	* (null)	Action	User pressed the button.
ImageListBox	DoubleClicked	Action	List item double-clicked.
	ImageSelected	Action	List item image selected.
InvisibleButton	*	ButtonPressed	Action User pressed the button.
LabelButton	* (null)	Action	User pressed the button.
ListPlus	(item text)	DblClicked	User double-clicked on a list item.
MultiList	* RowSelected:n	Action	Nth row selected, where n is the row's zero-relative index.
RollOverButton	*	(null)	Action User pressed the button.
SlideShow	nextImage	SlideChanged	The next image has been shown.
	previousImage	SlideChanged	The previous image has been shown.
Spinner	ScrollUp	Action	Increment button pressed and value incremented.
	ScrollDown	Action	Decrement button pressed and value decremented.
StateCheckBox		(null)	Action User pressed the checkbox.
Timer	* (null)		Timer delay has expired.
TreeView	(node text)	Action	User expanded/collapsed a node, or double-clicked, or pressed ENTER.
VerticalSlider	* VSliderMoved	Action	Slider value changed.

#### Listening for the event with code

Listen for this event using an object with the `ActionListener` interface. Register the listener with the event-generating object by using that object's `addActionListener` method. Handle events as they occur in the listener's `actionPerformed`

method. Stop listening for this event by using the event-generating object's `removeActionListener` method.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This semantic event is generated when the value of an object that implements the `Adjustable` interface, like a scrollbar, changes. It indicates the component's value has changed.

To find out why the value changed, use the `getAdjustmentType` method. The value returned indicates what occurred to make the value change. For Scrollbars:

<b>Adjustment Type</b>	<b>User Action</b>
BLOCK_DECREMENT	User "paged" up by clicking in the area between the thumb and the up arrow.
BLOCK_INCREMENT	User "paged" down by clicking in the area between the thumb and the down arrow.
TRACK	User selected a specific value by dragging the thumb.
UNIT_DECREMENT	User scrolled up one line by clicking the up arrow.
UNIT_INCREMENT	User scrolled down one line by clicking the down arrow.

#### **Listening for the event with code**

Listen for this event using an object with the `AdjustmentListener` interface. Register the listener with the event-generating object by using that object's `addAdjustmentListener` method. Handle events as they occur in the listener's `adjustmentValueChanged` method. Stop listening for this event by using the event-generating object's `removeAdjustmentListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is added to a container.

**Listening for the event with code**

Listen for this event using an object with the `ContainerListener` interface. Register the listener with the event-generating object by using that object's `addContainerListener` method. Handle events as they occur in the listener's `componentAdded` method. Stop listening for this event by using the event-generating object's `removeContainerListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is moved.

**Listening for the event with code**

Listen for this event using an object with the `ComponentListener` interface. Register the listener with the event-generating object by using that object's `addComponentListener` method. Handle events as they occur in the listener's `componentMoved` method. Stop listening for this event by using the event-generating object's `removeComponentListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is removed from a container.

**Listening for the event with code**

Listen for this event using an object with the `ContainerListener` interface. Register the listener with the event-generating object by using that object's `addContainerListener` method. Handle events as they occur in the listener's `componentRemoved` method. Stop listening for this event by using the event-generating object's `removeContainerListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is resized.

#### **Listening for the event with code**

Listen for this event using an object with the `ComponentListener` interface. Register the listener with the event-generating object by using that object's `addComponentListener` method. Handle events as they occur in the listener's `componentResized` method. Stop listening for this event by using the event-generating object's `removeComponentListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is made visible.

**Listening for the event with code**

Listen for this event using an object with the `ComponentListener` interface. Register the listener with the event-generating object by using that object's `addComponentListener` method. Handle events as they occur in the listener's `componentShown` method. Stop listening for this event by using the event-generating object's `removeComponentListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component is made invisible.

#### **Listening for the event with code**

Listen for this event using an object with the `ComponentListener` interface. Register the listener with the event-generating object by using that object's `addComponentListener` method. Handle events as they occur in the listener's `componentHidden` method. Stop listening for this event by using the event-generating object's `removeComponentListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component receives the focus. A component receives focus either as the result of user action, like clicking the component, or as a result of executed code, for example the `requestFocus` method. A parent container receives the focus only when all visible controls are disabled.

#### **Listening for the event with code**

Listen for this event using an object with the `FocusListener` interface. Register the listener with the event-generating object by using that object's `addFocusListener` method. Handle events as they occur in the listener's `focusGained` method. Stop listening for this event by using the event-generating object's `removeFocusListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a component loses the input focus.

**Listening for the event with code**

Listen for this event using an object with the `FocusListener` interface. Register the listener with the event-generating object by using that object's `addFocusListener` method. Handle events as they occur in the listener's `focusLost` method. Stop listening for this event by using the event-generating object's `removeFocusListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This semantic event is generated when a component item has become either selected or deselected. This event is generated by components: `Checkbox`, `CheckboxMenuItem`, `Choice`, `ImageListBox`, `List`, `MultiList`, and `TreeView`.

To find out why the value changed, use the `ItemEvent`'s `getStateChange` method. The returned value indicates the state change type that caused the event to be generated. A value of `SELECTED` indicates an item is newly selected. A value of `DESELECTED` indicates an item is no longer selected.

#### **Listening for the event with code**

Listen for this event using an object with the `ItemListener` interface. Register the listener with the event-generating object by using that object's `addItemListener` method. Handle events as they occur in the listener's `itemStateChanged` method. Stop listening for this event by using the event-generating object's `removeItemListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the user presses a key while a component has the focus.

**Listening for the event with code**

Listen for this event using an object with the `KeyListener` interface. Register the listener with the event-generating object by using that object's `addKeyListener` method. Handle events as they occur in the listener's `keyPressed` method. Stop listening for this event by using the event-generating object's `removeKeyListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the user releases a key while the component has the focus.

**Listening for the event with code**

Listen for this event using an object with the `KeyListener` interface. Register the listener with the event-generating object by using that object's `addKeyListener` method. Handle events as they occur in the listener's `keyReleased` method. Stop listening for this event by using the event-generating object's `removeKeyListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a key is typed. This indicates a key has been pressed then released.

**Listening for the event with code**

Listen for this event using an object with the `KeyListener` interface. Register the listener with the event-generating object by using that object's `addKeyListener` method. Handle events as they occur in the listener's `keyTyped` method. Stop listening for this event by using the event-generating object's `removeKeyListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the mouse clicks within a component (using the right or left mouse button), or when the SPACE bar is pressed while a button component has the focus.



**Note:** Using the [Action Performed](#) semantec event is the preferred way to trigger an action when a button is clicked. Listen to the Mouse Clicked event to modify and/or consume the event before it reaches the button.

#### **Listening for the event with code**

Listen for this event using an object with the `MouseListener` interface. Register the listener with the event-generating object by using that object's `addMouseListener` method. Handle events as they occur in the listener's `mouseClicked` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when a mouse button is pressed while inside a component, and the mouse then moved while the mouse button is still down. Mouse Drag events are repeatedly generated until the mouse button is released.

#### **Listening for the event with code**

Listen for this event using an object with the `MouseMotionListener` interface. Register the listener with the event-generating object by using that object's `addMouseMotionListener` method. Handle events as they occur in the listener's `mouseDragged` method. Stop listening for this event by using the event-generating object's `removeMouseMotionListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the mouse mouse first enters a component.

**Listening for the event with code**

Listen for this event using an object with the `MouseListener` interface. Register the listener with the event-generating object by using that object's `addMouseListener` method. Handle events as they occur in the listener's `mouseEntered` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the mouse exits a component.

**Listening for the event with code**

Listen for this event using an object with the `MouseListener` interface. Register the listener with the event-generating object by using that object's `addMouseListener` method. Handle events as they occur in the listener's `mouseExited` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the user moves the mouse inside a component with the mouse button not pushed.

**Listening for the event with code**

Listen for this event using an object with the `MouseMotionListener` interface. Register the listener with the event-generating object by using that object's `addMouseMotionListener` method. Handle events as they occur in the listener's `mouseMoved` method. Stop listening for this event by using the event-generating object's `removeMouseMotionListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the mouse button is pressed inside this component.



**Note:** Using the [Action Performed](#) semantec event is the preferred way to trigger an action when a button is clicked. Listen to the Mouse Pressed event to modify and/or consume the event before it reaches the button.

#### **Listening for the event with code**

Listen for this event using an object with the `MouseListener` interface. Register the listener with the event-generating object by using that object's `addMouseListener` method. Handle events as they occur in the listener's `mousePressed` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This low-level input event is generated when the mouse button is released inside this component.



**Note:** Using the [Action Performed](#) semantec event is the preferred way to trigger an action when a button is clicked. Listen to the Mouse Released event to modify and/or consume the event before it reaches the button.

#### **Listening for the event with code**

Listen for this event using an object with the `MouseListener` interface. Register the listener with the event-generating object by using that object's `addMouseListener` method. Handle events as they occur in the listener's `mouseReleased` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the value of a text component changes. This event is generated by components: `TextField` and `TextArea`.

**Listening for the event with code**

Listen for this event using an object with the `TextListener` interface. Register the listener with the event-generating object by using that object's `addTextListener` method. Handle events as they occur in the listener's `textValueChanged` method. Stop listening for this event by using the event-generating object's `removeTextListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a window is activated.

**Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowActivated` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated after a window has been closed by a call to the window's `hide` or `destroy` method. This event halts processing initiated from window components and releases memory to the Java garbage collector. Program processing is not halted unless the window is the parent window to an application or applet.

#### **Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowClosed` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a window's close box is clicked. The listener should explicitly hide or destroy the window in response to this event to close the window.

**Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowClosing` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a window is deactivated.

**Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowDeactivated` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the user restores the size of the window from an icon.

**Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowDeiconified` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the user minimizes a window to an icon.

**Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowIconified` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated the first time a window is made visible.

#### **Listening for the event with code**

Listen for this event using an object with the `WindowListener` interface. Register the listener with the event-generating object by using that object's `addWindowListener` method. Handle events as they occur in the listener's `windowOpened` method. Stop listening for this event by using the event-generating object's `removeWindowListener` method.

## Swing Events

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the source or one of its ancestors is made visible either by `setVisible(true)` being called or by its being added to the component hierarchy. For a source to be visible, all of its parents must be visible and it must be in a hierarchy rooted at a Window.

#### **Listening for the event with code**

Listen for this event using an object with the `AncestorListener` interface. Register the listener with the event-generating object by using that object's `addAncestorListener` method. Handle events as they occur in the listener's `ancestorAdded` method. Stop listening for this event by using the event-generating object's `removeAncestorListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the source or one of its ancestors is moved.

#### **Listening for the event with code**

Listen for this event using an object with the `AncestorListener` interface. Register the listener with the event-generating object by using that object's `addAncestorListener` method. Handle events as they occur in the listener's `ancestorMoved` method. Stop listening for this event by using the event-generating object's `removeAncestorListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the source or one of its ancestors is made invisible either by `setVisible(false)` being called or by its being removed from the component hierarchy. For a source to be visible, all of its parents must be visible and it must be in a hierarchy rooted at a Window.

#### **Listening for the event with code**

Listen for this event using an object with the `AncestorListener` interface. Register the listener with the event-generating object by using that object's `addAncestorListener` method. Handle events as they occur in the listener's `ancestorRemoved` method. Stop listening for this event by using the event-generating object's `removeAncestorListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the caret position of a text component is updated or changed.

#### **Listening for the event with code**

Listen for this event using an object with the `CaretListener` interface. Register the listener with the event-generating object by using that object's `addCaretListener` method. Handle events as they occur in the listener's `caretUpdate` method. Stop listening for this event by using the event-generating object's `removeCaretListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the cell editor has canceled editing.

#### **Listening for the event with code**

Listen for this event using an object with the `CellEditorListener` interface. Register the listener with the event-generating object by using that object's `addCellEditorListener` method. Handle events as they occur in the listener's `editingCanceled` method. Stop listening for this event by using the event-generating object's `removeCellEditorListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the cell editor is no longer editing.

#### **Listening for the event with code**

Listen for this event using an object with the `CellEditorListener` interface. Register the listener with the event-generating object by using that object's `addCellEditorListener` method. Handle events as they occur in the listener's `editingStopped` method. Stop listening for this event by using the event-generating object's `removeCellEditorListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a state has changed.

#### **Listening for the event with code**

Listen for this event using an object with the `ChangeListener` interface. Register the listener with the event-generating object by using that object's `addChangeListener` method. Handle events as they occur in the listener's `stateChanged` method. Stop listening for this event by using the event-generating object's `removeChangeListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a hypertext link has changed.

#### **Listening for the event with code**

Listen for this event using an object with the `HyperlinkListener` interface. Register the listener with the event-generating object by using that object's `addHyperlinkListener` method. Handle events as they occur in the listener's `hyperlinkUpdate` method. Stop listening for this event by using the event-generating object's `removeHyperlinkListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the popup menu is canceled.

#### **Listening for the event with code**

Listen for this event using an object with the `PopupMenuListener` interface. Register the listener with the event-generating object by using that object's `addPopupMenuListener` method. Handle events as they occur in the listener's `popupMenuCanceled` method. Stop listening for this event by using the event-generating object's `removePopupMenuListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated before the popup menu becomes visible.

#### **Listening for the event with code**

Listen for this event using an object with the `PopupMenuListener` interface. Register the listener with the event-generating object by using that object's `addPopupMenuListener` method. Handle events as they occur in the listener's `popupMenuWillBecomeVisible` method. Stop listening for this event by using the event-generating object's `removePopupMenuListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated before the popup menu becomes invisible. Popup menus may become invisible at any point in time.

#### **Listening for the event with code**

Listen for this event using an object with the `PopupMenuListener` interface. Register the listener with the event-generating object by using that object's `addPopupMenuListener` method. Handle events as they occur in the listener's `popupMenuWillBecomeInvisible` method. Stop listening for this event by using the event-generating object's `removePopupMenuListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a bound property is changed.

#### **Listening for the event with code**

Listen for this event using an object with the `PropertyChangeListener` interface. Register the listener with the event-generating object by using that object's `addPropertyChangeListener` method. Handle events as they occur in the listener's `propertyChange` method. Stop listening for this event by using the event-generating object's `removePropertyChangeListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when a constrained property is changed.

#### **Listening for the event with code**

Listen for this event using an object with the `VetoableChangeListener` interface. Register the listener with the event-generating object by using that object's `addVetoableChangeListener` method. Handle events as they occur in the listener's `vetoableChange` method. Stop listening for this event by using the event-generating object's `removeVetoableChangeListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when an item or node of a `JTree` changes.

#### **Listening for the event with code**

Listen for this event using an object with the `TreeExpansionListener` interface. Register the listener with the event-generating object by using that object's `addTreeExpansionListener` method. Handle events as they occur in the listener's `valueChanged` method. Stop listening for this event by using the event-generating object's `removeTreeExpansionListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when an item or node of a `JTree` changes.

#### **Listening for the event with code**

Listen for this event using an object with the `TreeSelectionListener` interface. Register the listener with the event-generating object by using that object's `addTreeExpansionListener` method. Handle events as they occur in the listener's `valueChanged` method. Stop listening for this event by using the event-generating object's `removeTreeExpansionListener` method.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This event is generated when the value of the selection changes.

#### **Listening for the event with code**

Listen for this event using an object with the `TreeSelectionListener` interface. Register the listener with the event-generating object by using that object's `add TreeSelectionListener` method. Handle events as they occur in the listener's `valueChanged` method. Stop listening for this event by using the event-generating object's `remove TreeSelectionListener` method.



\$ The Graphics ClassA The\_Graphics\_Class+ VCGUIDE:0

\$ Animation and MultithreadingA Animation\_and\_Multithreading+ VCGUIDE:0

# Maintaining\_and\_Porting\_Your\_Java\_Applets\_and\_Applications\$ Maintaining and Porting Your Java Applets and ApplicationsK Maintaining;Porting; Java;Applets;Applications;DebuggingA Maintaining\_and\_Porting\_Your\_Java\_Applets\_and\_Applications+ VCGUIDE:0

# Using\_Visual\_Caf\_to\_create\_Java\_Solutions\$ Using Visual Cafe to create Java SolutionsK Using Visual Cafe to create Java Solutions+ VCGUIDE:0Cafe

## **Two-Way tools**

In Visual Cafe, you have the option to use many of the visual tools available, or write your source code. As you use any of the visual tools, the source code is automatically generated and updated for you as you work.

## **.class files**

Are created when you compile your Java program with the Java compiler. For applications, .class files are interpreted by the Java Virtual Machine on each platform, for applets or by a Java-enabled browser.

## **Parent Language**

When a portion of a programming language is taken from a larger language, the original language is called the parent language.

## **Interpreted Language**

Some languages like C and C++ compile programs into instructions for specific machine processors. Other languages, like Java, compile programs into bytecodes, which are non-processor specific. An interpreter, like the Java Virtual Machine, must run on each platform and interpret the bytecode to run the program on that particular platform.

## **The Internet**

The global network of computers.

**platform**

A particular computer or particular operating system, or a combination of both.

## **Web Page**

A Web page is an HTML document that is transmitted over the World Wide Web (WWW.) It can be any combination of text, graphics, animation, and sound. Web pages often contain links to other web pages. Web pages also run applets.

## **Web Browser**

A Web browser is an application that is used to display web pages for the World Wide Web.

## **tag**

Used in HTML to indicate how specific pieces of text, sound, and graphics are to be formatted and displayed. Tags are also used to indicate to the Web browser that there is an applet embedded within the code.

**attribute**

Extra information that can used by certain HTML tags.

**\$ source code**

The list of machine instructions in a particular programming language to be compiled and interpreted or executed.

## **interface**

What a program user sees and uses to interact and accomplish tasks with a particular computer program. A Java interface is a template for assigning certain specified behaviors to classes.

**overhead**

The total amount of resources from the computer that a program needs or uses during its execution.

## **Dynamic-Link Libraries (DLLs)**

A component of Microsoft Windows programs that contain resource files a program uses as needed rather than having these libraries compiled into the program.

**argument**

Information that is passed to a method. Arguments allow methods to perform their operations on different data making them more versatile.

## **instance**

An component based on a class definition that you use in your programs. An instance is like a variable that represents a class and all it's data and methods. You can have multiple instances of a single class.

## **abstraction**

Removing the unimportant characteristics and details of a component so that only the essential aspects and characteristics remain to describe it.

## **modifier**

Keywords that assign characteristics to various program components like variables, methods, and classes.

## **controls**

Used for accepting and displaying user input and displaying it. Some examples of controls are text boxes, command buttons, and list boxes. Each control has its own set of properties, methods, and events.

## **debugging**

Correcting the logic of your program to get it to run the way you intend.

**syntax**

The important word order in programming source code.

## **UI**

User Interface. The collection of visual components in a window that collects and displays user input.

## **WYSIWYG**

What You See Is What You Get. Originally, this term came out of the development of word processors from cryptic, command line processing of text to creating documents such that the printed (hard copy) looks exactly like what you see on your monitor.

## **variables**

A symbol, or character, used to represent data.

**expression**

A collection of variables and operators that specify a computation.

## **URL**

An address of a resource on the Internet. (Acronym for Uniform Resource Locator).

## **instantiation**

The act of creating a new instance of a class. This is done using the `new` operator.

## **The CODE Attribute Tag**

Tells the browser the name of the .class file to load and run. Using the CODE tag the browser looks for the .class file in the same directory as the HTML file. If the .class file is in a different directory than the HTML file that displays the applet, you need to also include a CODEBASE attribute that specifies the directory where the .class file is located.

## **The WIDTH Attribute Tag**

Specifies the length (or WIDTH if you prefer) of the display area inside the Web page. If the Length parameter is smaller than the applet, the parts of the applet outside the display area will not be visible.

## **The HEIGHT Attribute Tag**

Specifies the height of the display area inside the Web page. If the height parameter is smaller than the applet, the parts of the applet outside the display area will not be visible.

## **The NAME Attribute Tag**

The name of a parameter the Java applet can request.

## The VALUE Attribute Tag

VALUE is the information held by a parameter.

## The `getParameter` Method

The `getParameter` method always returns a text string:

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

`Color` is the name of a parameter specified in an HTML document.

## **ToolTip**

When you move the mouse over a component on the Component Palette and pause for a moment, a ToolTip is displayed. ToolTips show a brief one or two word description of the functionality for the active toolbar button or component. The status bar shows a more detailed description of the tool.

## **data**

The information programs operate on.

## **constructor method**

A method with the same name as the class that contains it. Constructors are used to perform initialization operations when an object is first created.

## **encapsulation**

The concept of collecting the data and methods together into a complete object. Encapsulation collects the properties and behaviors into a single independent module that can be reused in multiple programs without modification.

**reusability**

The ability to use the same source code that defines an object in multiple programs without modification.

**extend**

Creating a new object that can use all the data and methods from an existing object. This is also called inheritance.

See Also

See Also

See Also

See Also

See Also

 See Also

See Also

See Also

 **See Also**

See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



The files needed to create one or more Java programs, such as [applets](#) (including related Web pages containing applets), [applications](#), [beans](#), or [servlets](#), are part of a Visual Cafe *project*. For example, you could place in a project a plane animation applet, a word processing application, or the Web pages and applets that make up an entire Web site.

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 how you want to view your project, as a list of [objects](#) and optionally HTML files, a list of [packages](#), or a list of files in a project. Projects provide vital organization to your programs, particularly as they grow more complex.

Before you can construct an applet or application with Visual Cafe, you need to create a new project, specifying one of the predefined [project templates](#). Then the Project window displays with the project name in the title bar. All Java programs created in Visual Cafe *must* be part of a project.

By default, when you first start Visual Cafe, a new project is automatically created based on the default project template, JFC Applet.

### Looking at the Objects view

If you click the Objects tab in the Project window, you see the [components](#) in your project and, if they were added to the project, the HTML files. Visual Java components are like C++ controls: they are user interface elements, such as windows, menus, buttons, and so on. Some components can contain other components, such as an application window containing a button; these components are called *containers*. In the Project window, 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 (called Visual Cafe [forms](#)), while the components in the containers are Java code within the container Java file.

The Objects view is the view many people use most often during Java program development with Visual Cafe. 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 the standard Java components and the Visual Cafe components to your project is to drag them from the Component Library or Palette to the Project window. Then an instance of the component class, an object, appears. See [Viewing the components and HTML files in a project](#), [Opening editors from the Project window](#), and [Dragging-and-dropping in the Project window](#) for more information.

### Understanding the objects in the project templates

The following components are in the standard Visual Cafe project templates:

- AWT Applet — An Applet component. The Applet container is a type of Panel component that is designed to appear in an HTML file (such as a Web page) viewed in a Web browser (such as Netscape Navigator).
- AWT Application — A Frame, AboutDialog, and QuitDialog component. 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, which is a standard Java component. The About and Quit dialogs are Visual Cafe components that you can customize like templates; they are tied to the About and Exit menu items.
- JFC Applet — A JFC Swing JApplet component.
- JFC Application — A JFC Swing JFrame component, which contains JPanel, JToolBar, and JMenuBar components. Several actions are provided to help you get started. The Open tool and menu item open an Open File dialog box, the Exit menu item opens an Exit dialog box, and the About tool and menu item open an About dialog box.
- Win32 AWT Application (Professional edition) — The same as basic application, except that the project options are set up to produce a native Win32 executable.
- Win32 Console Application (Professional edition) — A simple console application you can use as a starting point. The project options are set up for you.
- Win32 Dynamic Link Library (Professional edition) — A simple DLL you can use as a starting point. The project options are set up for a DLL.
- JavaBean Wizard — A wizard that helps you create a basic bean. The wizard creates two files: *beanclassBeanInfo.java* and *beanclass.java*, where *beanclass* is the name of your bean. See [Using the JavaBean Wizard](#).
- Servlet Project — A wizard that helps you build a servlet. (This feature is not available in the Standard Edition.)
- Empty Project — No objects.

You can also create your own project templates that contain the components, Java files, and project options you commonly use. See [Creating a project template](#).

### Looking at the Packages view

A Java *package* is a group of related classes that can be used by programs that import that package or any file in that package.

It is similar to a C library. The Packages view always shows the standard Java packages that Java programs require.

When components are added to a project, a Default Package, containing the **java** files for those objects, appears. If you add a Visual Cafe component, the Symantec package also displays in the Package 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).

You can also make other packages available to Visual Cafe projects by adding them to the Visual Cafe class path. Then if you add an **import** statement or use part of the package in your Java source code, the package appears in the Packages view. See [Adding packages to Visual Cafe](#).

Usually, only advanced Java programmers are interested in this view. See [Viewing the packages in a project](#) for more information.

### Looking at the Files view

The Files view lists all of the files contained 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 may have added to the project.

In the Imports folder are all of 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. In Visual Cafe Professional edition, you can sort in the Files view by clicking a heading button and turn RAD and the resulting automatic code generation on and off for a file.

As with the Packages view, usually only advanced Java programmers are interested in the Files view. See [Viewing the files in a project](#) for more information.

### Changing tab display

You can remove a tab and change the default tab by right-clicking on a tab and choosing an option from the pop-up menu.

### Constructing a logically arranged project directory

The first time you save a project, the files it contains are stored in the directory you specify. For efficient project management, you should store files in a project in a separate directory per project. Doing so will make deploying your applets, applications, and Web pages much easier.

The Java and Symantec packages are automatically available during development with Visual Cafe. 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.

If your project has a lot of files, you can group the files in subdirectories of the project directory. For example, if your project has many graphics files, you could store these files in a separate subdirectory of the project directory. Remember that when you create your Java programs and HTML files, you specify where they should look for certain files, such as graphics files. So whenever you move files in the project directory, you have to be aware of any dependencies that your Java programs and HTML files have on file locations.

For more information on deployment, see [Deploying applets](#) and [Deploying applications](#).

### Looking at the files in the project directory

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

Extension	Description
<b>.vpj</b>	The Visual Cafe project
<b>.vep</b>	Visual Cafe project options and file list
<b>.ve2</b>	Secondary project information
<b>.cdb</b>	Compiled database that Visual Cafe uses to track compilation dependencies (created after compilation)

These files are not needed for deployment. The **vep** file name displays in the title bar of the Project window; the rest of the project files do not appear in the Project window.



**Note** In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

Visual Cafe can also create these files and store them in the project directory:

Extension	Description
<b>.java</b>	A Java source file, such as for an applet
<b>.class</b>	A compiled version of a Java source file
<b>.obj</b>	An intermediate file produced when you compile a Java source file to create a native Win32 application or DLL (Professional edition)
<b>.lib</b>	A library file used with native Win32 applications and DLLs
<b>.html</b>	An HTML file
<b>.properties</b>	A resource bundle file for localization.

In the Project window, right-click a Java or HTML file and choose an edit item to open the file a Visual Cafe editor. For deployment, you need the compiled version of your Java files (the **class** files) and, for applets, any HTML files you want to use for your Web pages. In addition, you also need your graphics files.

The Java source files and compiled Java files usually 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** files are used for deployment.

### Adding files to a project

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 is not required.

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 directory, then add the **java** file to the Project window. Visual Cafe will attempt to translate the file into its visual environment.

For more information, see [Adding an existing file to a project](#) and [Adding a new file to a project](#).

### Working with multiple projects and subprojects

You can open multiple projects simultaneously; they are displayed in separate Project windows. This helps you to easily navigate between projects.

You can add the **vep** file of a project to another project. The project you added becomes a subproject of the parent project, which means that the subproject is compiled every time the parent project is. See [Adding a subproject](#) for more information.

### Sharing files across multiple projects

Besides imports, you should not share files containing code that was automatically created with Visual Cafe. If you want to reuse files from previous projects, you have a number of options. For example, you can copy the file into the new project directory 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. See [Copying a file in a project](#), [Creating a project template](#), and [Using the Component Library](#).

### Compiling projects

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. See [Running a project](#).

### Defining project options

You can customize your project by setting project options such as the project release type, runtime arguments, compiler settings, and search directories. See [Setting project options](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The files needed to create one or more Java applets, applications, or related Web pages containing applets are part of a Visual Cafe project. For example, in a project you could place a plane animation applet, a word processing application, or the Web pages and applets that make up an entire Web site. All Java programs created in Visual Cafe *must* be part of a Visual Cafe project; otherwise, Visual Cafe cannot compile and run them. A project also helps you manage and organize your files as you develop your Java programs.

1. Choose File  New Project.


The New Project dialog box displays.

2. Select the [project template](#) you want as the base for the new project.

The default template is indicated by an asterisk. You can easily change the default template by selecting a 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. Only one project displays in a Project window.

4. To save the project, choose File  Save As.

The Save As dialog box displays.

The project and all of the files contained within it should be in the same directory. For easier project management, you should save each project in its own directory.

5. Select a directory and 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.

6. Set project options as needed.


Often, the defaults provided with your project template are sufficient.

 See Also

[Details on this step](#)

 See Also

#### Note

- In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.
- The first time you save a project, all of the files it contains are saved to the directory you specify. After you save a project once, saving just the project does not save other files, such as applets. Use File  Save All to save all files in a project.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

#### **To open an existing project from within Visual Cafe**

1. Choose File  Open.

The Open dialog box displays.

2. Navigate to the project directory.
3. Make sure Visual Cafe Project is shown in the Files of type field.

The projects display.

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 the Windows Explorer or another file system window**

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


 See Also

**Note** In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

The first time you save a project, all of the files it contains are saved to the directory you specify. After you save a project once, saving just the project saves the project files only, not other files, such as applets. Use File  Save All to save all files within a project.

For efficient project management, you should store the files in a project in a separate directory per project. Doing so will make deploying your applets, applications, and Web pages much easier. If your project has a lot of files, you can group the files in subdirectories of the project directory. For example, if your project has many graphics files, you could store these files in a separate subdirectory of the project directory. Remember that when you create your Java programs and HTML files, you specify where they should look for certain files, such as graphics files. So whenever you move files in the project directory, you have to be aware of any dependencies that your Java programs and HTML files have on file locations.

#### To save a project the first time

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

The Save As dialog box displays.

2. Select a directory and type the project name in the File name 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.


 See Also

**Note** In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

3. Click Save.


The new file name appears in the title bar.

#### To save project files only

- Activate the Project window, then choose File  Save.

Only project files are saved, not files in a project, such as applets.

#### To save all files in a project

- Activate the Project window, then choose File  Save All.

Project files and files in the project are saved.

#### To save one file in a project

After you save a project once, you can save a file within the project separately from other files.

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

 See Also


[Details on this step](#)

Remember that a top-level [container](#) in the Objects view is associated with a Java source file.

2. Choose File  Save.

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

#### To save and rename a project

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

The Save As dialog box displays.

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.



**Note** In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

3. Using operating system tools, delete from the project directory the project files that have the old name and the extensions **vep**, **vpj**, **ve2**, and **cdb** (if present).

#### **To copy a project**

1. If Visual Cafe is running, make sure that the Project window for the project is closed.  
You cannot copy files while they are in use.
2. From the Windows operating system, copy the files in the project and paste them in a new directory, preserving the directory structure.  
If all of your files are in one project directory, you could just copy the project directory, then rename it.

#### **To delete a project**

1. If Visual Cafe is running, make sure that the Project window for the project is closed.  
You cannot delete files while they are in use.
2. From the Windows operating system, delete the project directory and all files in the project (except for imports).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe version 2.0 project files (the **vep**, **vpj**, **ve2**, and **cdb** files) are not backward-compatible with Visual Cafe version 1.0 project files. If you save a 1.0 project in the Visual Cafe 2.0 software, you cannot convert the project back to 1.0. So you might want to copy your 1.0 project directories before opening them in the 2.0 software. That way you will still have your 1.0 project files.

When you open a 1.0 project in Visual Cafe version 2.0, 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 1.0 Java file without first opening its 1.0 project, the Java file will not be set up to use the new Visual Cafe components. You should first open the 1.0 project in the newer version of Visual Cafe so that the file is updated for you.

If you want to automatically convert a Java source file from the 1.0 event model to the 1.1 event model, see [Migrating a Java file from the 1.0 to 1.1 event model](#).

 See Also

#### 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.
- In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe lets you add subprojects to a 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.

 See Also

**Note** There is currently no particular order in which subprojects are compiled.

1. Open the project you want to add subprojects to.

 See Also

[Details on this step](#)

2. Add the **vep** file of a project to the parent project.

 See Also

[Details on this step](#)

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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

#### **To close the active project**

- Choose File  Close Project while the Project window is active.

You are 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 File  Exit.

You are prompted to save any unsaved changes. The project and Visual Cafe close.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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 copying or moving components, if a box appears over a [container](#) in the Project window, the component is inserted within the container. If a line appears under a component, the new component appears after the component that is underlined. A **+** 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. While dragging, a circle with a line through it means the operation is not allowed.

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. Or press CONTROL 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 <a href="#">form</a> (the z-order), the tab order, and the order they are declared in the Java code.
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 CONTROL 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 CONTROL 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 directory. See <a href="#">Sharing files between projects</a> .)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Project window helps you to quickly edit the items in a project.

To access other editor windows from the Project window:

<b>Double-click...</b>	<b>To get this editor...</b>
top-level <a href="#">component</a>	Form Designer
menu	Menu Designer
Java or HTML file	Source window

Components and menus are displayed in the Objects view.

Java files are displayed in the Packages and Files view.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can remove a tab and change the default tab by right-clicking on a tab and choosing an option from the pop-up menu.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Objects view of the Project window shows the [components](#) in your project and, if they were added to the project, the HTML files. 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. The Objects view is the view many people use most often during Java program development with Visual Cafe.

#### **To look at the Objects view**

- Click the Objects tab in the Project window.

Following are some tasks you can perform with components in the Objects view:

[Adding a component to a project](#)

[Copying a component in a project](#)

[Renaming a component in a project](#)

[Deleting a component from a project](#)

[Dragging-and-dropping in the Project window](#)

[Creating a component interaction](#)

Following are some tasks you can perform with files in the Objects view:

[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 between projects](#)

[Running a project](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Packages view lists the Java source files, grouped as Java [packages](#), in your project. The Packages view always shows the standard Java packages that Java programs require. When components are added to a project, a Default Package, containing the **java** files for those objects, appears. If you add a Visual Cafe component, the Symantec package also displays in the Package view; it contains the Java source file for each Visual Cafe component you include.

You can also make other packages available to Visual Cafe projects by adding them to the Visual Cafe class path. See [Adding packages to Visual Cafe](#).

Usually, only advanced Java programmers are interested in the Packages view.

#### To look at the Packages view

- Click the Packages tab in the Project window.

You can expand and collapse a package like you would for a Windows file system. Press Shift then + or - to expand all or collapse all entries in the tree view.



**Note** Classes that have not been added to a package display under the Default Package. Only source files that have been added to the project are listed.

Following are some tasks you can perform with files and packages in the Packages view:

[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 between projects](#)

[Adding packages to Visual Cafe](#)

[Using the Class Wizard](#)

[Running a project](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Files view lists all of the files contained in your project. At the top level are the files for the components in your project, as well as any HTML files you may have added to the project.

In the Imports folder are all of the Java source files for the packages your project uses. Whereas in the Packages view the files are listed by package (see [Viewing the packages in a project](#)), in the Imports folder the files are listed alphabetically.

In Visual Cafe Professional edition, you can sort files in the Files view by clicking a heading button and turn and the resulting automatic code generation on and off for a file.

HTML files are also listed in the Objects view (see [Viewing the components and HTML files in a project](#)).

Usually, only advanced Java programmers are interested in the Files and Packages view.

#### To look at the Files view

- Click the Files tab in the Project window.



**Note** If a file with the extension **java** appears, the corresponding **class** file is not listed, because you are interested in **java** files during development.

Following are some tasks you can perform with files in the Project window:

[Adding a new file to a project](#)

[Adding an existing file to a project](#)

[Sorting files in the Files view](#)

[Deleting a file from a project](#)

[Copying a file in a project](#)

[Sharing files between projects](#)

[Running a project](#)

[Enabling and disabling RAD and automatic code generation](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

Visual Cafe provides several ways to add components to your project. When components are added directly to a [form](#) in the Form Designer, they are also added to the associated project.

#### To add a component by using drag-and-drop

You can drag a component from the following areas into the Project window, Form Designer, or Menu Designer:

- the Component Palette or Library
- a Project window or Form Designer of the same project (press CONTROL and drag to copy the component)
- a Project window or Form Designer of a different project

 See Also

**Note** When you copy a component within the same or a different project, only component code that is automatically generated by Visual Cafe is copied. This does not include [custom code](#) and [interactions](#).

#### To add a component by using an Insert menu item

- While the Project window or Form Designer is active, choose a menu item from the Insert menu to add components to your project.
- Right-click the Project window and choose an Insert menu item.

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

 See Also

**Note** If the parse fails, you can see a file in the Packages and Files view, but not see 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 code to a Java source file](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also



You can copy and paste a component listed in the Objects view of the Project window from one project to another, or within the same project:

- 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, the Java code of that component is placed 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](#).

You can copy components between the Project window (Objects tab), Form Designer, and Menu Designer as needed.

Visual Cafe only allows appropriate copies; for example, a component that is not a container cannot be copied to the top level. While dragging, a circle with a line through it means the operation is not allowed.

#### 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 that you want to copy.
3. Choose Edit  Copy, right-click and choose Copy, or click the toolbar Copy button.
4. If you want to paste the component within another container, select that container.
5. While the target Project window (Object tab), Form Designer, or Menu Designer is active, choose Edit  Paste, right-click and choose Paste, or click the toolbar Paste button.

The component appears in the Project window. If needed, the component is renamed to prevent name conflicts.

#### To copy and drag a component

- Within the same project, press CONTROL and drag the component to the location you want it.
- To a different project, drag the component to the location you want it.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can rename the components listed in the Objects view of a project. 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 name in the entire Java source file, even in [custom code](#).

**To rename a component from the Project window**

- Click the Objects tab in the Project window, select the component name, press TAB or F2, and retype the name in the edit frame. Press ENTER or click somewhere else when you are finished.
- Click the Objects tab in the Project window, slow double-click the component name and type the new name in the edit frame. Press ENTER or click somewhere else when you are finished.

**To rename a component from the Property List**

- Make your project active, then change the Name property of the component in the Property List.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you delete a top-level container in the Objects view of the Project window, the corresponding **java** file is deleted from the project and the other views. The file is not deleted from the directory on your hard disk.

 See Also

**Note** If you want to delete a file, you should first delete it from the project then delete it using Windows operating system commands. Do not delete a file using operating system commands before first deleting it from the project.

If you delete a component within a container, the code automatically generated by Visual Cafe is deleted. You must manually delete any [custom code](#).

If you delete a component's code from a Java file, the component is removed from the Project window after you click out of the Source window or save the Java file.

 See Also

**Note** To avoid deleting the wrong code or not all of the code, it is recommended that you delete components from the Project window instead of from the source code. Then delete from the source code any custom code.

#### **To delete components from the Project window (Objects tab), Form Designer, or Menu Designer**

- Select the component and press DELETE or choose Edit  Delete.

If you delete a component from a container, you must manually delete any custom code involving that component.

If the component is a top-level container, the corresponding Java file is not deleted from the directory on the hard disk. You must manually delete it.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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 add these files to your project.

1. Choose File  New File.

An empty Source window displays.

2. Choose File  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. Optionally select or deselect Enable RAD.

For more information, see [Enabling and disabling RAD and automatic code generation](#).

5. Click Save.

The file name appears in the title bar.

6. In the Source window, add appropriate code or text.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can add a variety of files to a project, including HTML, Java, properties, and **vep** project files to a project.

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 is not required.

You can also add Java source files (with the extension **java**). 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 directory, then add the **java** file to the Project window. Visual Cafe will attempt to translate the file into its visual environment.

 See Also

**Note** If you add a Visual Cafe version 1.0 Java file without first opening its 1.0 project, the Java file will not be set up to use the new Visual Cafe components (specifically, the INIT portion of the code). You should first open the 1.0 project in the newer version of Visual Cafe so that the file is updated for you.

If you add the **vep** file of a project to a project, that project becomes a subproject of the project it resides in.

Remember that it is recommended that you store the files in a project in a separate directory per project. Doing so will make deploying your applets, applications, and Web pages much easier. You should also avoid sharing files between projects. See [Sharing files between projects](#) for more information.

 See Also

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

#### To add files from the Visual Cafe environment

1. Choose Insert  Files into Project.

The Project Files dialog box displays.

2. Select the file(s) that you want to add.
3. Click Add or Add All, as appropriate.
4. Click OK.


The file(s) appear in the Project window.

#### To add files from the Windows Explorer or other file system window

- Drag Java or HTML files from the Explorer into the Project window, Files view.

#### To add files from the Find in Files dialog box

You can easily add a large subdirectory to your project using this technique.

1. Choose Search  Find in Files.
2. Specify a directory.
3. Click Find.
4. Right-click on the window with the list of files. Then choose Add All to Project.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In Visual Cafe Professional edition, you can sort files in the Files view of the Project window by clicking a heading button.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




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

**To remove a file by pressing DELETE**

- In the Files view of the Project window, select a file and press DELETE.

**To remove files by using the Project Files dialog box**

1. Make the Project window active.
2. In the Packages or Files view, right-click the window to display the pop-up menu, then choose Insert/Remove Files.

Or while any view is displayed, choose Insert  Files into Project.

3. In the Project Files dialog box, select one or more files from the list at the bottom pane of the dialog box.
4. Click Remove.
5. Click OK.



**Note**



- When you remove a Java source file from a project, all associated visual elements that are created in the Java code are also removed.
- Deleting a file from a project does not delete it from the project directory. You must manually delete it.
- You cannot remove imported files (through the import statement).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can copy and paste a file from one project to another, or within the same project. The file is duplicated and placed in the target project directory.

1. Open the project(s) you want to use and click the Objects tab.
2. In the Project window, select the file that you want to copy.
3. Choose Edit  Copy or click the toolbar Copy button.
4. Activate the Project window that you are copying the file to.
5. Choose Edit  Paste or click the toolbar Paste button.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

For efficient project management, it is recommended that you store the files in a project in a separate directory per project. Doing so will make deploying your applets, applications, and Web pages much easier.

You can reuse a file if it does not have code generated by Visual Cafe. Just add the file to the other project. For more information, see [Adding an existing file to a project](#).

Any file that contains code that was automatically generated by Visual Cafe should not be shared between 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 directory 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](#), [Creating a project template](#), and [Using the Component Library](#).

 See Also

**Note** File names are relative to the current project. For example, if you add a file called foo.java and it resides in a subdirectory (called foo1) of the project directory, the file name is stored as foo1\foo.java and is not fully qualified. If foo1 was a directory at the same level as the project directory, the file would be stored as ../foo1\foo.java.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The standard Java packages and the Symantec Visual Cafe packages are available to you as you create Java programs in the Visual Cafe environment. You can also make other packages available to Visual Cafe projects by adding them to the Visual Cafe class path. 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. See [Specifying file search paths and the output directory](#) for more information.

Remember to keep the directory structure of your package intact, and make sure that your file names use the same uppercase and lowercase letters, exactly as they were before copying the package.

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.

## Project Options dialog

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

In the Project Options dialog box, Visual Cafe lets you perform the following tasks to set options that apply to a single project:


[Specifying whether builds are debug or final](#)  
[Setting the project type to applet or application](#)  
[Specifying what applets to run and the HTML file](#)  
[Making applets run in the Applet Viewer or a browser](#)  
[Specifying the main class to run for an application](#)  
[Specifying the name of a native application or DLL](#)  
[Setting the working directory for a native program](#)  
[Specifying a program for running and debugging a DLL](#)  
[Specifying arguments for application execution](#)  
[Specifying execution settings for a servlet](#)  
[Specifying whether to parse imports](#)  
[Enabling and disabling RAD and automatic code generation](#)  
[Specifying whether to clear messages before builds](#)  
[Localizing auto-generated code through your project options](#)  
[Specifying whether to update project beans on project open](#)  
[Setting compiler options for a project](#)  
[Producing Javadoc for a project](#)  
[Specifying the exports of a DLL or native application](#)  
[Specifying libraries to link into your native program](#)  
[Specifying file search paths and the output directory](#)  
[Enabling a version control system for a project](#)  
[Setting debugger options for a project](#)

 See Also

#### Note

- Most changes made in this dialog box do not take effect until the next time they are needed. For example, if you change runtime arguments, they do not take effect until the next time you run your Java application.
- To set global options that affect all projects, use the Environment Options dialog box.

#### To set project options

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

#### To set environment options

- Choose Tools  Environment Options.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



From the Project tab of the Project Options dialog box, you can perform the following tasks:

[Specifying whether builds are debug or final](#)

[Setting the project type to applet, application, or DLL](#)

[Specifying what applets to run and the HTML file](#)

[Making applets run in the Applet Viewer or a browser](#)

[Specifying the main class to run for an application](#)

[Specifying the name of a native application or DLL](#)

[Setting the working directory for a native program](#)

[Specifying a program for running and debugging a DLL](#)

[Specifying arguments for application execution](#)

[Specifying whether to parse imports](#)

[Enabling and disabling RAD and automatic code generation](#)

[Specifying whether to clear messages before builds](#)

[Localizing auto-generated code through your project options](#)

[Specifying whether to update project beans on project open](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe has Debug and Final project option sets that let you specify how you want to compile your Java code. By default, for Debug the compiler includes debug information, while for Final no debug information is included and Java optimizations for speed and compactness are performed. Debug information enables you to use all Visual Cafe debug 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 for a project](#) and [Specifying file search paths and output directories](#).

#### To set the release type

1. Activate the Project window of the project you want to work with.
2. Choose Project Options.
3. In the Project Options dialog box, click the Project tab.
4. Select one of the following options:

Select...	To do this...
Debug	By default, build an executable that contains debugging information.
Final	By default, build a more compact executable that is optimized and contains no debugging information.

You can change the default, as described in the next procedure.

5. Click OK.

The change takes effect the next time you compile your Java program.

#### To change a release type option set


1. From the Project Options dialog box, set the release type to Debug or Final.
2. Click the Compiler and Directories tabs and set the options for that release type.
3. Click OK.

The change takes effect the next time you compile your Java program.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Project tab.
4. Select one of the following options:

Select...	To specify this...
Applet	<p>The project is an applet.</p> <p>When you run a project, the <b>Start with Web page</b> setting determines which HTML file is used. The HTML file determines which applets are run. See <a href="#">Specifying what applets to run and the HTML file</a>.</p> <p>To specify that your applets should run in your default Web browser, select the <b>Execute applet in default Web browser</b> option. Deselect it if you want to run applets in the Applet Viewer associated with the Visual Cafe environment. For more information, see <a href="#">Making applets run in the Applet Viewer or a browser</a>.</p>
Application	<p>The project is a standalone application.</p> <p>To run the application from Visual Cafe, the main class must also be specified. See <a href="#">Specifying the main class to run for an application</a>.</p>
Servlet (Professional Edition)	<p>The project contains a servlet, most likely created using the Servlet Wizard.</p> <p>To run the servlet from Visual Cafe, you can specify various parameters. See <a href="#">Specifying execution settings for a servlet</a> for more information.</p>
Win32 Application (Professional edition)	<p>The project is a native, standalone executable.</p> <p>To run the application from Visual Cafe, the main class must also be specified. See <a href="#">Specifying the main class to run for an application</a>.</p> <p>You can also set the application name, which is by default the project name appended with the <b>exe</b> extension. See <a href="#">Specifying the name of a native application or DLL</a>.</p> <p>If you want to run your executable from a different directory than where it is located, see <a href="#">Setting the working directory for a native program</a>.</p>
Win32 DLL (Professional edition)	<p>The project is a native Dynamic Link Library (DLL).</p> <p>You can specify the program to use to run and debug the DLL. See <a href="#">Specifying a program for running and debugging a DLL</a>.</p> <p>You can also set the library name, which is by default the project name appended with the</p>

appropriate extension. See [Specifying the name of a native application or DLL](#).

If you want to run the calling program from a different directory than where it is located, see [Setting the working directory for a native program](#).

5. Click OK.

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


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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 Applet Viewer, 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 Applet Viewer, 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.

1. Activate the Project window of the project you want to work with.
2. Choose Project  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 added to your project automatically appear in the pop-up menu.
  - Click ... to browse 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>.
5. Click OK.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

When running applets from Visual Cafe, you can launch your applets in the Applet Viewer associated with the Visual Cafe environment or in the Web browser of your choice. The Web browser must be set up to be the default Web browser for your computer.

Running applets in the Applet Viewer can be faster, because the browser does not have to start up. However, it is a good idea to test your applets in popular browsers before deployment.


 See Also

**Note** When you run an applet in the debugger, it runs in the Applet Viewer.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Project tab.
4. Select Applet, if needed, then select Execute applet in default HTML viewer if you want to run the applet in a browser or deselect it if you want to run the applet in the Applet Viewer.
5. Click OK.

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

 See Also

**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. On Windows NT 3.x, you set file associations in File Manager  File

 See Also

Associate; on Windows 95/98 and Windows NT 4.0 and higher, from a Windows file system window (such as the Explorer), choose View

 See Also

Options (or Folder Options)

 See Also

File Types tab to set the association, including the **open** action. A browser might have already set the association for you, for example, the file type Netscape Hypertext Document.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

To run an application from within the Visual Cafe environment, you must specify the starting point of your application (the Java class file containing the **main** method) so Visual Cafe knows how to run your application. If you started a project with an 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 base project name or, for a native Win32 application, the base executable name (specified in the application name field).

If your application also accepts arguments on the command line (the **main** method takes arguments), you need to specify those. See [Specifying arguments for running an application](#).

There are a few differences in the significance of a the **main** method in a bytecode and native Java application. See [The main class in bytecode and native applications](#) for more information. You can create a native Win32 application or DLL only if you have Visual Cafe Professional edition.

1. Activate the Project window of the project you want to work with.
2. Choose Project  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. Click OK.

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


 See Also

**Note** If you rename a class that appears in this field, Visual Cafe updates the field for you.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe Professional edition lets you create native Win32 applications and DLLs. You can specify the name of the compiled executable or library.


1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
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 name in the Application or Library 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.
5. Click OK.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe Professional edition lets you create native Win32 applications and DLLs. 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 where you want to run it from.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
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 where you want to run the executable from.
5. Click OK.


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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The Visual Cafe Professional edition lets you create and debug DLLs. You need to specify a Java program from which to run and debug the DLL.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Project tab.
4. While the Project Type is Win32 Dynamic Link Library, type the name of the program in the Calling program field.
5. Click OK.

The change takes effect the next time you debug the DLL.

 See Also

#### Note


- In the Calling program field, you can specify the name and fully qualified path to the executable. If you do not specify a fully qualified name, Visual Cafe looks in the project directory, then through the directories in your Windows PATH environment variable.
- When the program tries to load the DLL, it looks for the DLL as follows:
  1. The directory from which the application loaded. This is the project directory or, if you specified a full path, the directory where the program resides.
  2. The current working directory, if different from the directory from which the application loaded.
  3. For Windows 95, 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



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](#).

1. Activate the Project window of the project you want to work with.
2. Choose Project  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. Click OK.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe lets you run and debug a [servlet](#) from its environment. For more information on creating a servlet project, see [Using the Servlet Wizard](#). (This feature is not available in the Standard Edition.)

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
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	<p>Specify an HTML file in one of these ways:</p> <ul style="list-style-type: none"><li>• 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.</li><li>• 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.</li><li>• Click ... to browse for an HTML file.</li><li>• 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.</li></ul>
Servlet class	The name of the class file.
ServletRunner arguments	<p>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> <p>-p <i>port</i> — the port number to listen on</p> <p>-b <i>backlog</i> — the listen backlog</p> <p>-m <i>max</i> — maximum number of connection handlers</p> <p>-t <i>timeout</i> — connection timeout in</p>

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 you run your project.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe ships with the standard Java package imports from the Sun Microsystems Java Developers 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.


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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


Visual Cafe Professional edition lets you turn off the visual environment for a file, and the automatic code generation that occurs by translating the visual environment into source code. You can specify in your project options that new files you add to the project — by choosing File  New File, Insert

 See Also

Files into Project, or dragging files from a file system window into the Project window — have the Rapid Application Development (RAD) features 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, you can design your [forms](#) in the [Form Designer](#), and Menu Designer, and you can use the Interaction Wizard to specify [interactions](#). You might want to turn off RAD if, for example, you do not want Visual Cafe to automatically generate code for you, and 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. Remember, if RAD is off and you turn it on, Visual Cafe might change your code 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. Choose Project  Options.
3. In the Project Options dialog box, click the Project tab.
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 for a file from the Save As dialog box


The Save As dialog box can appear when you save a file displayed in the Source window.

- Select or deselect Enable RAD.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




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, if 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 set 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](#).

You can create a native Win32 application or DLL only if you have Visual Cafe Professional edition.

#### To access the Compiler view

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Compiler tab.

#### To specify general compiler options

1. From the Compiler view, choose Compiler Category  General.
2. Select the options you want:

Select...	To specify this...
Use Java optimizations	Optimize the Java executable for a more compact executable that runs faster. (Default: enabled for Final and disabled for Debug release types)
	Select the Disable function inlining option if needed. 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. (Default: not selected)
Generate debug information	Create debugging information used by the Visual Cafe debugger. For example, this option lets you see local variables during debugging. (Default: enabled for Debug and disabled for Final release types)
Use the Sun Java compiler	Use the Sun Java compiler, javac.exe. When this option is cleared, the Symantec Java compiler, which is faster, is used. Note that you cannot compile native Win32 applications and DLLs with the Sun Java compiler. (Default: disabled)
Show compiler warnings	Display compiler warnings in the Messages window. During development, you may want to look at items identified by warnings. (Default: enabled)
Show progress messages	Display compiler progress messages in the Messages window. (Default: disabled)
Show dependencies	Display file dependencies, such as imports, in the Messages window. (Default: disabled)
Show all Java messages	When using the Sun Java compiler, this option causes the compiler to report diagnostic messages about its own execution. This option is ignored by the

	Symantec compiler. (Default: disabled)
Warn on deprecated	If selected, when the compiler encounters code that uses deprecated classes or members, it issues a warning. If this option is not selected, the warnings are disabled, so the compiler is faster and uses less memory. (Default: enabled)

3. Click OK.

The change takes effect the next time you compile your project.


### To specify exports (native Win32 only)

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.

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 an image, the faster it can be loaded.

Follow these steps to specify exports:


1. From the Compiler view, choose Compiler Category  Exports.  
A list the same as the Packages view of the Project window appears.
2. Select the packages and classes you want to export.  
A grayed, selected box for a package means that not all classes are selected for that package.
3. Click OK.  
The change takes effect the next time you compile your project.

### To specify libraries to link into your source (native Win32 only)

The Libraries option lets you 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 when you compile your native application or DLL. Visual Cafe can automatically locate libraries for you if they are in your library search path.

1. From the Compiler view, choose Compiler Category  Libraries.
2. 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 Delete.
3. Click OK.  
The change takes effect the next time you compile your project.

### To specify advanced compiler options

1. From the Compiler view, choose Compiler Category  Advanced.
2. Specify the options you want:

Option	Description
Make Settings	Specify how imports are handled when the project is compiled. (Default: Don't check dependencies on imports)
	Generate classes for out-of-date imports means that Visual Cafe searches for the Java source of imports and updates the classes to the most current version, if needed. This was the default for Visual Cafe

version 1.0.

Warn on out-of-date imports means you get a warning message if a class is out-of-date, but Visual Cafe does not regenerate the class.

Don't check dependencies on imports means Visual Cafe does not check imports to determine if they are out-of-date.

GUI application  
(Professional edition)

Suppresses the console window. (Default: disabled)

Use performance profiling  
(Professional edition)

Attributes the code with profiling calls so the executable can generate profiling information. (Default: not selected)

P6 Pentium code generation  
(Professional edition)

Generate P6 Pentium code. (Default: generate P5 Pentium code)

Custom compiler flags

Pass command line options to the compiler. For information on the **sj** compiler options, see [Compiling from the command line](#).

3. Click OK.

The change takes effect the next time you compile your project.

#### **To specify Javadoc Options**

For information on Javadoc fields, see [Producing Javadoc for a project](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

From the Directories tab of the Project Options dialog box, you can add and remove directories that Visual Cafe uses to locate source and class files and control where Visual Cafe puts compiler output files. This view changes the option set for the Debug or Final release type. See [Specifying whether builds are debug or final](#) for more information.

You can also set class path information for the entire Visual Cafe environment. Remember that Visual Cafe does not inherit Windows class path information unless it is set up to do so.

You can create a native Win32 application or DLL only if you have Visual Cafe Professional edition.

#### To access the Directories view

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Directories tab.

#### To specify source file search paths

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

 See Also

**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.

1. Go to the Directories view of the Project Options dialog box.
2. Choose Source Files in the Show directories field.

The directory order affects the search order: the topmost directory is the first to be searched. The default setting, the project directory, does not appear in the list; it is the first directory in the search order.
3. Modify the list as needed:
  - To change the order that 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 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.
4. Click OK.

The change takes effect immediately.

#### To specify the output directory for a project

You can specify where compiler output files, such as **class** files, are stored for this project.

 See Also

**Note** When you run a project in the Applet Viewer, the output directory is temporarily added to the class path.

1. Go to the Directories view of the Project Options dialog box.
2. Choose Output Files in the Show directories field.

The output directory displays. If no output directory is specified, Visual Cafe uses the default: it recreates the package structure under the project directory. If you do specify an output directory, all output files are placed in this directory and any package file hierarchy structures are created as well.

For information on Javadoc fields, see [Producing Javadoc for a project](#).
3. Type a directory and full path in the field, or select a directory by clicking the ... browse button that displays in the field.
4. Click OK.

The change takes effect the next time you compile your project.

### To specify library file search paths (native Win32 only)

If a native application or DLL uses libraries, you need to make sure Visual Cafe can find them. To specify locations to look for libraries, follow these steps:

1. Go to the Directories view of the Project Options dialog box.
2. Choose Native Library Files in the Show directories field.

A list of directories that can contain library files displays. The order affects the search order: the topmost directory is the first to be searched. The default is to search directories specified for the Windows LIB environment variable; these directories do not appear in the list and are last in the search order.

3. Modify the list as needed:
  - To change the order that 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 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.
4. Click OK.

The change takes effect the next time you compile.

### To specify class 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 are using. You can make up your own custom list of directories, let Visual Cafe 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](#)), and use the class path set for the Visual Cafe environment. If you specify one or more of these options, the search order is your list, the automatically generated list, then the class path for the environment.

1. Go to the Directories view of the Project Options dialog box.
2. Choose Input Class Files in the Show Directories field.

A list of directories, and JAR and ZIP files displays. The directory order affects the search order: the topmost entry is the first to be searched.

3. Modify the list as needed:
  - To change the order that 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 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.
4. To generate the class path based on the files in the project, select Autogenerate class path. Otherwise, deselect it.

The default is selected. When this option is selected, if a file is not in the project directory, the file path is added to the class path.
5. 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.

The default is selected.
4. Click OK.

The change takes effect immediately.

### To set the class path for the Visual Cafe environment

Visual Cafe gets its class path information from this file: Visual Cafe Bin\sc.ini. You can modify the sc.ini file to set class path information globally for your Visual Cafe environment. You should append your information to the **classpath** statement, not delete the default **classpath** statement, because it can affect the operation of Visual Cafe. See [Setting environment variables in the sc.ini file](#) for more information.

### To have the Visual Cafe environment inherit 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,

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 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, 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 3.51, you can set the **classpath** variable by opening Control Panel then System and entering a value for the **classpath** so it appears in the System Environment Variables list box.

For Windows NT 4.0 and higher, you can set the **classpath** variable by opening Control Panel  System

 Environment tab, and entering a value so it appears in the System or User Variables list box.

To inherit the Windows class path setting:

1. In the Environment section of Visual Cafe \Bin\sc.ini, add the following specification at the end of the **classpath** statement:  
**;%classpath%**
2. Restart Visual Cafe for the change to take effect.

#### To set the class path for a Web browser

Before deployment, 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**

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



If you want to set Visual Cafe environment variables (which do not affect the setting for your entire computer), you can edit the [Environment] section of Visual Cafe \Bin\sc.ini. The statement syntax is similar to that for autoexec.bat, except you omit the **set** keyword. Any changes take effect the next time you start Visual Cafe.

### 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 you have:

```
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:

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

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



From the Debugger tab of the Project Options dialog box, you can do the following:

- From General, you can enable debugging in Internet Explorer or Netscape Navigator. For more information, see [Debugging code in a Web browser](#). (This feature is not available in the Standard Edition.)
- From Exceptions, you can view and customize how the debugger works with [exceptions](#) for this project. For more information, see [Defining how exceptions are handled by the debugger](#).

If you have Visual Cafe Professional edition, you can create and debug DLLs. During debugging, you need to specify a Java program from which to debug the DLL. See [Specifying a program for running and debugging a DLL](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe Professional Edition allows you to debug your code in Netscape Navigator or Microsoft Internet Explorer instead of the Applet Viewer.

Debugging in a browser allows you to test your applet in the environment where the applet will be used. You will be able to refine your code to reflect differences in the virtual machines of the supported browsers. Visual Cafe currently supports supports Netscape Communicator 4.05, Netscape Communicator 4.04 (with the JDK 1.1 Patch), and Microsoft Internet Explorer 4.x.

#### To turn on debugging in a Web browser

1. Activate the Project window of the project you want to work with.
2. Choose Project Options.
3. In the Project Options dialog box, click the Debugger tab.
4. Choose General in the Category field.
5. Select either Netscape Navigator or Microsoft Internet Explorer/Jview.  
The default is to debug using Sun Java VM (the Visual Cafe Applet Viewer).
6. Choose Project Run in Debugger or Project



Step Into to start your debugging session in the selected web browser.




#### Notes

- Netscape can only debug applets: if you change the project type to java application, then you will notice that the debug using option will be set back to Sun Java VM and the Netscape Navigator option is unavailable. Internet Explorer supports both applets and applications.
- If Visual Cafe cannot detect the supported version of either web browser then 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, then for some reason you remove Netscape Navigator or Microsoft Internet Explorer and try to debug your project, you will get 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 and Internet Explorer: Incremental debugging, Expression evaluation, and Remote debugging.
- The steps below must all be done to allow debugging in Netscape Navigator.
  1. Install Communicator 4.04
  2. Install the Netscape JDK 1.1 Support Patch for 4.03 or greater
  3. Change to your current Communicator "user" directory, and edit your "prefs.js" file to allow local applet access to the hard disk in ...\\netscape\\user\\(your username)\\prefs.js add: user\_pref("signed.applets.low\_security\_for\_local\_classes", true);

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can view and customize how the debugger works with [exceptions](#) for a project. You can even add exceptions while your Java program is executing and they take effect immediately.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Debugger tab.
4. Choose Exceptions in the Category field.
5. Modify the exceptions list as needed:
  - To specify that the debugger stop only if a particular exception is not handled in the code, deselect the checkbox for the exception.
  - To stop the debugger whenever a particular exception occurs, select the checkbox for an exception. The program will stop whenever the exception is encountered, regardless of whether the program handles the error or not. For example, you could use this feature to see if your code is getting an exception that is being handled by an inherited exception handler you do not know about.
  - To add a new exception to the list, click Add, then enter the exception in the blank field that appears.
  - To delete an exception from the list, select an exception to highlight it, then click Delete.
  - To restore the default exception settings, click Restore Defaults.
6. Click OK.

The change takes effect the next time the Java program is executing in the debugger; if it is already executing in the debugger, the change takes effect immediately.

## Project Templates

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

When you create a new project, you specify a project template as the starting point. Visual Cafe provides these project templates:

Template	Description
AWT Applet	A project that contains a single Applet.
AWT Application	A project containing a Frame (with an Open File dialog and menu bar), an About dialog (with a label and Done button), and a Quit dialog (with a Yes and a No button).
JFC Applet	A JFC Swing JApplet component.
JFC Application	A JFC Swing JFrame component, which contains JPanel, JToolBar, and JMenuBar components. Several actions are provided to help you get started. The Open tool and menu item open an Open File dialog box, the Exit menu item opens an Exit dialog box, and the About tool and menu item open an About dialog box.
Win32 AWT Application (Professional edition)	The same as basic application, except that the project options are set up to produce a native Win32 executable.
Win32 Console Application (Professional edition)	A simple console application you can use as a starting point. The project options are set up for you.
Win32 Dynamic Link Library (Professional edition)	A simple DLL you can use as a starting point. The project options are set up for a DLL.
Bean Wizard	A wizard that helps you create a basic bean. The wizard creates two files; <i>beanclassBeanInfo.java</i> and <i>beanclass.java</i> , where <i>beanclass</i> is the name of your bean. See Using the Bean Wizard.
Servlet	Wizard that helps you create a servlet.
Empty project	A project with no components.

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 is 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](#) for more information.

If you do not need to create an entire custom project, but you would like to create a custom component, see [Using the Component Library](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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

1. Choose File  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 the default is changed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


Project templates are used as the basis of new projects. When you create a project from a template, the project initially contains the files and components in the template. You can create your own templates as the starting point of new projects.

 See Also

**IMPORTANT** When you save a project as a template, make sure all of your source files (both **java** and **html**) are in the same project directory or a subdirectory of the project directory.

1. Create a project to be used as the template.

Add components, such as forms, dialogs, and menus, that you want to reuse.

2. With the current project in a Project window, choose Project  Create Project Template.

The Create Project Template dialog box displays.

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 displays when the template is listed in the Component Library window.


6. Click OK.

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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

1. To open the Component Library, choose View  Component Library.
2. Display the contents of the Project Templates folder.
3. Select the template from the template list.
4. Press the DELETE key.

## Obsolete Project Topics

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The primary component you work with in Visual Cafe is a project. Visual Cafe uses projects to store all the source files that make up your programs. Typical projects include a set of [components](#) including [forms](#) (such as windows and dialog boxes) and applets. Projects provide vital organization to your programs as they grow more complex. Visual Cafe cannot create a program outside the project environment; meaning you cannot just compile and run a single source file with Visual Cafe. Every Java program you create with Visual Cafe must start out as a project. It is good practice, for organizational reasons, to create a separate directory for each new project you create.

You work with projects in the Project window which allows you to open forms in the Form Designer, open source code and manually create interactions in the [Source window](#). In the Project window, you can also add and remove files from a project. The Project window lists all forms, components, and classes in a project. Visual Cafe lets you choose how you want to view your project, either as a list of visual components or as a list of source files grouped in packages.

In Visual Cafe, you can also create projects that have no visual elements but are just made up of source code, for example, you may be building a Java component.

### Automatic code generation for your project

As you build or edit an applet, code is dynamically generated for the components in your program. Visual Cafe generates Java source code for your programs and automatically adds the source file to your project. Furthermore, Visual Cafe fully supports [two-way editing](#) of visual components. For example, if you open the source code for your applet and change the label of a button or even declare a new button, the change is visible in the form as you write the code.

Double-click a component on your form to open the source window, and the Source Window shows the [events](#) available for the component. Visual Cafe also generates event-handler code and embeds it into your project. Select an event from the drop-down list and Visual Cafe inserts a method for the event, and you are ready to add the body code for the method.

### Creating projects

You create new projects from the File menu. Just as a word-processing document contains text and perhaps graphics, projects are containers for forms, components, source file, etc. When a project is created, a Project window is opened and each item in the project is shown in the project list. If you create more than one new project, a Project window is opened for each new project.

### Default projects

When you start Visual Cafe, a new project is automatically created. When the project is generated, the Project window is updated to show an item representing each of the components in the project. Initially, the applet "Applet1" appears in the Project Window. The applet opens in a separate window where the window is active and ready to be edited.

Initially, the default project is a JFC Applet project. The default project type is shown in the New Project dialog box in boldface text with an asterisk before its name. You can change the default project type in the New Project dialog box.

Projects include a set of [components](#) including [forms](#) (such as windows and dialog boxes). A typical project is a collection of forms. Forms are containers for components, such as static text, list boxes, menu bars, graphics, scroll bars, and buttons. In addition to these standard Java class components, Visual Cafe also provides you with many other components. All Visual Cafe components are organized in the Visual Cafe Component Library.

The Project window lists all forms, components, and classes in a project.

Each object in the project is made of two elements:

- a visual attribute
- the object's class (Java code)

As you build or edit an applet, code is dynamically generated for the components in your program. Visual Cafe generates Java source code for your programs and automatically adds the source file to your project. Furthermore, Visual Cafe fully supports [two-way editing](#) of visual components. For example, if you open the source code for your applet and change the label of a button or even declare a new button, the change is visible in the form as you write the code.

Packages view shows **java** source files, not the corresponding compiled **class** files, because you are interested in source files during development.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

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

Extension	Description
<b>.vpj</b>	The Visual Cafe project
<b>.vep</b>	Visual Cafe project options and file list
<b>.ve2</b>	Secondary project information
<b>.cdb</b>	Compiled database that Visual Cafe uses to track compilation dependencies (created after compilation)

These files are not needed for deployment. The **vep** file name displays in the title bar of the Project window; the rest of the project files do not appear in the Project window.



**Note** In Visual Cafe 1.0, you opened a project by opening the **vpj** file. In Visual Cafe 2.0, you open a project by opening the **vep** file.

Visual Cafe can also create these files and store them in the project directory:

Extension	Description
<b>.java</b>	A Java source file, such as for an applet
<b>.class</b>	A compiled version of a Java source file
<b>.html</b>	An HTML file


Double-clicking a Java or HTML file in the Project window opens that file in a Visual Cafe editor. For deployment, you need the compiled version of your Java files (the **class** files) and, for applets, any HTML files you want to use for your Web pages. In addition, you also need your graphics 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** files are used for deployment.



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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 **See Also** **Open**  
**File**  
**File**

 **See Also** **New Project**

The Project window displays a list of objects in the current project (Objects tab) and the Java packages that are associated with the project (Packages tab).

You can expand and collapse project [forms](#) to view the object's child components by using the standard Windows keyboard sequences and visual expansion element (+).

Clicking a Project window makes that project current.

To access other tools from the Project window:

<u>Double-click</u>	<u>Opens</u>
component	Form Designer
Java or HTML file	Source window
menu component	Menu Designer

#### **Tasks**

[Adding objects to a project](#)

[Connecting two components](#)

[Creating a form](#)

[Deleting objects from a project](#)

[Editing a form](#)

[Viewing a project by objects](#)

[Viewing a project by packages](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

If you are not going to generate code with Visual Cafe again, you can reuse a file from an existing project by adding that file to a new project. See [Adding an existing file to a project](#).

However, if you will be making changes that change code, you should instead copy the file. See [Copying a file in a project](#).



**Note** File names are relative to the current project. For example, if you add a file called foo.java and it resides in a subdirectory (called foo1) of the project directory, the file name is stored as foo1\foo.java and is not fully qualified. If foo1 was a directory at the same level as the project directory, the file would be stored as ../foo1\foo.java.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can create documents from within Visual Cafe. You may want to create documents for custom HTML pages and Java files.

1. Choose File  New File.

A new, untitled, blank file opens in the Source window.


2. In the Source window, edit the document.

3. Save the document with File  Save As or Save.

When you save the file, be sure to select the correct document type: HTML, Java, definition.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can debug your program in Netscape instead of the Applet Viewer.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Debugger tab.
4. Choose General in the Category field.
5. Select Debug in Netscape if you want to debug in Netscape instead of the Applet Viewer.
6. Click OK.

The change takes effect the next time the Java program is executing in the debugger.


## In the Project window, Objects view

<u>Dragging</u>	<u>From</u>	<u>Into</u>	<u>Effect</u>
A component or <b>java</b> file	Component Library or Palette	Project window	Copies the component to the new project. 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 <b>java</b> file is added to your project; components added to a container mean that code is generated in the <b>java</b> file for the top-level container.
	Project window	A different Project window	Copies the file or component to the new project.
	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 directory. See <a href="#">Sharing files between projects</a> .)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can use version control software that integrates into the Visual Cafe environment. If you have installed version control software on your computer, you will be able to select it in the Visual Cafe project options. You enable version control software on a per-project basis.

See the documentation provided with your version control system for more information.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Version Control tab.
4. Choose the version control software you want to use.

You must properly install the software before you can use it.

5. Click OK.

The change takes effect immediately.



**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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can use version control system (VCS) software that integrates into the Visual Cafe environment. Visual Cafe supports version control in two ways: the Microsoft Source Code Control (SCC) interface or the Visual Cafe Version Control interface. If you have properly installed version control software that works with Visual Cafe on your computer, you will be able to select it in the Visual Cafe project options. You enable version control on a per-project basis.

See the documentation provided with your version control system for more information.


 See Also

**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 (supported by vendors such as StarBase Versions and MKS Source Integrity), choose that version control install option, which might be listed as a Visual Cafe install option.

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

 See Also

**Note** After creating a new project, you should save the project to the location you want it before choosing a version control system.

2. Choose Project  Options.
3. In the Project Options dialog box, click the Version Control tab.
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 have the line “SCC Provider: StarBase Versions;” whereas, if StarBase does not use the SCC interface, the line would show “StarBase Versions;” but not “SCC Provider.”

5. Click OK.

The change takes effect immediately.

If you are using the SCC interface, the Version Control Options dialog box appears, the provider may prompt you to create a new project or open an existing project, and then the Add to Version Control dialog box appears.

 See Also

[Details on setting version control options](#)

 See Also

[Details on adding files to version control](#)

 See Also

**Tip** If you are using the SCC interface, choose Tools  Version Control to view the version control menu items available to you. For example, you can set version control options and check files in and out.

## Designing forms, including Form Designer, Property List, and layout managers

See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe makes it much easier 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, dialogs, text, buttons, and graphics, to the project. 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. Components can be added to a project from the Component Library or Palette.
- Open top-level components, or [forms](#), in the Form Designer, then visually arrange components on the form. You can drop components into the Project window or directly into the Form Designer, and then arrange them on the Form Designer.
- Set component [properties](#), such as color and text, in the Property List. These changes are immediately reflected in the Form Designer.
- Use customizers to edit components when you want a wizard-like interface for changing properties. These changes are immediately reflected in the Form Designer.
- Create [interactions](#) between components. For example, if a user clicks a Clear button, a text field clears.

### What are Visual Cafe forms?

Some components can only appear at the top level in the Objects view of the Project window. These are the Visual Cafe forms, including the Applet, Frame, Window, and some dialog components. If you look in the Component Library, the forms provided with Visual Cafe are all in one Forms group. A form can contain other components, such as text, buttons, and graphics.

### Working in the Form Designer

The Form Designer is a separate Visual Cafe window that displays a form. The window size is the size of your form. Components contained by the form appear on the form. For example, if you run an applet in a Web browser, the applet will appear the same size as it does in the Form Designer window. The components contained by the form appear as shown in the Form Designer. While working with Visual Cafe, you can have multiple Form Designer windows open at once — one per form.

The Form Designer uses an integrated [Java Virtual Machine](#) 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, such as an on-screen animation, can be accounted for in the layout design.

### Creating Java code

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.

### Using top-level components in Java programs

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 an 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 an Application template provided with Visual Cafe, the main window is already set up for you programmatically.

Other top-level components are the Window component and some dialog components. When you create a Java program, you can display these top-level components from an Applet or Frame, for example.

### 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

In general, to design your graphical user interface (GUI), you follow these basic steps:

1. Add [forms](#) to the project.

When you created the project, your project template might have already added forms to your project.

 See Also

[Details on this step](#)

2. Add [components](#) to your forms and arrange them.

 See Also

[Details on this step](#)

3. Modify component [properties](#).

 See Also

[Details on using the Property List](#)

 See Also

[Details on using customizers](#)

4. Create component [interactions](#).

 See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Some [components](#) can contain other components, such as text, buttons, graphics, charts, and so on. These components are called [containers](#).

In general, while components are independent objects, there is a certain parent and child relationship that exists between containers and other components.

Java defines containers as components that hold any number of components, including other containers. Containers hold and organize your components, but they also contain code for [event](#) handling and many essential things such as controlling the cursor image and the program's icon. All containers allow for operations such as adding, removing, and painting their contents, or components. Containers allow for the grouping of related components together and treating them as a single unit, simplifying the amount of programming you have to do.

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 components as well.

The top-level containers, or forms, are Applet, Frame, Window, and some dialogs. When displayed in the Form Designer, the size of the window is the bounds of the form.

Other containers are panels, MenuBar, Menu, and some dialogs. These containers can be contained by a form, but are not forms themselves.

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



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

When editing the source code for a component directly, 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 a background thread 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. The parser automatically detects changes to your Java source code and refreshes the object information as you save the source changes.

When you edit source files outside of the Visual Cafe environment, the changes are reflected the next time that you open the project in Visual Cafe. See [Adding Code to a Java Source File](#) for important guidelines.

## Form Designer

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Form Designer is the main tool for designing the interface to your [applet](#) or [application](#). You use the WYSIWYG Form Designer to design windows, applets, dialog boxes, and message boxes.

The Form Designer is fully integrated into the Visual Cafe development environment. As you design your [forms](#), its Java source, properties, and project are dynamically kept in sync. Visual Cafe provides this powerful functionality by using a Java virtual machine to read and maintain source code while you design.

You can perform these tasks that relate to the Form Designer:

[Adding a form to a project](#)

[Adding components to a form and arranging them](#)

[Dragging-and-dropping into the Form Designer](#)

[Renaming a component in a project](#)

[Deleting a component from a project](#)

[Overlapping components in applets](#)

[Tabbing between fields](#)

[Displaying invisible components](#)

[Viewing and changing component properties](#)

[Adding an interaction between components](#)

#### **To access the Form Designer**

- Double-click the form in the Objects view of the Project window. Or right-click the form, and choose Edit *form*.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




Some [components](#) can only appear at the top level in the Objects view of the Project window. These are the Visual Cafe forms, including the Applet, Frame, Window, and some dialog components. If you look in the Component Library, the forms provided with Visual Cafe are all in one Forms group. A form can contain other components, such as text, buttons, and graphics.

The form for an [applet](#) is the Applet component. The form for an [application](#) with a GUI is the Frame component.

Other forms are the Window component and some dialog components. When you create a Java program, you can display these forms from an Applet or Frame, for example.

#### To add a form by using the Insert menu

1. While the project is selected, choose Insert  Form.
2. Select a form template.
3. Click OK.

The new form is added to the project and the Form Designer opens.

#### To add a form by using drag-and-drop

You can drag a form from the following areas into the Project window:

- the Component Palette
- the Component Library
- the same Project window (press CONTROL and drag to copy the form)
- a different Project window



**Note** If the parse fails, you can see a file in the Packages and Files view, but not see 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 code to a Java source file](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

After you add a [form](#) to a project, you can add [components](#) to it and arrange the components in the Form Designer.

To display the Form Designer, double-click the form in the Objects view of the Project window. Or right-click the form, and choose Edit *form*.

Then you can perform these tasks:

- Use the default layout tools or choose a layout manager for the form.

 See Also

[Details on this task](#)

- Add components to the form.

 See Also

[Details on this task](#)

You can copy components to add them.

 See Also

[Details on copying components](#)

- Size components, and position the components on the form by dragging them.

When a component is selected and you move the cursor over a square (handle) on the edge of the component, the cursor changes to a two-way arrow, which means you can click 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 means you can click and drag the component to another location.

To select multiple components, Shift-click them or drag over them.

 See Also

#### 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](#) for more information.
- For deployment purposes, it is best to use a relative URL. That way you can move your Java program to another location and files can still be found. A relative URL specifies the 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/file*.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can use standard graphical user interface (GUI) techniques for dragging and dropping [components](#) into a [form](#) displayed in the Form Designer.

When dragging components, a + appears over the cursor when a copy operation is being performed.

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. While dragging, a circle with a line through it means the operation is not allowed.

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 location of the item. Or press CONTROL and drag an item to copy 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 location of the component; or press CONTROL and drag to copy a component.
Form Designer	Form Designer in different project	Copies the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe allows you to call a customizer to configure the characteristics or behavior of a [bean](#). Use a customizer when you want more guidance about configuring a bean's behavior than the Property List provides.

If Visual Cafe detects a `bd.SetValue("hidden-state", Boolean.True)` statement in the BeanInfo of a bean, you can access the customizer and properties exposed in the Property List can be edited. If a bean has a hidden state set to true, it may be designed to work with serialization.

If Visual Cafe cannot detect whether a bean has a hidden state set to true, any changes made in the customizer that do not correlate to an item on the Property List will be lost.



**Note** Customizers will vary from bean to bean since each bean has a unique customizer created by the bean developer. Not all beans may have a customizer.

#### To access a customizer

- Right-click a component in the Project window (Objects view) or Form Designer and choose Customize.
- Select the component in the Project window (Objects view) or Form Designer, then choose Object Customize.
- Double-click the Customize property in the Property List.

The customizer menu item is not available unless the component has a customizer.


After you access the customizer, you can make the changes you want.

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```

 See Also

 See Also

Different Web browsers use a different “z-order” 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 is a good idea to “sandwich” the InvisibleHTMLLinks on top and beneath lightweight components they overlap. Use

Layout  Send to Back or Send to Front.

You need to pay attention to whether heavyweight and lightweight components overlap, because heavyweight components always overlap lightweight components. See [Using lightweight components](#) for more information.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



You can allow users to tab between fields on a [form](#) by placing the fields in a KeyPressManagerPanel container. The tab order of the [components](#) contained by the panel is listed in the Project window. To change the tab order, change the order of components in the Project window list.

To tab out of the panel to the next traversable component in the Java application, press CTRL+TAB.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```


 See Also

 See Also

An *invisible [component](#)* 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. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not effect the [form](#) layout.

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 Layout  Invisibles.

 See Also

**Note** When you add an invisible component to a form, all invisible components display automatically.

## Menu Designer

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}




You can add menu bars to frames (they inherit from MenuContainer); applets and dialogs do not support menu bars. To create a menu bar, you can do the following:

- Add a MenuBar component to a frame.
- Add one or more Menu components to the MenuBar container.
- Add one or more MenuItem components to a Menu container. You can also add CheckboxMenuItem components.
- Add one or more submenus to a menu item.
- Add command key equivalents to menu items.
- Add [interactions](#) between menu items and other components in your project. For example, specifying that a menu item open a dialog within the same project.
- Bind [custom code](#) to menu items.

The Frame component in the AWT Application template already has a menu bar, which you can modify and enhance.

The Menu Designer makes it easier 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.
- In the Project window, select the MenuBar component, then choose Object  Edit MenuBar or right-click and choose Edit MenuBar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right mouse click in the Menu Designer, Visual Cafe displays a pop-up menu with these commands:

**Cut/Copy/Paste**

These edit commands perform standard Windows functions.

**Insert Menu**

Inserts a menu in the menu bar; you can then add menu items to the menu. Also lets you create a submenu for the selected menu item.

**Insert Menu Item**

Inserts a menu item into the selected menu that is in a menu bar.

**Insert Checkbox Menu Item**

Inserts a checkbox menu item into the selected menu that is in a menu bar.

**Edit Source**

Opens the source file for the menu bar so you can bind code to menu items.

**Add Interaction...**

Opens the Interaction Wizard so you can define an interactive behavior between the selected menu item and another component in the same project.

**Properties**

Displays the list of properties for the selected menu component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

You can add menu bars to frames (they inherit from MenuContainer); applets and dialogs do not support menu bars. The Menu Designer makes it easier to create a menu bar by letting you edit a visual representation of the menu bar.

In general, to design your menu bar and menus, you follow these basic steps:

1. Add a menu bar to a frame.

If you used the AWT Application project template, you already have a menu bar in the frame it created for you.

 See Also

[Details on this step](#)

2. Add menus to your menu bar.

 See Also

[Details on this step](#)

3. Add menu items to your menus.

 See Also

[Details on this step](#)

You can also add menu items that can have submenus.

 See Also

[Details on this step](#)

4. Edit the menu structure as needed.

 See Also

[Details on this step](#)

5. Associate command keys with menu items, as needed.

 See Also

[Details on this step](#)

6. Add [interactions](#) between menu items and other components in your project, as needed. For example, specifying that a menu item open a dialog within the same project.

 See Also

[Details on this step](#)

7. Bind [custom code](#) to menu items, as needed.

 See Also

[Details on this step](#)

 See Also


**Tip** You can save time by copying menu bars and menus from other projects. See [Copying a component in a project](#) for more information. You can also add commonly used menu bars and menus to the Component Library and Palette. See [Using the Component Library](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


You can add the MenuBar component to a frame in the following ways:

- Drag a MenuBar component from the Component Library or Palette into the Project window or Form Designer.
- Use an Insert menu item:
  - While the Project window or Form Designer is active, choose Insert  Component.
  - Right-click the frame in the Project window and choose Insert Component.
- Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

To open the Menu Designer:


- Double-click a MenuBar component in the Project window or Form Designer.
- In the Project window, select the MenuBar component, then choose Object  Edit MenuBar, or right-click and choose Edit MenuBar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can add a Menu component to a MenuBar component.

1. Add a Menu component to a MenuBar component in one of the following ways:
  - Drag a Menu component from the Component Library or Palette into its position in the Project window or Menu Designer.
  - Use an Insert menu item:
    - 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 Insert  Component.
  - Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

2. In the Property List, set the menu properties, including the menu name in the Label property.


The Help Menu property lets you integrate the menu item into an existing Help menu that is part of the operating system, for example.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can add a MenuItem or CheckboxMenuItem component to a Menu component. To create a menu item that can have submenus, see [Adding submenus](#).

1. Add a MenuItem or CheckboxMenuItem component to a Menu component in one of the following ways:
  - Drag a component from the Component Library or Palette into its position in the Project window or Menu Designer.
  - Use an Insert menu item:
    - 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 Insert  Component.
  - Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

2. In the Property List, set the menu item properties, including the menu item name in the Label property.

 See Also

**Tip** To add more menu items, you can now select the empty bottom menu item in the list and enter its properties.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can add a MenuItem or CheckboxMenuItem component to a Menu component that is within another Menu component.

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, in the Property List set the menu properties, including the name of the menu in the Label field.
3. Click the submenu box, then add a menu item.



[Details on this step](#)



**Tip** To add more submenu items, you can now select the empty bottom submenu item in the list and enter its properties.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can quickly add command keys to menu items.

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.



[Details on this step](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also


 See Also

You can change the menu structure as needed.

**To move items in the menu structure**

- Move items in the Menu Designer or Project window to visually change it.

**To delete a menu or menu item**

- Select the component in the Menu Designer or Project window, then press DELETE or choose Edit  Delete.

 See Also

**Note** If you delete a component from a container, you must manually delete any custom code involving that component.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Code can be bound to a menu item just as it can be bound to a component. Menu items respond to one event: `ActionEvent`. This event occurs when the user selects the menu command.

1. Open the menu bar in the Menu Designer or Project window.
2. Select the menu item.
3. 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](#).

## Property List

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Each component 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.

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 pull-down menu. When multiple components are selected, only their common properties are shown and editable.

Properties with multiple values are marked with a plus sign (+); for example, the Font property. Properties that are defined using a dialog box are marked with the ellipsis button (...).

If you change the properties for a component in a project, only that project is affected. If you change the properties of a component in the Component Library, the change appears anytime you add the component to a project. Projects that already contained the component before the change are not affected.

Components may also have a customizer, a wizard-like interface bean developers use to help guide you through changing a component. See [Using a customizer to configure a component on a form](#) for more information.

#### To display the Property List

- Choose View  Property List.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Each component 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 components in a project, select one or more components in the Form Designer or Project window.

To view or change the properties of components in the Component Library and Palette, select one or more components 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. Any projects that already contain the component before the change are not affected.

1. To display the Property List, choose View  Property List.
2. Select a component in the Form Designer, Project window, Component Library, or from the Property List pull-down menu. To select multiple components, CONTROL-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 title "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 are marked with a plus sign (+); for example, the Font property. Click the + to expand the list.

Properties that are defined using a dialog box are marked with the ellipsis button (...). Click the ... button to get the property editor.

4. Press ENTER or click somewhere else to make the change.



**Note** Press ESCAPE to cancel an edit and return the property to its previous value.


## Layout Managers

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```

 See Also

 See Also

When you do not use a layout manager, components are placed by exact pixel positions in a layout. In the Form Designer, you can do the following:

- Drag components on the Form Designer to position them.
- Explicitly set x and y coordinates in the Property List.
- Arrange objects with the Align, Center, Space Evenly, Make Same Size, Bring to Front, and Send to Back menu items of the Layout menu.
- Use the Form Designer grid, which you can set by choosing Layout  Grid Options.
- 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.

Java also provides layout managers that help you arrange [components](#) inside a [container](#) so the appearance adjusts automatically across platforms and Web browsers, and at varied screen sizes and resolutions:

#### **BorderLayout**

Arranges the 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.

#### **CardLayout**

Arranges the components on several cards. Only one card is visible at a time, which allows you to flip through the cards. 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.

#### **FlowLayout**

Arranges components in a row from left to right until there is no more room, then it starts the next row. You can specify center, left, or right alignment, as well as the horizontal and center gaps.

#### **GridLayout**

Arranges components in definable rows and columns. This layout is similar to FlowLayout except 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.

#### **GridBagLayout**

Arranges objects by size and space. Like GridLayout, GridBagLayout treats the form or panel as a grid of cells. Unlike GridLayout, however, a component can occupy more than one cell.

For information on BoxLayout, see [Arranging components in a Swing BoxLayout](#).

You specify a layout for a container by setting the Layout property. 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 are using.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe provides [layouts](#) (also called layout managers) to help you organize components on a [form](#) or in a [panel](#). You can group components in panels on your form and use different layout managers to suit the components contained by the panel.

For information on BorderLayout, see [Arranging components in a Swing BorderLayout](#).

#### To choose a layout manager


1. Open the Property List for a form or panel.
2. 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. Or, you might have to position components first.

3. Rearrange the components in the Form Designer, if needed.

#### To arrange components in a layout manager of None

When you choose a layout of None, components are placed by exact pixel positions on a form or panel. In the Form Designer, you can do the following:

- Drag components on the Form Designer to position them.
- Explicitly set x and y coordinates in the Property List.
- Arrange objects with the Align, Center, Space Evenly, Make Same Size, Bring to Front, and Send to Back menu items of the Layout menu.
- Use the Form Designer grid, which you can set by choosing Layout  Grid Options. See [Manipulating the Form Designer grid](#) 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 it on different operating systems and screens, as applicable. For maximum portability, it is sometimes best to not overlap components in your layout.

#### To arrange 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.

1. Choose BorderLayout for the Layout property of a form or panel.
2. If components are already on the form or panel, rearrange components, as needed. Set the Placement property for each component you want to position.



**Note** In the 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 Placement property is blank (which is the same as CENTER).

5. Test your layout by running it at different form sizes and resolutions. For example, you can resize the form when you run it from Visual Cafe.

#### To arrange components in CardLayout

Use CardLayout to arrange components on several cards. Only one card is visible at a time, which allows you to flip through the cards. You have to programmatically implement flipping through the cards. 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.

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 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 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 resolutions. For example, you can resize the form when you run it from Visual Cafe.

### To arrange 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.

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 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 resolutions. For example, you can resize the form when you run it from Visual Cafe.

### To arrange components in GridLayout

Use GridLayout to arrange components in definable rows and columns. This layout is similar to FlowLayout except 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.

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 nonzero, the Rows value is used.
3. If components are already on the form or panel, rearrange 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 resolutions. For example, you can resize the form when you run it from Visual Cafe.

### To arrange components in GridBagLayout by using the GridBag Constraints Editor

Use GridBagLayout to arrange components by size and space. Like GridLayout, GridBagLayout treats the form or panel as a grid of cells. Unlike GridLayout, however, a component can occupy more than one cell.

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 activated, choose Layout  Edit Constraints.

The GridBag Constraints Editor for that component appears. Only one GridBag Constraints Editor appears at a time.
4. Change Grid Bag Constraints 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.



[Details on Grid Bag Constraints](#)

5. Use the GridBag Constraints Editor or Property List to set the Grid Bag Constraints 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 Grid Bag Constraints 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 resolutions. For example, you can resize the form when you run it from Visual Cafe.
7. To close the GridBag Constraints Editor, click OK.

#### **To arrange components in GridBagLayout from the Property List**

1. 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.

2. Rearrange and add components as needed. In the Property List, choose a component, then set the Grid Bag Constraints properties to adjust its place in the layout. You can think of these properties as “suggestions” that GridBagLayout uses.

Press F1 on a Grid Bag Constraints property to get a description of it. Insets specify how much space to leave between the borders of a component and its display area.


3. Test your layout by running it at different form sizes and resolutions. For example, you can resize the form when you run it from Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



While designing a [form](#), the Form Designer grid is useful in laying out the form components.

**To open the Grid Options dialog box**

- While a Form Designer is the active window, choose Layout  Grid Options.

**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**

This option automatically aligns components to the grid when moving, sizing, and creating them.

- In the Grid Options dialog box, select or deselect Snap to Grid.

**To set the grid spacing**

- In the Grid Options dialog box, type the amount of space between grid points, both horizontally and vertically.

## Obsolete Topics

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

Within the Property List you can modify the [properties](#) of top-level components. When multiple components are selected, only common properties of the selected objects are shown and editable.

1. Select an object in the Project window, Form Designer, or from the Property List drop-down list box.
2. Click the appropriate property in the left column.

The right column activates and displays a list of valid values or the text string is made editable. Properties that have the "+" symbol next to their name are multiple value properties. When you edit these properties, either a dialog box displays or the individual property list expands.

3. Make the desired changes.
4. Press ENTER or select another property to save the changes.

**Note** Pressing the ESC key cancels any edits and returns the property to its previous value setting.

Property changes are immediately effective. If you change the properties of an object in the Component Library, you see the changes the next time that you use the object.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

Within the Property List you can change the current component by selecting the component's name for the Property List's drop-down list.

If the Form Designer is open, any changes that you make are dynamically reflected.

To select multiple components, you must select them in the Form Designer by CONTROL-clicking the desired components, SHIFT-clicking, or dragging to select.


```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can view the properties of multiple components at the same time. Only a subset of the properties common to all selected components displays.

1. In the Form Designer, CONTROL-click the components that you want to edit as a group.  
In the Property List, you see the heading title "Multiple Selection."
2. Change properties as needed.  
The change is applied to all selected components.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

Property values are changed in the Property List.

1. In the Form Designer, select an object.
2. Choose View  Property List.
3. In the Property List, click the property that you want to modify.
4. Edit the value directly in the property field or change the value by using the dialog box that displays when you click the ... button.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The Component Library window contains all the [components](#) in the Component Library. The Component Library can be used to add items to projects, [forms](#), and to the palette. Components defined in a [.class file](#) can be dragged and dropped from the File Manager onto the Component Library to make them permanent components in the library. The default Component Library is installed in the Visual Cafe library directory.

### **Adding components to the Component Library**

Forms and applets, as well as menu bars and other types of components, can be saved and reused. To save a form as a reusable component, you activate the form window and choose Add to Library from the Object menu. This opens a dialog box where you can enter a name and description and select an icon for the form template.

The Add to Library command adds the component to the Visual Cafe Component Library. When you add a component to the library, all aspects of the component are saved, including any child components such as a form, code, and current property settings.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

Forms are the basis for creating a user interface to your applets and applications. With forms, you can create and add [components](#) like windows, dialog boxes, graphics, and sound to your applets and applications. 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 with code that handles [events](#) like mouse interactions.

Forms are also 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. Some examples of components are:

- Check Boxes
- Radio Buttons
- Text Boxes
- Scroll Bars

To add a component to a form, select the component from the Component Palette and drag it into the Form Designer. The Component Palette contains a variety of components that you can add to forms including standard Java window components. Visual Cafe also includes an extensive list of custom and third-party components.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can copy components and menu bars to another form, within the same form, or to another project by:

- dragging and dropping the component to the target form
- copying and pasting the component with menu commands and the toolbar
- selecting the component and using the Copy and Paste pop-up menu commands

#### Notes

- The object's bound events are not moved to the new location.
- You must manually move any code to the new form.
- Forms and components must have unique names. If necessary, an object is renamed when pasted.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can move a component to another form by either of these methods:

- In the Project window, drag the component on top of the new form.
- In the Form Designer, cut and paste the component.

#### Notes

- Moving a component to another form does not move the associated code. You need to delete any code that references the component and add it to the new form's source code.
- In the Form Designer, dragging a component from one form window to another moves the component.
- The object's bound events are not moved to the new location. You must manually move the code to the new class.
- Forms cannot be moved into another form.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

Components are deleted using one of these methods:

- Select the component and press DELETE
- From the Project window, expand the object listing of the form that you want to edit, select the component, and press DELETE

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

All form editing is done in the Form Designer. Some editing tasks that you may want to perform are:

[Adding components to a form](#)

[Deleting components from a form](#)

[Modifying a component's properties](#)

[Adding a connection between two components](#)

[Adding a component to the Visual Cafe library](#)

[Arranging components on a form](#)



[Enabling and disabling the grid](#)

[Enhancing an object's Java code](#)

[Displaying invisible components](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

Copying menus allows you to reuse menus that are common among forms and projects.

1. To open the Menu Designer, double-click the menu bar component on a form or in the Project window.
2. In the Menu Designer, click the menu that you want to copy.
3. Choose Edit  Copy.
4. Open the target form and double-click the target menu bar component.  
The component displays in the Menu Designer.
5. Select a location for the new menu. You can paste a menu to the top-level or within a menu to create a hierarchical menu.
6. Choose Edit  Paste.

See Also

See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Your Cafe project is converted automatically when you open it with Visual Cafe.

 See Also

**Warning** Converted projects cannot be saved back into Cafe format. However, you can move Java source files between Cafe and Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

Your Visual J++ version 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 open the project file directly.

For a more complete list of considerations, see [Considerations when importing Visual J++ projects](#).


### To import a project through the dsw workspace file

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 directory as the workspace. So if multiple projects are in your workspace, you should convert the projects that are in their own directories first, then convert the project in the same directory as the workspace file. It is best to work from a copy of your Visual J++ files instead of the originals.

1. Choose File  Open.  
The Open dialog box displays.
2. Navigate to the directory with 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 display.
4. Select the **dsw** file from the list. Then click Open.
5. If there is more than one project in the workspace, a dialog box appears that asks what 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 are 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.

 See Also

[Details on this step](#)

8. If the Visual Cafe project was in a different directory than the **dsw** file, choose File  Save As to save your new project to the directory where the project **dsp** file is. If you want, you can rename the project at the same time.


 See Also

[Details on this step](#)

If you save the project to another directory, you could delete the Visual Cafe project files in the same directory as the workspace. Or you can just let them be overwritten when you import the next project.

### To import a project through the dsp project file

It is best to work from a copy of your Visual J++ files instead of the originals.

1. Choose File  Open.  
The Open dialog box displays.
2. Navigate to the directory with 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 display.
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 are 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.



[Details on this step](#)

7. Save your new project.



[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Your Visual J++ version 1.1 project is converted automatically when you open a workspace file (**dsw**) or project file (**dsp**) with Visual Cafe. While Visual Cafe has projects, Visual J++ has workspaces that contain projects. A workspace is made of a workspace file and an **opt** options file. Information about the projects a workspace contains is stored in a project file (one per project) and the options file. Because the name of the options file is not stored in the project file, it is 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
- you import a **dsp** file that has the same name as the **opt** file and the **opt** file is in the same directory.

**Visual Cafe project option...**

**Project type** field in Project tab

**Start with Web page** field in Project tab

**Main class** field in Project tab

**Runtime arguments** field in Project tab

**Class Files** list in Directories tab

**Visual J++ configuration that sets it...**

The **Debug/Execute project under Browser** and **Standalone interpreter** options determine the setting.

The **Use parameters from HTML page** value is used; or the **Enter parameters below** option tells Visual Cafe to use Automatic.

The **Class for debugging/executing** value is used.

The **Program arguments** value is used.

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...**

**Output directory** field in Directories tab

**Show compiler warnings** option in Compiler tab

**Generate debug information** option in Compiler tab

**Visual J++ configuration that sets it...**

The **Output directory** value is used.

A **Warning Level** of None tells Visual Cafe to deselect the option; any other value means the option is selected.

The **Generate Debug info** value is used.

Here are some more considerations when importing Visual J++ projects:

- If you want to maintain your Visual J++ workspace and projects, you should copy the files to another directory 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](#).
- 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 probably want to make sure your output directories are different for each product.
- You cannot 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 cannot 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 per project. If the project type is different for the debug and final option sets, Visual Cafe uses the project type specified for debug.

#### See Also

**Note** In Visual Cafe Professional edition, 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](#) for more information.

#### See Also

**Tip** In Visual Cafe, when you close the Project window, the project closes. Whereas in Visual J++, when you close the Workspace (a project), just the window closes.

**Native application, DLL, and library help topics (Project Options are in project.doc)**

See Also

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Visual Cafe Professional edition lets you create native Win32 Windows applications that run like any other application on a Windows computer. You do not need java.exe to run the application. You can also create native Win32 libraries (**lib** files) and Dynamic Link Libraries (DLLs).

Native libraries and DLLs are provided with Visual Cafe to help you develop and deploy native applications and DLLs. See [Native libraries and DLLs included with Visual Cafe](#) for more information.

Visual Cafe also has project templates to help you get started quickly with these tasks:

- [Assembling native GUI applications](#)
- [Assembling native console applications](#)
- [Assembling native DLLs](#)

Be sure to look in the Samples directory for detailed examples and instructions on how to create the examples.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

Visual Cafe Professional edition includes libraries and DLLs to support native application and DLL development.

Library	DLL	Description
Snjrt.LIB	Snjrt11.DLL	Core API
Snjbeans.LIB	Snjbeans11.DLL	Beans
Snjres.LIB	Snjres11.DLL	Resource support including audio
Snjnet.LIB	Snjnet11.DLL	Networking and Internet support
Snjmath.LIB	Snjmath11.DLL	Math
Snjzip.LIB	Snjzip11.DLL	Compression
Snjsec.LIB	Snjsec11.DLL	Security
Snjrmi.LIB	Snjrmi11.DLL	RMI
Snjawt.LIB	Snjawt11.DLL	AWT
Symbeans.LIB	Symbeans.DLL	Visual Cafe components
Snjsql.lib	Snjsql11.DLL	Standard JDBC support (Database edition only)

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



For a native project to link, there may be extra steps you need to perform that you do not 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 some reference made to a class in your project. Typically, this means an object file corresponding to a class that your project uses did not 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 might report "symbol undefined" errors in the Messages window when you build your project. 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 mechanisms 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 Visual Cafe Samples directory.

As for bytecode programs, native Java applications allow programs to dynamically load arbitrary classes at runtime, 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 Dynamic Link Library (DLL) and be registered in the Windows registry by using the SNJREG tool.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Unlike bytecode Java applications, native Win32 Java applications cannot use the classpath to find classes and packages. Instead, the Windows PATH is searched to locate classes and packages in DLLs. Classes that are dynamically loaded must be stored in the Symantec registry. 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 stored in the DLL will not be found by your program and you 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 DLLs assigned to a class already registered.
-noprompt	Disables the prompt before making changes to existing entries
-nowarn	Turns off all warnings
-verbose	Reports all registrations made
-reg:file.reg	Creates a registry file

To register a package in a **dll** or **exe** file by using SNJREG:

1. Open a DOS Window.
2. Change to the directory where the DLL files are located.
3. Enter:

**SNJREG** [ *options* ] { *file.dll* }

*options* can be any of the parameters in the previous table. *file.dll* is the name of a DLL containing the classes or packages you want to register. You can specify as many DLLs as you want, delimited by a space. If the DLLs are not in the current directory, specify the DLL name including the full path.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe Professional edition provides powerful tools for assembling a native, Win32 application quickly. In general, the development cycle is made of these basic steps:

1. Create a new project.

To get started quickly, you can use the Win32 AWT Application project template.

 See Also

[Details on this step](#)

2. Design the user interface.

 See Also

[Details on this step](#)

3. Enhance Java source code.

 See Also

[Details on this step](#)

4. Set project options.

If you started with the Win32 AWT Application project template, you probably do not need to set project options.

 See Also

[Details on this step](#)

5. Test run the application in Visual Cafe.

 See Also

[Details on this step](#)

6. Debug the application, if needed.

 See Also

[Details on this step](#)

7. Test run the application outside of Visual Cafe.

8. Deploy the application.

 See Also

[Details on this step](#)

 See Also

**Note** Look in the samples directory for examples of creating and using native applications.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe Professional edition provides powerful tools for assembling a native, Win32 application quickly. In general, the development cycle is made of these basic steps:

1. Create a new project.

To get started quickly, you can use the Win32 Console Application project template.

 See Also

[Details on this step](#)

3. Enhance Java source code.

 See Also

[Details on this step](#)

4. Set project options.

If you started with the Win32 Console Application project template, you probably do not need to set project options.

 See Also

[Details on this step](#)

5. Test run the application in Visual Cafe.

 See Also

[Details on this step](#)

6. Debug the application, if needed.

 See Also

[Details on this step](#)

7. Test run the application outside of Visual Cafe.

8. Deploy the application.

 See Also

[Details on this step](#)

 See Also

**Note** Look in the samples directory for examples of creating and using native applications.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe Professional edition provides powerful tools for assembling a native, Win32 DLL quickly. In general, the development cycle is made of these basic steps:

1. Create a new project.

You can start with the Win32 Dynamic Link Library project template.

 See Also

[Details on this step](#)

2. Design the user interface.

 See Also

[Details on this step](#)

3. Enhance Java source code.

 See Also

[Details on this step](#)

4. Set project options.

If you started with the Win32 Dynamic Link Library project template, you probably do not need to set project options.

 See Also

[Details on this step](#)

5. Test run the DLL in Visual Cafe.

 See Also

[Details on this step](#)

6. Debug the DLL, if needed.

 See Also

[Details on this step](#)

7. Test run the DLL outside of Visual Cafe.

8. Deploy the DLL.

 See Also

[Details on this step](#)

 See Also

**Note** Look in the samples directory for examples of creating and using DLLs.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



To deploy your native Win32 application, you need 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 redist directory. You only need snjrmregistry.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 snjrmregistry.exe.) See [Native libraries and DLLs included with Visual Cafe](#) for a list of DLLs included with Visual Cafe.

You also need any DLLS you wrote yourself.

You need to register the DLLs on the computer where the executable will run. To do so, you can use the SNJREG tool. See [Registering DLLs with SNJREG](#) for more information.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



To deploy your DLL, you need the required DLLs.

Visual Cafe comes with the standard DLLs you need to run your Java programs. The redistributable Visual Cafe DLLs are in the `redist` directory. You only need `snjrmregistry.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 `snjrmregistry.exe`.) See [Native libraries and DLLs included with Visual Cafe](#) for a list of DLLs included with Visual Cafe.

You also need any other DLLs that your DLL requires.

You need to register the DLL on the computer where the DLL is used. To do so, you can use the SNJREG tool. See [Registering DLLs with SNJREG](#) for more information.

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 class path and the library should be specified in the compiler options for the project. To run a program that uses a DLL, all you need is the DLL, and you need to register it.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Most Java programmers do not need to directly use OPTLINK and SMAKE to create native Java programs with Visual Cafe Professional edition.

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 directly only if you need more flexibility while linking your programs.

SMAKE is a make tool for command-line users. It is recommended that you instead use Visual Cafe projects, which are a lot easier to use. Only very complicated programs, such as those that include multiple languages, might require the use of makefiles.

Keep in mind that while OPTLINK and SMAKE run under both MS-DOS and Windows 95/NT, and that 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 or Windows NT.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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.

Following is the format:

**OPTDUMP** [ *options* ] *infile* [ *outfile* ]

/B[SeekOffset[,length]]      Force binary dump

/C      Disable CodeView display

/N      No logo

/P      Enable PharLap # RVA's

/H      Disable **exe** header display

/V      Verbose PE object display

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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. It is intended to be invoked from the Visual Cafe Bin directory.

Following is the format:

C:\vcafe\bin> **cafever** [ *component\_name* ]

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

coff2omf converts coff object (**obj**) files to omf format. Visual Cafe uses omf for the format of object files. Some other vendors use coff format **obj** files.

Following is the format:

**coff2omf** *filenames.obj*

It converts the file(s) in place. The **obj** extension is optional.

## Interactions

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



One of the most powerful features of Visual Cafe is the ability to quickly build a relationship between two [components](#). In Visual Cafe, this is called an [interaction](#). Interactions are implemented as methods and imply an event notification.

Anytime a button is pushed, or a box is checked, a program action called an [event](#) is generated. To link those events to a corresponding action in your program, you need an [event handler](#) that responds to that event.

Visual Cafe lets you specify these high-level interactions between components and events. As you create interactions by visually connecting components, Visual Cafe automatically generates code for the relationship. So, you can assemble interactive applets and applications without writing 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 on which a defined "action" happens. 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).

An interaction does not have to contain two components. You can create an interaction where the trigger and action component are the same. For example, you can create an interaction on an animation component where a mouse click anywhere within the boundaries of a component starts an animation in that component; another click ends the animation.

In the JDK 1.1 event model, when an interaction is created, code for the event handler, listener registration, and [adapter/listener](#) class with a call is automatically generated. If you want to delete an interaction, you need to delete the interaction from the Form Designer or the code.

Visual Cafe changes the source code in five ways when you create an interaction:

- 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.addTypeListener*) after the REGISTER\_LISTENERS tag. 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.
- Fifth, a call to the interaction method is generated in the event handler. If the call already exists, it is used.
- Sixth, an interaction method for the interaction specified in the wizard is generated. If the method already exist, its contents are replaced.

For example, if you have a button, called NextButton, that displays the next image of a slide show, called VacationSlides, the event handler could look like this and appear toward the end of the Java source file:

```
void NextButton_ActionPerformed(java.awt.event.ActionEvent event)
{
    // to do: code goes here.

    NextButton_ActionPerformed_Interaction1(event);
}

void NextButton_ActionPerformed_Interaction1(java.awt.event.ActionEvent event) {
    try {
        // VacationSlides Go to the SlideShow's next image
        VacationSlides.nextImage();
    } catch (Exception e) {
    }
}
```

More 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);
    }
}

```

If you then add an interaction between VacationSlides and a Label component, Visual Cafe would generate a new event handler for VacationSlides, and the listener registration code would change:

```

//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
NextButton.addActionListener(lSymAction);
VacationSlides.addActionListener(lSymAction);
//}}

```

And the adapter/listener class code would change:

```

class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == NextButton)
            NextButton_Action(event);
        else if (object == VacationSlides)
            VacationSlides_ActionPerformed(event);
    }
}

{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}

```

 See Also

 See Also

[Interactions](#) are implemented as methods in the [container](#) class of a [component](#). The interaction methods are called [event handlers](#). You can edit an interaction in the Interaction Wizard, by double-clicking its arrow in the Form Designer or right-clicking the arrow and choosing Edit Interaction. However, if you need to edit the code directly, you can follow these steps.

 See Also


**Warning** After you have edited an interaction in the source, you might not be able to edit it again in the wizard. If the wizard could have generated the interaction, Visual Cafe can convert it back to editable form in the wizard. However, if you find Visual Cafe cannot convert it back, once you have edited an interaction in the source you must continue to edit it in the source to avoid losing your changes.

1. Make the project active by clicking the Project window.

2. Choose View  Class Browser

You should see a listing of project classes in the Classes pane.

 See Also

**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed by choosing Classes  Options.

3. In the Classes list, click the container class that contains the trigger component.  
In the Members pane, you should see a list of methods and variables associated with the container class.
4. Identify the appropriate event handler. Event handlers are named using a combination of the component name and the trigger event; for example, NextButton\_Action.

 See Also

**Note** If you rename a component after an event handler is created, the method is not renamed.

5. Display the method's source code by clicking the event handler. The interaction code is marked by the `//{{ CONNECTION` comment and a comment that explains the interaction.

For example:

```
void NextButton_ActionPerformed(java.awt.event.ActionEvent event)
{
    // to do: code goes here.

    NextButton_ActionPerformed_Interaction1(event);
}

void NextButton_ActionPerformed_Interaction1(java.awt.event.ActionEvent event)
{
    try {
        // VacationSlides Go to the SlideShow's next image
        VacationSlides.nextImage();
    } catch (Exception e) {
    }
}
```

6. In the editing pane, make any source code changes you need.



[Details on this step](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

When deleting an [interaction](#) or [event binding](#), the following code is involved:

- the [event handler](#)
- the listener registration
- the event handler call in the [adapter/listener](#) class

For example, if you have a button, called NextButton, that displays the next image of a slide show, called VacationSlides, the event handler could look like this and appear toward the end of the Java source file:

```
void NextButton_ActionPerformed(java.awt.event.ActionEvent event)
{
    // to do: code goes here.

    NextButton_ActionPerformed_Interaction1(event);
}

void NextButton_ActionPerformed_Interaction1(java.awt.event.ActionEvent event) {
    try {
        // VacationSlides Go to the SlideShow's next image
        VacationSlides.nextImage();
    } catch (Exception e) {
    }
}
```

The event handler contains the interaction, which is marked by the CONNECTION tag.

More toward the middle of the Java source file would be the listener registration:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
NextButton.addActionListener(lSymAction);
//}}
```

```
//{{REGISTER_LISTENERS
SymAction lSymAction
NextButton.addActionL
//}}
```

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);
    }
}
```

```
class SymAction imple
{
    public void actio
    {
        Object object
        if (object ==
            NextButton
    }
}
```


If you then add an interaction between VacationSlides and a Label component, Visual Cafe would generate a new event handler for VacationSlides, and the listener registration code would change:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
NextButton.addActionListener(lSymAction);
VacationSlides.addActionListener(lSymAction);
//}}
```

And the adapter/listener class code would change:

```
class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == NextButton)
            NextButton_Action(event);
        else if (object == VacationSlides)
            VacationSlides_ActionPerformed(event);
    }
}
```

```
class SymAction imple
{
    public void actio
    {
        Object obj
        if (object
            NextBu
        else if (
            Vacati
    }
}
```

You can delete an interaction by pressing DELETE or choosing Edit  Delete for any of the following items:

- the interaction arrow on the Form Designer
- an object that triggers the interaction

However, if you want to modify the source code directly, you can follow any of these procedures.

#### To delete the event handler by using the Source window


1. In the Source window, open the source file containing the event handler.




[Details on this step](#)

2. In the Objects drop-down list of the Source window, choose an object.
3. In the Event/Methods drop-down list, choose the event.  
Existing events and methods are shown in bold.
4. Manually remove the interaction in the event handler code, or remove the entire event handler if it only handles one interaction.

#### To delete the event handler by using the Class Browser

1. Make the project active by moving focus to the project's Project window.
2. Choose View  Class Browser.  
A listing of project classes displays in the Classes pane.
3. In the Classes pane, click the container class that contains the trigger component.  
In the Members pane is a list of methods associated with the container class.



**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed by choosing Classes  Options.

4. Select the appropriate event handler for the interaction. An event handler is named using a combination of the component's name and the trigger event; for example, `NextButton_Action`.



**Note** If you rename a component after an event handler is created, the method is not renamed.

5. Remove the interaction in the event handler code. Or remove the entire event handler if it only handles one interaction.  
To delete the event handler, right-click the event handler and choose Delete Member.

#### To manually delete the applicable portions of the listener registration and the call in the adapter/listener class

1. Open the Source window.



[Details on this step](#)

2. Delete the applicable portions of code.

Consult a Java book for more information. In general, these are the edits you need to make:

- In the adapter/listener class, if there is only one call to an event handler, delete the adapter/listener class. If there is more than one call to an event handler, delete the check for the object requesting the handling of the event and the call to the event handler. This is an **if** or **else if** statement; make sure the code you are left with is syntactically correct.
- In the listener registration (marked by the `REGISTER_LISTENERS` comment), delete the `object.addKeyListener` call for that component, if the listener registration is not used by another event tied to that component.



**Obsolete**

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Anytime a button is pushed, or a box is checked, a program action called an [event](#) is generated. To link those events to a corresponding action in your program, you need an [event handler](#) that responds to that event.

One of the most powerful features in Visual Cafe is the ability to specify these high-level interactions between [components](#) and events. The key elements of any interaction are the trigger event and the action. The trigger event is the originator of the action; the interaction determines when to execute the code. You can add conditions to an interaction just like you would add an `if` statement in code. The action element is the meat of the interaction; it specifies what to do when the condition is satisfied.


The Interaction Wizard allows you to graphically build relationships between components: Visual Cafe automatically generates code for the specified relationship.

With Visual Cafe, you use the Interaction Wizard to create the interaction between the visual component and a corresponding event. The Interaction Wizard automatically updates your source code for you.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



An [interaction](#) is created by connecting two [components](#). The most common interaction is one between two components on a [form](#). You may also want to create an interaction between a component and a form.

1. In the Project window, double-click the target form to open it.
2. Click the Interaction tool icon  on the Palette.
3. In the Project window, double-click the component that you want to initiate the action.
4. Define the interaction in the Interaction Wizard. This step is the same as defining an interaction between two components.



[Details on this step](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



One of the most powerful features of Visual Cafe is the ability to quickly build a relationship between two [components](#). In Visual Cafe, this is called an [interaction](#). Interactions are implemented as methods and imply an event notification.

Anytime a button is pushed, or a box is checked, a program action called an [event](#) is generated. To link those events to a corresponding action in your program, you need an [event handler](#) that responds to that event.

Visual Cafe lets you specify these high-level interactions between components and events. As you create interactions by visually connecting components, Visual Cafe automatically generates code for the relationship. So, you can assemble interactive applets and applications without writing 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 on which a defined "action" happens. 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).

An interaction does not have to contain two components. You can create an interaction where the trigger and action component are the same. For example, you can create an interaction on an animation component where a mouse click anywhere within the boundaries of a component starts an animation in that component; another click ends the animation.

In the JDK 1.1 event model, when an interaction is created, code for the event handler, listener registration, and [adapter/listener](#) class is automatically generated. If you want to delete an interaction, you need to delete this code.

Visual Cafe changes the source code in five ways when you create an interaction:

- 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.addtypeListener*) after the REGISTER\_LISTENERS tag. 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.
- Fifth, the interaction specified in the wizard is generated in the event handler.

For example, if you have a button, called NextButton, that displays the next image of a slide show, called VacationSlides, the event handler could look like this and appear toward the end of the Java source file:

```
void NextButton_Action(java.awt.event.ActionEvent event)
{
    // to do:  code goes here.

    //{CONNECTION
    // Go to the SlideShow's next image
    VacationSlides.nextImage();
    //}}
}
```

More 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_Action(event);
    }
}
```

If you then add an interaction between VacationSlides and a Label component, Visual Cafe would generate a new event handler for VacationSlides, and the listener registration code would change:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
NextButton.addActionListener(lSymAction);
VacationSlides.addActionListener(lSymAction);
//}}
```

And the adapter/listener class code would change:

```
class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == NextButton)
            NextButton_Action(event);
        else if (object == VacationSlides)
            VacationSlides_SlideChanged(event);
    }
}
```

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



One of the most powerful features of Visual Cafe is the ability to create [interactions](#) between [components](#). The Interaction Wizard lets you graphically build relationships between components, or between a component and itself (for example, double-clicking an item in a list box could remove the item from the list box and add it to another list box). These relationships specify the actions to take when an [event](#) is triggered on a component. For example, in a slide show, a button click could cause the next image to be displayed. Visual Cafe automatically generates the necessary code for the specified relationship.

You can perform these tasks related to interactions:

[Creating a component interaction](#)

[Changing an existing interaction](#)

[Deleting an interaction](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}




<<OBSOLETE TOPIC>> With the Interaction Wizard, you can create an [interaction](#):

- by connecting two [components](#) on a [form](#) in the Form Designer
- by connecting two components in the Project window
- by connecting two components across the Project window and Form Designer (such as a button in the Form Designer and a dialog component in the Project window)
- by connecting a component to itself
- by connecting a form and a component contained by the form

The components must be within the same project.

### To visually connect components

1. In the Form Designer or Project window, click the Interaction tool icon , then drag a line from the trigger component to the action component within the same project. To connect a component to itself, click the component.

The action component is highlighted to help you identify which component you selected. If a component will not highlight, the interaction could be inappropriate.

After you release the mouse button, the Interaction Wizard appears.



**Tip** Before you release the mouse button, you can press ESCAPE to cancel the interaction, or move the interaction line to another component.

2. In the Interaction Wizard, verify that the component names are correct.

The trigger component should appear in the Start an interaction field title; if it is wrong, you need to start over.

The action component should appear in the Select the item you want to interact with field; if it is wrong, choose another component.

3. In the Start an interaction field, select the [event](#) that activates the interaction.
4. In the Choose what you want to happen field, select an action to invoke when the interaction event occurs.
  - If the action takes no parameters, the interaction is finished.
  - If the action takes parameters, the Next button is enabled. When you click Next, a second window displays allowing you to fill in those parameters.
5. If you see a Next button, click it. In the second window, specify the interaction condition.



**Tip** A variable must be of the type listed for the bottom radio button, A *type* constant or expression, for it to appear. If you define a variable of this type and the variable is in scope, it will appear in the variable list.


6. Click Finish.

In the Java source file, code is generated for the call handler, listener registration, and adapter/listener class.

In the source code, Visual Cafe performs five edits:

- 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.addtypeListener*) after the REGISTER\_LISTENERS tag. 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.
- Fifth, the interaction specified in the wizard is generated in the event handler.

### To connect components within the Interaction Wizard

1. Select a component in the Form Designer or Project window.
2. Choose Object  Add Interaction.

3. Complete the Interaction Wizard as described in the previous procedure.

## Running, compiling, executing

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also



 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can compile and run a project any time during its development cycle. Visual Cafe automatically saves files in the project before running.

- Choose Project  Execute to run the project without the debugger.
- Choose Project  Run in Debugger to run the project and start the debugger.

 See Also

**IMPORTANT** Before running in the debugger, make sure your project options are set to debug. When you are ready to compile your final program, make sure the project options are set to final. See [Specifying whether builds are debug or final](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



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. 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 an argument to java.exe, after switches. 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 the main class to run for an application](#) and [Specifying the name of a native application or DLL](#) for more information.
- An application must have a main class containing a **main** method with this signature:

```
static public void main(String args[])
```

For a bytecode application, if it does not have a method of this format, the application can compile but will not run. For a native application, the application cannot compile or run.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can compile using the SJ command line:

```
sj { { file.java } { @file } { switches } }
```

The braces ( { } ) mean "repeated zero or more times." The only required parameter is a Java source file. For native Win32 applications or DLLs (available with the Professional edition only), 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 it 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 *file* is not found, no text is inserted.

### Switches

SJ takes the following switches.

Switch	Description
-cdb <i>file.cdb</i>	Generate compilation database <i>file.cdb</i> . This database stores dependency information and is needed by the make switch.
-make[:r:w]	Build out-of-date files only by checking dependencies between all files that were passed on the command line. You must supply a <b>cdb</b> file with the cdb switch.
-make:r	Check dependencies on all imported files and rebuild them as needed.
-make:w	Check dependencies on all imported files and warn if they are out-of-date.
-classpath <i>path</i>	Use <i>path</i> instead of the setting of the environment variable CLASSPATH.
-d <i>outputdir</i>	Set the output directory. The default is to put <b>class</b> files with their respective <b>java</b> 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 <i>file</i>	Generate dependencies into a test file, <i>file.dep</i> . You can read it to see what classes your files need to deploy.
-g	Add debug information (including line numbers) to output files so they can be debugged.
-gl	Add line number information to the class files so you can step through the source during debugging. Line numbers are also added when you use the -g option.
-help	Print help on compiler switches.
-j <i>codepage</i>	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. But 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 switch (optimize) 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 imbeds 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.
-xdepend	Generate dependency information to stdout that is compatible with javac.

### Javadoc switches

SJ has the following [Javadoc](#) switches, of which only -javadoc is required:

Switch	Description
-javadoc	Enable Javadoc for writing HTML documentation. This switch is unique to SJ.
-docdir <i>dir</i>	Specify the HTML Documentation output directory. This switch is unique to SJ. The default is the current directory.
-systemdocdir <i>dir</i>	Specify the HTML System Documentation directory. This switch is unique to SJ. The default is the API directory specified in sc.ini.
-noauthor	Omit the Javadoc information specified with the @author tag. This switch is unique to SJ.
-noversion	Omit the Javadoc information specified with the @version tag. This switch is unique to SJ.
-includesystem	Include imported system classes (java.*, sun.*, com.*) in the index, tree, and package files that are generated. This switch is unique to SJ. It generates 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. Visual Cafe generates AllNames.html, which 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 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.
-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 Generate Index is selected. This switch is unique to SJ.
-tree	Generate the class/interface hierarchy. Visual Cafe generates 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 Javadoc directories are not 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. (Default: enabled)
-package	Include only package, protected, and public classes and members.
-private	Include all classes and members.
-nodeprecated	<p>If this option is used without the -javadoc switch, Javadoc comments are skipped during compilation. This means that no warnings are issued if deprecated code is used, no Deprecated attribute is written, and the compiler will be faster and use less memory.</p> <p>If this option is used with -javadoc, no warnings are issued if deprecated code is used and no deprecated information will appear in the HTML documentation.</p>

### Native Win32 switches

Here are the command line options that apply only to native applications and DLLs (available in Professional edition only). Either the **obj** or **link** switch is required; the rest are optional.

Option	Description
-obj	Generate native x86 object files.
-link <i>outputname</i>	Link generated object files into <i>outputname</i> . [exe dll]. This command supplies the same functionality as the obj switch, plus runs the linker.
-main <i>mainclass</i>	Specify the main class for the native

	executable. <i>mainclass</i> is a fully qualified class name. (If no main switch is specified, it will default to <i>outputname</i> .)
-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 <i>package class</i>	Specify classes or packages to be exported from the DLL, so they are accessible from other executables. <i>package class</i> can be either a fully qualified class name or a package specification (for example, java.awt.*).
-g6 <i>file.tdb</i>	Add debug information to output files so that the result can be debugged. Use <i>file.tdb</i> as the debug "type database" file. This is the format required to debug in the Symantec Visual Cafe environment.
-5 -6	Generate Pentium (p5) or Pentium Pro (p6) code.
-profile	Do performance profiling.
-L/ <i>switch</i>	Pass <i>/switch</i> to the linker. To see the valid options, type <b>link /?</b> .
<i>file.lib</i>	Link in the library file. Used with the link switch.
<i>file.res</i>	Link in the resource file. Used with the link switch.
<i>file.def</i>	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 file SC.INI:

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 directory where SJ resides.

### How SJ searches for programs

SJ first searches in the directory where SJ.EXE was found. If the programs are not found there, then 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 is important that SJ find the correct imports if you want your build to be correct.

SJ looks in the directory where sj.exe resides for the file SC.INI. SC.INI is a text file, containing environment variable settings in a similar manner as they might be set in AUTOEXEC.BAT:

```
;Comments are lines where the first non-blank character
```

```
; is a ';'
[Environment]
CLASSPATH=C:\CAFE\BIN

;Note that %PATH% gets replaced by the previous value of
; the environment variable PATH.
PATH=C:\CAFE\BIN;%PATH%
```

The special environment variable @P gets replaced with the path to where the SC.INI file resides. For example, the previous lines can be replaced with the following 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 is not there, no error results. This feature is useful for avoiding 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:\CAFE\LIB
```

The system Java documentation directory is defined in SC.INI as follows:

```
API=%@P%\..\JAVA\DOCS
```

The Javadoc generator built into SJ uses this variable to determine the relative links that should be generated and to locate the images directory that contains GIF files used by the documentation.

The Javadoc Viewer search path is defined as follows:

```
JAVADOC_VIEWER_PATH=%@P%\..\JAVA\DOCS;%@P%\..\JFC\DOC\API;%@P%\..\JAVA\DOCS\SYMANTEC
```

## Deployment

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

After you complete an applet in Visual Cafe, you are ready to deploy it on a Web site. You need to know how your particular Web site is set up to get your applet up and running. However, here are some guidelines. 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.

 See Also

**IMPORTANT** The classes needed by Visual Cafe components are stored in `symbeans.jar` and `sfc.jar`. You do not want to deploy using these JARs, because it will affect the 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. If you want to deploy JFC Swing applets, you must provide `swingall.jar`; Visual Cafe deployment tools do not add Swing classes to a JAR.

1. Use the Project  Deploy command to create an archive containing your applets and all supporting files, including Symantec **class** files.

 See Also

[Details on this step](#)

 See Also

**Tip** Your applets should use relative URLs for graphics files so you can easily move your applets to different computers.

2. If you used a JAR, add the variable `ARCHIVE="name.jar"` to the applet tags in your HTML files. You can specify multiple JAR files by delimiting them with a comma (,).

 See Also

**Note** Not all browsers support multiple JARs. Remember that the applet tags specify where your applets are relative to the location of the HTML files. So you must place the JAR 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 JAR file and HTML files in a directory on the Web server, as they appeared in your Visual Cafe project directory.
5. After completing the setup of your Web site, test your Web pages from remote computers.

You can also test it with different operating systems and Web browsers.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

After you complete an application in Visual Cafe, you are ready to deploy it. Requirements for different applications vary. However, here are some guidelines.

#### To deploy your application in a JAR

1. Use the Project  Deploy command to create an archive containing your application and all supporting files, including Symantec **class** files.

 See Also

[Details on this step](#)

 See Also

**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 of the Visual Cafe environment.

To start your application, the archive must be in the class path. For example, you could type at a DOS prompt:

```
set classpath=%classpath%;name.jar
```

Then you can run your application:

```
java application-name
```

**java** invokes java.exe and, if needed, 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; it 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.

 See Also

**Note** Your users need to obtain or you need to provide the [Java virtual machine](#) and standard Java class files. The Symantec Java virtual machine must be licensed from Symantec.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can learn what class files you need by creating a [JAR file](#) with the Visual Cafe JAR utility or by using SJ.

Remember that when you deploy you need to keep the class directory structure intact and you must use the same case in the class names. (Class names are case-sensitive.)

#### **To use the JAR command to get the class files your Java program needs**

When you create a JAR file, Visual Cafe adds the class files your Java programs require. You can use the JAR file to deliver your Java programs, or expand the JAR file with jar.exe to extract the class files from the JAR file. For more information, see [Expanding a JAR file](#).

#### **To use SJ to determine what class files your Java program needs**

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.dep* mainclass.java**

*mainclass.java* is the name of the main class Java file in your project.

*listname.dep* is the file where the list of classes used by your applet or application will be generated. sj.exe also lists the name of the ZIP file (if any) where the class file was found. This helps you figure out what class files you would need to unzip from each class zip file used by your applet or application. The filename must have a .dep extension.

Even though the standard java.\* class files are logged in this file, you are not actually required to copy those to your Web server for your applets, as they are usually 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 command line](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The following instructions for system administrators are guidelines only; they have been tested with Apache. After this is set up, all applets in the user's home HTML directory would have access to the Visual Cafe classes.

 See Also

**IMPORTANT** The classes needed by Visual Cafe components are stored in symbeans.jar. You do not want to deploy using symbeans.jar, because it will affect the performance of your applets when it is downloaded. Instead, you want to deploy using just the classes needed by the components in your applets.

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.

 See Also

[Details on this step](#)

3. Create a symbolic link from /home/symantecclasses/symantec to the user's home HTML directory. For example:


**In -s /home/symantecclasses/symantec /home/joeuser/public\_html**

joeuser can now run applets from Web pages without the need to copy the full Symantec class structure to his public\_html directory.

OBSOLETE

## To deploy your applets outside of a JAR

Here is one way:

1. <<gone>> Use the Project  JAR command to create a [JAR file](#) containing your applets and all supporting files, including Symantec **class** files.

 See Also

[Details on this step](#)

 See Also

**Tip** Your applets should use relative URLs for graphics files so you can easily move your applets to different computers.

2. Create a new directory for your deployment files.
3. Expand the JAR file into the new directory.

 See Also

[Details on this step](#)

 See Also

**Note** If you want to put the class files your applets use into a JAR file, create the JAR file then add the variable `ARCHIVE="name.jar"` to the applet tags in your HTML files. You can specify multiple JAR files by delimiting them with a comma (.). Not all browsers support multiple JARs.

4. Copy the HTML files for your Web pages into the new directory.

 See Also

**Tip** Remember that the applet tags specify where your applets are relative to the location of the HTML files.

5. On your local computer, test your Web pages by opening them in a Web browser from outside of the Visual Cafe environment.
6. Put your files in a directory on the Web server, as they appeared in the directory on your local computer.
7. After completing the setup of your Web site, test your Web pages from remote computers.

You can also test it with different operating systems and Web browsers.

Here is another way:

1. Create a deployment directory on your local computer.
2. Put all of your applet class files, HTML files, and other supporting files (such as graphics files) in the deployment directory, as they appeared in your Visual Cafe project directory.

 See Also

**Tip** Your applets should use relative URLs for graphics files so you can easily move your applets to different computers.

3. Enable your applet to access the class files it needs, including the Symantec class files. (Remember that the Web browser should provide access to standard Java class files.)

 See Also

[Details on this step](#)

If you want to put the class files your applets use into a JAR file, create the JAR file then add the variable `ARCHIVE="name.jar"` to the applet tags in your HTML files. You can specify multiple JAR files by delimiting them with a comma (,), but most browsers do not consistently support this.


If the class files are not in a JAR file, you can place the files in the deployment directory, preserving the directory structure and case of the class names

4. On your local computer, test your Web pages by opening them in a Web browser from outside of the Visual Cafe environment.
5. Put your files in a directory on the Web server, as they appeared in the directory on your local computer.
6. After completing the setup of your Web site, test your Web pages from remote computers.

You can also test it with different operating systems and Web browsers.

## To deploy your application outside of a JAR <<gone>>

Here is one way:

1. Use the Project  JAR command to create a [JAR file](#) containing your application and all supporting files, including Symantec **class** files.

 See Also

[Details on this step](#)

 See Also

**Tip** Your application should use relative URLs for graphics files so you can easily move your application to different computers.

2. Create a new directory for your deployment files.
3. Expand the JAR file into the deployment directory.



[Details on this step](#)



**Note** You can put the class files your application uses into a JAR file, but remember that the JAR file must be added to the class path on the computer the application runs on.

4. On your local computer, test your application by running it from outside of the Visual Cafe environment.

To run your application:

**java** *application-name*

**java** invokes java.exe and, if needed, 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; it is case-sensitive and you might need to include the complete path.

For example:

**\visualcafe\java\bin\java AmazingTour**

5. Test your application from remote computers.

You can also test it with different operating systems.



**Note** Your users need to obtain or you need to provide the [Java virtual machine](#) and standard Java class files. The Symantec Java virtual machine must be licensed from Symantec.

Here is another way:

1. Create a deployment directory on your local computer.
2. Put all of your application class files and other supporting files (such as graphics files) in the deployment directory, as they appeared in your Visual Cafe project directory.



**Tip** Your application should use relative URLs for graphics files so you can easily move your application to different computers.

3. Enable your application to access the class files it needs, including the Symantec class files. (Remember that the [Java virtual machine](#) should come with the standard Java class files.)



[Details on this step](#)

You can put the class files your application uses into a JAR file, but remember that the JAR file must be added to the class path on the computer the application runs on.

If the class files are not in a JAR file, you can place the files in the deployment directory, preserving the directory structure and case of the class names

4. On your local computer, test your application by running it from outside of the Visual Cafe environment.

To run your application:

**java** *application-name*

**java** invokes java.exe and, if needed, 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; it is case-sensitive and you might need to include the complete path.

For example:

**\visualcafe\java\bin\java AmazingTour**

5. Test your application from remote computers.

You can also test it with different operating systems.



**Note** Your users need to obtain or you need to provide the [Java virtual machine](#) and standard Java class files. The Symantec Java virtual machine must be licensed from Symantec.

## HTML and VC

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



[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.

Visual Cafe lets you run and debug applets within its environment without supplying an HTML file. However, to test your applet within an HTML page, you can specify an HTML file in which to run and debug the applet.

You can run and debug your applet in the Applet Viewer or a Web browser. A Web browser lets you view the HTML page and applet at the same time.

You can add the HTML files to a project as a useful organizational tool, but it is not required. Once an HTML file is added to the project, you can view it in the Object and Files view, open it quickly from the Project window, and see it in the project options.

If Visual Page is on your computer and the software has been set up properly, double-clicking an HTML file in the Project window (Objects view) launches Visual Page and visually displays the HTML file in Visual Page window. You can also view and edit an HTML file directly in the Source window of Visual Cafe; for example, you can double-click the HTML files in the Project window (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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



HTML stands for HyperText Markup Language. A markup language is a system of tags that preserve a document's structure so it can be disassembled, moved electronically, and reassembled when it reaches its final destination. The "Markup" part of HTML refers to tags that affect the way information in a document will be displayed. By using different HTML tags, you control text editing, formatting, and how graphics and animations are displayed.

Java is a programming language that allows you to design, build, and execute [applications](#) and [applets](#). Before they can be used, Java programs must be coded, compiled, and debugged similar to development with any other computer programming language.

Java is an [interpreted](#), object-oriented programming language with a syntax and structure similar to C++; it was designed specifically for the [Internet](#) by Sun. One of the fundamental features about the Java programming language is that it is [platform-independent](#). Java applets and applications will run on any computer platform that has Java installed on it.

An HTML document can contain links to Java programs so that you can display a Java program next to regular text in a [Web page](#). Because HTML now supports Java, users get the best of both worlds. Visual Cafe can provide the necessary HTML you need to run and view your applet within Visual Cafe. You can also create and edit HTML in Visual Cafe. However, for features specifically designed to assist you in writing HTML documents, you can use Symantec Visual Page, a visual HTML editing application.

#### 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.

For the Web browser to be able to display an applet, it requires some basic information, provided through the use of the [applet tag](#). Within the applet tag you specify where to find the **class** file, and how large to make the display space within 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](#). For the most up-to-date information on applet tag attributes, see [www.javasoft.com](http://www.javasoft.com), such as in the JDK documentation.

Using the applet tag, you specify (at a minimum) the location of the Applet subclass and the dimensions of the applet's onscreen display area. When a Java-capable browser encounters an applet tag, it reserves onscreen space for the applet, loads the Applet subclass onto the computer the browser is executing on, and creates an instance of the Applet subclass. Next, the browser initializes the applet, and the applet is off and running.

The following listing shows the applet tag that includes the "Hello World" applet 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>` specifies that the browser 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 are like the same attributes in 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 tags. You can specify multiple JAR files by delimiting them with a comma (,). You can use `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 cannot 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 applets](#).

### Passing parameters to applets from an HTML file

It is 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. It is necessary to 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If HTML files are included in your project, you can view the file and edit its HTML code from within Visual Cafe, and if installed, Visual Page.


1. Open the file by using one of these methods:
  - In the Project window (Object or Files view), double-click the HTML file name.
  - In the Project window (Object or Files view), right-click and choose Edit Source.
  - Choose File  Open, then select the file and click Open.

The HTML file appears in the Source window. If you have installed Visual Page from the Visual Cafe software package, opening a file from 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 make your edits in the Source window or Visual Page.

 See Also

[Details on using the Source window](#)

3. Save the file by choosing File  Save.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can add an applet to an HTML page by:

- Adding an applet tag to the HTML code
- Dragging an 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 width.

*pixh* is the number of pixels for the 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:

*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.

Consult an HTML book for more information on the applet tag.

## Source code

 See Also

 See Also

 See Also


 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe provides the Source Editor (available in the Source window and pane), Class Browser, and Hierarchy Editor to assist you in viewing and editing your Java source code and class hierarchy.

### Working with the Source Editor

The Visual Cafe Source Editor helps you create, examine, and modify your project source files. It is also designed to work with other Visual Cafe tools. You can access the Source Editor from the Source window, or from the Source pane of the Class Browser.

The Visual Cafe Source window is where you can create, display, and edit code. It is a full-featured text editor with many options and features particularly useful when editing Java source code. It provides for full Java [syntax](#) highlighting and flexible navigation tools. 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.

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 text box.

The Visual Cafe Source window and the Class Browser Source pane both share the same editing functionality. The editor uses standard Windows editing commands and has special features that make working with Java files easier. For example, the editor can automatically indent or unindent after braces and can check delimiters.

In addition, the Source Editor can display keywords and comments in special font styles and colors. This technique helps track errors in source code while you are 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 are in a different color or font style. Misspelled keywords can be caught immediately when they remain displayed in the default font.

Source windows are an integral part of the Visual Cafe environment and work together with other Visual Cafe windows to make application development easier. For example, 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 an error message, Visual Cafe opens a Source window on the corresponding source file, if necessary, and then jumps to the line in the source code that caused the error.

The Source Editor supplies standard Windows functions for cutting, copying, pasting, and deleting text. These functions can be accessed through either the Edit menu or the pop-up menu.



[Click here for guidelines on adding custom code](#)

### Working with the Class Browser

[Classes](#) are the foundation of object-oriented programming. To make working with classes easier, Visual Cafe provides this powerful tool for working with classes.

The Class Browser is a three-pane window that lists all of 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 are free from the clutter of other member implementations in the same source file. The Class Browser helps you work with your source code in an organized way. 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 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](#) and [methods](#) for each class and an edit area for working directly with the body of a method.

There are three panes in the Class Browser: the Classes, Members, and Source panes. Press Shift then + or - to expand all or collapse all entries in the tree view.

#### The Classes pane

The 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 have not created a package structure yet. In this case, the default package represents the entire project.

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

### The Members pane

The data and methods in a class are called members of that class. The Members pane of the Class Editor gives you an organized look at the data and methods the selected class is made of. The color of the icon indicates level of access for a class. A yellow icon indicates protected, green indicates it is a public class, and blue indicates it is a package.

Clicking a method or variable displays the source code for that member in the Source pane. You can drag and drop a member into the source pane; the member declaration is inserted into the source.

Right-clicking in the Members pane allows you to add new members to the class. You are 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.

### The Source pane

The Source pane displays the source code for the member data or method that is 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.

### Working with the Hierarchy Editor

The Hierarchy Editor is a visual tool that provides you with a visual representation of the classes in your project, and their [inheritance](#) relationships.

All Java programs have a hierarchical structure. You can use the Hierarchy Editor to directly manipulate the class relationships of your projects, by simply dragging and dropping from one class to another.



**Note** Any changes made to classes or inheritance relationships in the Hierarchy Editor are automatically changed in the underlying source code. For that reason, you should be careful when making changes with the Hierarchy Editor. Visual Cafe does not allow you to change the structure of the existing Java hierarchy, such as for the JDK and JFC.

You can extend existing classes just 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 Editor so you can view its data and methods.

## Source Editor

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

The Source window is the primary window for viewing, writing, and editing source code and for debugging. This window allows you to create, display, and edit Java and HTML code, and text files. Navigation aids help you move through your code and full Java syntax highlighting makes it easy to find code areas.






The Object field displays the name of the current active object. You can use the drop-down list to select other objects in the source file.

In the Events/Methods drop-down box, 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.

Keywords, comments, and Java commands are colored for easy editing. By default, comments are dark green, Java language keywords are blue, and standard code text is black.



{button ,JI('VCAFE.HLP','Adding\_Code\_to\_a\_Java\_Source\_File\_howto')}} [Click here for coding guidelines](#)

#### To display the Source window

- Select a file in the Project window, then choose Object  Edit Source, or right-click and choose Edit Source.
- Double-click a file in the Packages or Files view of the Project window. (If 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 Object  Edit Source, or right-click and choose Edit Source.
- In the Classes pane of the Class Browser, select a class then choose Classes  Go to Source, or right-click a class and choose Go to Source.
- In the Hierarchy Editor, select a class then choose Hierarchy  Go to Source, or right-click a class and choose Go to Source.
- Open a file by choosing File  Open.

#### To cut, copy, or paste source code

You can copy and move text within the same window, or between Source Editor windows, including the Source window and the Class Browser Source pane.

- Select code, then choose Edit  Cut or Copy, or right-click and choose Cut or Copy.
- Position the cursor where you want to paste, then choose Edit  Paste, or right-click and choose Paste.
- Drag a block of selected text to move it.
- Press CONTROL while dragging a block of selected text to copy the text. (A + appears over the cursor when a copy operation is being performed.)

#### To create or go to an event or method

1. In the Objects drop-down list of the Source window, 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 is not bold, it is created for you.

 See Also


**Note** If you choose an event, Visual Cafe adds the [event handler](#) and other code. See [Adding an event handler to a component](#) for more information.

#### To go to a definition

1. Select or click in a class or member, then choose Search  Go to Definition, 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.

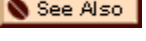
 [Details on using the Class Browser](#)


#### To search or replace in a file

- While the Source window is active, choose a command from the Search menu.

 [Details on this task](#)

#### To search in multiple files

- While the Source window is active, choose Search  Find in Files.


 [Details on this task](#)


#### To get help on a Java keyword or method

- Position the cursor on a keyword or method, then press F1 to display a list of associated topics in the Java Reference help.
- To add help files, see [Defining the help file set](#).


#### To set format options for all Source windows

- Choose Tools  Environment Options

 Format and select the options you want.

 [Details on this step](#)


#### To set format options for the current file

- While the file is displayed in the Source window, choose Source  Format Options, or right-click and choose Format Options. Complete the dialog box.

 [Details on this step](#)

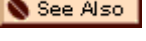
#### To set editing options for all Source windows


- Choose Tools  Environment Options

 Edit and select the options you want.

 [Details on this step](#)

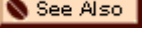
#### To set edit-key assignments for all Source windows


- Choose Tools  Environment Options


 Keyboard and set the options you want.

 [Details on this step](#)


#### To set display options for all Source windows

- To change the source code text display, such as colors for comments, choose Tools  Environment Options

 Display. The Editing tab also has options that apply to the Source window.

 [Details on this step](#)

#### To indent or unindent code

- Select the lines of code, then choose Source  Indent or Unindent.  
The code is moved one tab position. You can also press Tab or Shift-Tab.

#### To convert tabs to spaces or spaces to tabs

- Select the code, then choose Source  Tabs to Spaces or Spaces to Tabs.

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 (which were described previously).

#### To change text to all uppercase or lowercase letters

- Select the code, then choose Source  Uppercase or Lowercase.

#### To set a breakpoint for debugging


- Select a line of code, then choose Source  Set Breakpoint, or right-click and choose Set Breakpoint.

#### To set a conditional breakpoint for debugging


- Select a line of code, then choose Source  Set Conditional Breakpoint and complete the dialog box.

 [Details on this step](#)

#### To clear a breakpoint

- Select a line of code with a breakpoint, then choose Source  Clear Breakpoint, or right-click and choose Clear Breakpoint.

#### To evaluate an expression during debugging

- Select an expression, then choose Source  Evaluate Expression, or right-click and choose Evaluate Expression. Complete the dialog box.

 [Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Source  
Source

 See Also

Format Options

 See Also

Go to Buffer

 See Also

Options

The options in this window allow you to define a format style for the current editing window/buffer. Opens are active until the associated source window/buffer closes.

1. Set the standard format options.

The formatting style options on this dialog box are the same as on the Environment Options, Format Tab. See [Specifying text formatting for Visual Cafe windows](#) for complete option information.

2. Set the remaining options as needed.

Option	Description
Use as Default for .java	Assign the current settings as the default setting.
Read only	The buffer can not be changed.
Keep file in memory	No prompts are given for saving on close or changes. Permanently stores file in memory.

3. Click OK.

Class Browser

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

The Class Browser is a three-pane window that lists the Java classes in your project and the methods and data members contained within each class.

The Class Browser allows you to quickly navigate, edit, and add classes. You only see 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 do not 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).

Both the Classes and Members panes support keyboard incremental searches. As you type the name of a class or member, the list of matching items is refined until the class or member you want is automatically selected.

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


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

The Source pane provides the same editing features as the Source window.

 See Also

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

#### To display the Class Browser

- While the project you want to view is active, choose View  Class Browser.

#### To expand or collapse all nodes in the tree view

- Press Shift then + or - to expand all or collapse all entries in the tree view.

#### To locate a class in the Classes pane

- 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, as described next.

#### To change what classes are displayed in the Classes pane

1. In the Classes pane, choose Classes  Options, or right-click and choose Options.
2. In the Class Options dialog box, set the options you want:
  - In the Group/Sort tab, specify whether you want to group classes alphabetically, hierarchically, or by package.
  - In the Filter tab, specify where you want to show imported classes.

#### To create a new class with the Class Wizard

1. In the Classes pane, optionally select a class you want the new class to extend.
2. Right-click in the Classes pane and choose Insert Class.

By the default in the Environment Options, you can also press INSERT.


The Insert Class Wizard appears.

3. Complete the Insert Class Wizard.

 [Details on this step](#)

The new class appears in the Class Browser display and is added to the project.

#### To edit an existing class with the Class Wizard

1. In the Classes pane, select a class you want view or edit.
2. Choose Classes  Edit Class, or right-click in the Classes pane and choose Edit Class.


The Edit Class Wizard appears.


3. Complete the Edit Class Wizard.

 [Details on this step](#)


#### To view or edit the source code for a class

You can view source code in the Source pane of the Class Browser or in a Source window. The Source pane has the same editing options as the Source window.

- To display source code in the Source pane, select a class in the Classes pane then a member in the Members pane.
- To display source code in a Source window, from the Classes pane, select a class then choose Classes  Go to Source, or right-click a class and choose Go to Source.

 [Details on using the Source window](#)

#### To delete a class inheritance

1. In the Classes pane, choose Classes  Options, or right-click and choose Options.
2. In the Group/Sort tab, specify a Class Grouping of Hierarchically, then click OK.

When you are showing classes by 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 is 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 delete a class

- In the Classes pane, select a class, then press DELETE.

#### To locate a method or data variable in the Members pane

1. Select a class in the Classes pane.



**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed, as described previously.

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 are 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 pane, you might need to change what members are displayed, as described next.

#### To change what members are displayed in the Members pane

1. In the Members pane, right-click and choose Options. Or while the Class Browser is the active window, choose Classes

 Options.

2. In the Class Options dialog box, set the options you want:
  - In the Group/Sort tab, specify whether you want to sort members by access (public, private, protected, and package),

alphabetically, or not at all.

- In the Group/Sort tab, specify whether you want to group members by kind or by access.
- In the Filter tab, specify what members you want to show (choose from public, private, protected, package, final, static, and regular) and if you want to display their types.
- In the Inheritance tab, specify whether you want to show inherited methods, and if so, if you want to display full method names or show overridden methods.

#### To create a new member from the Members pane

1. In the Class pane, select a class you want to add a method or data variable to.



**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed, as described previously.

2. Right-click in the Members pane and choose Insert Member. Or while the Class Browser is the active window, choose Insert Member.

By default in the Environment Options, you can also press INSERT.

The Insert Member dialog box appears.

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.

#### To view attributes of a member from the Members pane

1. In the Class pane, select a class that contains the member.



**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed, as described previously.

2. Select the member in the Members pane.



**Note** If the method or data variable you want is not displayed in the pane, you might need to change what members are displayed, as described previously.

3. Right-click in the Members pane and choose Member Attributes. Or while the Class Browser is the active window, choose Classes Member Attributes.

The Member Attributes dialog box displays.

4. Change the access type if needed, then click OK.

#### To delete a member from the Members pane

1. In the Class pane, select a class the contains the member.



**Note** If the class you want is not displayed in the pane, you might need to change what classes are displayed, as described previously.

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 what members are displayed, as described previously.


3. Right-click in the Members pane and choose Delete Member. Or while the Class Browser is the active window, choose Classes Delete Member.

The member is deleted from the class.

#### To view or edit the source code of a member

You can view source code in the Source pane of the Class Browser or in a Source window. The Source pane has the same editing options as the Source window.

- To display source code in the Source pane, select a class in the Classes pane then a member in the Members pane.

- To display source code in a Source window, from the Classes pane, select a class then choose **Classes**  **Go to Source**, or right-click a class and choose **Go to Source**.



[Details on using the Source window](#)

#### **To rename a class or member**

1. Select the class or member in the Classes or Members pane. Then click it again.
2. When an edit box appears, type the new name or edit the existing name.

The constructor is renamed for you.

#### **To add a class or member by dragging it into source code**

- Select the class or member in the Classes or Members pane, then drag it into the Source pane or a Source window.  
The full class name, full method signature, or data variable is added at the location you drop it.

#### **To collapse all packages**

- Press Shift-.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

**Classes**  See Also **Member Attributes**

Use this window to change the [access type](#) of selected members.

The class declaration is modified and the Members display is updated to reflect the change.

If you are 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.

## Hierarchy Editor

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

The Hierarchy Editor shows a visual representation of the classes in the inheritance hierarchy. It shows the classes that the objects in your project inherit from; you can optionally show imports as well.

From the Hierarchy Editor, you can do the following:


- Change the inheritance hierarchy by selecting an inheritance line and dragging it to a different class.
- Remove an inheritance.
- Create a new class or edit an existing class with the Class Wizard.
- Select a class and view its source in a Source window or the Class Browser.
- Locate a class in the display by typing its name.

Changes made to classes or inheritance relationships are automatically changed in the underlying source code and in all open windows in the Visual Cafe environment.


 See Also

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

#### To display the Hierarchy Editor

- While the project you want to view is active, choose View  Hierarchy Editor.

#### To enable and disable viewing imports

- While the Hierarchy Editor is the active window, choose Hierarchy  View Imports or right-click and choose View Imports to toggle the display.

#### To locate a class in the display

- 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.

#### 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 its source code.

#### To remove an inheritance

- Select the line between a parent and base class, then right-click and choose Remove Inheritance.

The class now extends directly from `java.lang.Object`.

#### To create a new class


1. Optionally select a class you want the new class to extend.
2. Drag from the class into the background space, or right-click and choose Insert Class.  
The Insert Class Wizard appears.
3. Complete the Insert Class Wizard.

 See Also

[Details on this step](#)

The new class appears in the Hierarchy Editor display and is added to the project.

#### To edit an existing class

1. Select a class you want view or edit.
2. Right-click and choose Edit Class. Or choose Hierarchy  Edit Class.


The Edit Class Wizard appears.

3. Complete the Edit Class Wizard.



[Details on this step](#)

**To view a class in a Source window**

- Select a class then choose Hierarchy  Go to Source, or right-click a class and choose Go to Source.

**To view a class in the Class Browser**

- Double-click a class in the Hierarchy Editor.

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```

 See Also

 See Also

Visual Cafe can locate **class** files placed in a **zip** file. The files must be on your class path, and you must preserve the directory structure of the packages.

## Adding Code to a Java Source File

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

Although you can create some applets with Visual Cafe that do not require any custom Java code, you might need to add code for advanced applets and for applications. For example, you can add [custom code](#) for error handling, event control, and complex [component](#) relationships and behavior. You can bind [event handlers](#) to any component and to menu commands.

You can use the Visual Cafe visual environment to quickly create your form layouts. If you do not want to use the visual environment, or you are 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 file or for all new files that are added to a project. This RAD on/off feature is available in Visual Cafe Professional edition. See [Enabling and disabling RAD and automatic code generation](#) for more information.

If you are using the visual environment of Visual Cafe, before you add custom code, you should create your project and use Visual Cafe's powerful visual tools to create your [forms](#). 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 a component and itself with the Interaction Wizard. It is a good idea to do as much as you can with the Visual Cafe tools before you add your custom code.

When you are satisfied with the layout of your project forms, you can then add any necessary code enhancements using the Source window or Class Browser.

Visual Cafe places special comments in the source code to indicate the beginning and end of blocks of code that it needs to manage. 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.setBounds(107,50,61,56);
add(button1);
//}}
```

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.

Avoid adding to or modifying custom code within code blocks that are regenerated (between the `//{{` and `//}}` comment tags). To modify the code within these tags, try to use code syntax that matches what Visual Cafe can generate. Realize that in some cases Visual Cafe might be unable to back parse your Java file into its visual environment.

Also avoid moving these code blocks. But if you do, be sure to move the entire code block, including the special comment tags.

In general, when you make a manual coding change, you need to make sure that all areas of the source file are jointly updated. Some changes you make in one part of the file require an additional modification in another part of the code.

Whenever possible, make object changes using Visual Cafe, versus adding code directly to the source file. For example, adding components, classes or class members, and changing component properties can be done through the Component Library or Palette, using a menu selection, or making a change in the Property List.

 See Also

**Warning** [Custom code](#) is not deleted from the source file when you delete a component. You must manually maintain any file that contains custom code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

 See Also

Visual Cafe uses a certain code format so it can create and manage code resulting from changes made in its visual environment. Understanding these coding guidelines will help you achieve the best results when adding a file that was created outside of the Visual Cafe environment, and when you modify the source code generated by Visual Cafe.

 See Also

**Tip** If you do not want to use the visual environment, you can turn off the visual environment for a file or for all new files that are added to a project. See [Enabling and disabling RAD and automatic code generation](#) for more information.

When Visual Cafe automatically generates code, it places coding fences (known as *tags*) in the source. Visual Cafe uses these tags to clearly define what code should be processed by the visual environment and what code should not. Here are some tags and their uses:

Tag	Description
<pre>//{{INIT_CONTROLS  code  //}}</pre>	Delineates the initialization of components. The init controls tag is located in the first constructor or init method found by parsing from the top of the class. If this tag does not appear, it may be recreated when needed. If this tag appears multiple times, the results can vary.
<pre>//{{DECLARE_CONTROLS  code  //}}</pre>	Delineates the instances of components. The declare controls tag is located in the scope of the class outside of any methods or inner classes.

When a new component is dropped onto a form, both the component instance and the initialization of that instance are generated into the source code. For example, when dropping a button onto the form, a component is added into the declare controls tag as follows:

```
java.awt.Button button1 = new java.awt.Button();
```

And the following code is added to the init controls tag:

```
button1.setLabel("button");  
add(button1);  
button1.setBackground(java.awt.Color.lightGray);  
button1.setBounds(71,25,111,49);
```

There are also coding conventions for events. See [Understanding component interactions](#) for more information.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



Every [component](#) has a set of events that users can trigger at runtime. To bind code to an event, you create an [event handler](#) in the source file and add [custom code](#). When the event occurs, the code in your event handler executes.

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.addtypeListener* or *object.addtypeAdapter*) 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, plus the interaction specified in the wizard is generated in the event handler. See [Using the Interaction Wizard](#) for more information.

To add an event handler from the Source window:

1. Open the Source window for the component by using one of these methods:
  - Select the component in the Project window, then choose Object Edit Source, or right-click and choose Edit Source.
  - While the form with the component is displayed in the Form Designer (which is the active window), choose Object Edit Source, or right-click and choose Edit Source.
2. In the Source window, choose the component from the Objects drop-down list.
3. In the Events/Method drop-down list of the Source window, select the event that you want to bind to.

The event handler is added to the source file. The cursor insertion point is positioned inside the event handler. Code is also added to the proper listener so that when an event occurs in the object the event handler gets executed.
4. In the event handler, replace the placeholder text `// to do: code goes here` with appropriate Java code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can configure buttons to accept keyboard input, such as CTRL+C. When certain keys are pressed, it would be the same as clicking the button; these are sometimes called *hot keys*. Here is a code sample:

```
/*
This simple extension of the java.awt.Frame class
contains all the elements necessary to act as the
main window of an application.
*/

import java.awt.*;

public class Frame1 extends Frame
{
    public Frame1()
    {
        // This code is automatically generated by Visual Cafe when you add
        // components to the visual environment. It instantiates and initializes
        // the components. To modify the code, only use code syntax that matches
        // what Visual Cafe can generate, or Visual Cafe may be unable to back
        // parse your Java file into its visual environment.

        //{{INIT_CONTROLS
        setLayout(null);
        setVisible(false);
        setSize(insets().left + insets().right + 405, insets().top + insets().bottom + 305);
        openFileDialog1 = new java.awt.FileDialog(this);
        openFileDialog1.setMode(FileDialog.LOAD);
        openFileDialog1.setTitle("Open");
        //$ $ openFileDialog1.move(36,276);
        setTitle("An Application");
        //}}

        //{{INIT_MENUS
        mainMenuBar = new java.awt.MenuBar();

        menu1 = new java.awt.Menu("File");
        miNew = new java.awt.MenuItem("New"); menu1.add(miNew);
        miOpen = new java.awt.MenuItem("Open..."); menu1.add(miOpen);
        miSave = new java.awt.MenuItem("Save"); menu1.add(miSave);
        miSaveAs = new java.awt.MenuItem("Save As..."); menu1.add(miSaveAs);
        menu1.addSeparator();
        miExit = new java.awt.MenuItem("Exit"); menu1.add(miExit);
        mainMenuBar.add(menu1);

        menu2 = new java.awt.Menu("Edit");
        miCut = new java.awt.MenuItem("Cut"); menu2.add(miCut);
        miCopy = new java.awt.MenuItem("Copy"); menu2.add(miCopy);
        miPaste = new java.awt.MenuItem("Paste"); menu2.add(miPaste);
        mainMenuBar.add(menu2);

        menu3 = new java.awt.Menu("Help");
        mainMenuBar.setHelpMenu(menu3);
        miAbout = new java.awt.MenuItem("About.."); menu3.add(miAbout);
        mainMenuBar.add(menu3);
        setMenuBar(mainMenuBar);
        //$ $ mainMenuBar.move(4,277);
        //}}

        //{{REGISTER_LISTENERS
        SymWindow lSymWindow = new SymWindow();
        addWindowListener(lSymWindow);
        SymAction lSymAction = new SymAction();
        miOpen.addActionListener(lSymAction);
        miAbout.addActionListener(lSymAction);
    }
}
```

```

miExit.addActionListener(lSymAction);
SymKey lSymKey = new SymKey();
addKeyListener(lSymKey);
//}}
}

public Frame1(String title)
{
    this();
    setTitle(title);
}

public synchronized void show()
{
    move(50, 50);
    super.show();
}

static public void main(String args[]) {
    (new Frame1()).show();
}

public void addNotify()
{
    Dimension d = etSize();

    super.addNotify();

    if (fComponentsAdjusted)
        return;

    // Adjust components according to the insets
    setSize(insets().left + insets().right + d.width, insets().top + insets().bottom + d.height);
    Component components[] = getComponents();
    for (int i = 0; i < components.length; i++) {
        Point p = components[i].getLocation();
        p.translate(insets().left, insets().top);
        components[i].setLocation(p);
    }
    fComponentsAdjusted = true;
}
//Used for addNotify check.
boolean fComponentsAdjusted = false;

//{{DECLARE_CONTROLS
java.awt.FileDialog openFileDialog1; //}}

//{{DECLARE_MENUS
java.awt.MenuBar mainMenuBar;
java.awt.Menu menu1;
java.awt.MenuItem miNew;
java.awt.MenuItem miOpen;
java.awt.MenuItem miSave;
java.awt.MenuItem miSaveAs;
java.awt.MenuItem miExit;
java.awt.Menu menu2;
java.awt.MenuItem miCut;
java.awt.MenuItem miCopy;
java.awt.MenuItem miPaste;
java.awt.Menu menu3;
java.awt.MenuItem miAbout;
//}}

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent event)
    {
        Object object = event.getSource();
        if (object == Frame1.this)
            Frame1_WindowClosing(event);
    }
}

```

```

}

void Frame1_WindowClosing(java.awt.event.WindowEvent event)
{
    hide();          // hide the Frame
    dispose();       // free the system resources
    System.exit(0);  // close the application
}

class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == miOpen)
            miOpen_Action(event);
        else if (object == miAbout)
            miAbout_Action(event);
        else if (object == miExit)
            miExit_Action(event);
    }
}

void miAbout_Action(java.awt.event.ActionEvent event)
{
    //{{CONNECTION
    // Action from About Create and show as modal
    (new AboutDialog(this, true)).show();
    //}}
}

void miExit_Action(java.awt.event.ActionEvent event)
{
    //{{CONNECTION
    // Action from Exit Create and show as modal
    (new QuitDialog(this, true)).show();
    //}}
}

void miOpen_Action(java.awt.event.ActionEvent event)
{
    //{{CONNECTION
    // Action from Open... Show the OpenFileDialog
    openFileDialog1.show();
    //}}
}

class SymKey extends java.awt.event.KeyAdapter
{
    public void keyPressed(java.awt.event.KeyEvent event)
    {
        Object object = event.getSource();
        if (object == Frame1.this)
            Frame1_KeyPress(event);
    }
}

void Frame1_KeyPress(java.awt.event.KeyEvent event)
{
    // to do: code goes here.
    System.out.println("Button Key Press :" + event.getKeyText(event.getKeyCode()));
}
}

```

## Searching in a File

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


The Source Editor has 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 provides a powerful means of locating a string in any set of files.

You can also jump to specific points in a file by using commands from the Search menu.

#### To find a string

1. Optionally select text in the Source window.

The selected text becomes the "Find what" criteria.

2. While the Source window is active, choose Search  Find.
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 any options you want.

By checking Regular Expression in the Find dialog box, you can use the wildcard characters in [regular expression syntax](#).

Regular expressions permit more powerful text searches. The  button lists all available regular expression syntax. When you select from the list, the regular expression option is automatically checked.

4. Click Next to search forward or Previous to search backward.


#### To repeat the search in the same direction

- Choose Search  Find Again.


#### To replace a string with a different string

1. Optionally select text in the Source window.

The selected text becomes the "Find what" criteria.


2. While the Source window is active, choose Search  Replace.
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 substitution string in the Replace field or choose a previous string from the drop-down list.
5. Select any options you want.

If you select Confirm changes, the editor scrolls the display to each occurrence of the search string, selects it, and asks whether you want to replace it by displaying the Confirm Replacement dialog box. If you do not select this option, every occurrence of the string is replaced when you click Replace.

By checking Regular Expression, you can use the wildcard characters in [regular expression syntax](#). Regular expressions permit more powerful text searches. The  button lists all available regular expression syntax. When you select from the list, the regular expression option is automatically checked.


6. Click Replace.

 See Also

**Tip** Use Edit  Undo Search to undo the entire set of replacements of the search string.


#### To jump to a matching delimiter


A common problem in source code is parentheses ( ( ) ), brackets ( [ ] ), and braces ( { } ) that do not match. Visual Cafe lets you search for a matching delimiter.


1. In the Source window, position the insertion point in front of one of the delimiters.
2. Choose Search  Go to Matching Delimiter.

The insertion point moves to the other half of the pair.


 [See Also](#)

**Tip** If you have chosen the Check delimiters option, which can be set for a file by choosing Source  Format Options or globally with Tools


 [See Also](#) Environment Options

 [See Also](#) Format, the Source Editor performs delimiter checking for you automatically, including text in strings and comments. If you type a right parenthesis, right square bracket, or right brace, the editor briefly highlights the corresponding left delimiter. If no matching delimiter is found, an error message is displayed in the status bar of the editor.

#### To jump to a specific line

1. While the Source window is active, choose Search  Go To Line.  
The Go To Line dialog box opens.
2. Type the line number in the text box and click OK.  
The editor moves the insertion point to the beginning of the specified line.


#### To jump to a function

1. Choose Search  Go to Function.  
The Go to Function dialog box opens.
2. Select a function name from the list or type a function name.  
You can change how function names display in the list with the Show member name first option.
3. Click OK.  
The insertion point moves to the beginning of the specified function.

#### To jump to an event or method

1. In the Objects drop-down list of the Source window, 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 is not bold, it is created for you.

#### To go to a buffer

- Choose Search  Go to Buffer and complete the dialog box.

 [See Also](#) [Details on this step](#)

#### To set a bookmark

- Choose Search  Bookmarks and complete the dialog box.

 [See Also](#) [Details on this step](#)

#### To search for a string in multiple 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 filename, directory, date, time, and file attributes

1. Choose Search  Find in Files.
2. Choose what files to search.

 [See Also](#) [Details on this step](#)

3. Set advanced search options as needed.

 [See Also](#) [Details on this step](#)


#### To compare two files

- Choose Tools  Compare Files and complete the dialog box.



[Details on this step](#)

#### **To go to a definition**

1. Select or click in a class or member, then choose Search  Go to Definition, 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.



[Details on using the Class Browser](#)

{ewc\_HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Search Find


Use the Find dialog box to search the current file for a text string.

1. Enter the search pattern in the Find What field.

You can initialize the pattern by selecting text before choosing Find. (The text must not span a line break.) You can also use the drop-down list to select text from previous search strings.

The pattern may also be a [regular expression](#). The Regular Expression option is selected by default.

2. Select appropriate search options. If no options are selected, a match is any string that matches the search criteria of any case.

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. The Source Editor considers text a match only if it appears exactly like the search string, not as a portion of a larger string. For example, <b>in</b> matches <b>in</b> , but not <b>include</b> .
Regular Expression	Accepts regular expression wildcards in the expression syntax. Click the  button to display a list of valid special characters.



**Tip** Leave Regular Expression deselected (for speed) if you simply want to locate a string.

3. Search the file for matches by using the Next and Previous buttons.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Search Replace


Use the Replace dialog box to search the current file for a text string and perform a string replacement.

1. Specify the search criteria in the Find What field.
2. Specify the replacement string in the Replace With field. You cannot use wildcard characters in a replacement string.
3. Click Replace.

The file is sequentially scanned for matching strings and the matching string is replaced with the replacement text.

You can use the Undo command to undo the entire set of replacements of the search string.

You can select these search options:

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	This option allows you to define a search string using regular expression wildcard syntax. Click the  button to display a list of valid special characters.
Confirm changes	You are 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also


**Search**  **Go to Line**

The Go to Line dialog box allows you to specify a line number in the current source file that you want to scroll to. If any text is currently selected, the selection is extended to include that line.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

**Search**  **Go to Function**

Use the Go to Function dialog box to select a function to go to.

- Type the function name, or select a function name from the list. When you click OK, the insertion point will be moved to the beginning of the specified function.

Select the Show Member Name to change the method display to show the name of the member first. The default is the form name first.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

## Search Go to Buffer

The Go to Buffer dialog box allows you to switch to an editing buffer. A buffer is created for each editing window, Source window, and Class Browser.

1. Select the Category of buffers that 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.

2. Select a buffer from the list.
3. Perform any appropriate task(s):

To	Click
Make the selected buffer's window current.	Go To
Display the Current Buffer Options dialog box to define automatic buffer formatting and maintenance.	Options
Save the buffer's content with its current file name	Save
Save the buffer's content with a new name.	Save As
Close the selected buffer's window and prompt you if there are unsaved changes.	Close

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

## Search Bookmarks

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.

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 can 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

## File Find in Files

The Source Editor's global find feature provides a powerful means of locating a string in any set of files.

To start a search, click Find. When the search is finished, Visual Cafe displays the results in the search results window. You can double-click a file to open it.

The result window's pop-up menu allows you to perform another search, go to the selected file source, add the selected item to the project, or add all files that matched the criteria to the project.

### **Tasks**

[Specifying the name and location of the files to be searched](#) (Name & Location tab)

[Setting advanced search criteria](#) (Advanced tab)


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can specify the search criteria, the type of files to search, and the files location on the Name & Location tab of the Find in Files dialog box.

1. Select the appropriate search criteria options at the bottom of the window.

<u>To</u>	<u>Do</u>
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 symbols	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 search string that starts or ends a line.

2. Specify the search pattern in the Find what field. The drop-down list displays the previous sixteen search strings. If Regular expression is selected, you can use the more button  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.

<u>To</u>	<u>Do</u>
Expand a search into subfolders	Select the Search subfolders option.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You specify file attribute and modification search criteria from the Find in Files dialog box's Advanced tab.

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 check boxes are three-state check boxes. If an attribute is checked, then files with the given attribute are searched. If an attribute is cleared, then files without the given attribute are searched. If an attribute is grayed, the attribute is ignored when Visual Cafe decides which files to search.

2. Specify modification criteria by enabling the Files created or modified option.

Set the appropriate values in the Date and Time fields. The field value "ignore" disables the associated data 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Tools Compare Files

The Compare Files dialog box provides the ability to 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.

1. 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 to show a drop-down list box containing the names of files recently compared. Click the browse button to display an open file dialog box for file selection.

2. Select the file that is mapped against file 1 -- File 2.
3. Specify the line number that the search should start from in both files. These numbers can be different.
4. Specify the arrangement of the source windows that display the results.

Visual Cafe then performs the comparison on a line-by-line basis.

Option	Description
Horizontal	Displays one window above the other.
Vertical	Displays one window next to the other.

When the Source Editor finds a mismatch, the editor highlights the lines in both files and reports where the mismatch was found. Click Next Match to re-synchronize the comparison. The Source Editor then highlights the next set of matching lines and the Compare 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.

## Editing a File

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can select text in any of these standard ways:

- Clicking and dragging the mouse
- Shift-clicking
- ALT-clicking

ALT+click+drag performs a column select.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Source windows support two typing modes: overwrite, 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 overwrite and insert modes.

The typing mode is displayed in the status bar.

 See Also

**Note** A change in typing mode applies to all open Source windows, not just to the active window.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

By default, the Source Editor 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.

#### **To set format options for a file**

- Choose Source  Format Options.

#### **To set format options for the environment**

- Choose Tools  Environment Options

 Format.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```




Because the editor is designed for source files, word wrap is not enabled by default. 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 locally or globally using the text format options.

#### **To set format options for a file**

- Choose Source  Format Options.

#### **To set format options for the environment**

- Choose Tools  Environment Options

 Format.

## Classes

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

You can locate classes in the Source window, Class Browser, Hierarchy Editor, and Project window, and view class definitions.

#### 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 Search

 See Also

Find to look in the current file.

- Choose Search  Find in Files to look in multiple files.

 See Also

[Details on this task](#)

#### 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 -). Press Shift then + or - to expand or collapse all entries in the tree view.

- If the class you want is not displayed in the pane, you might need to change what classes are displayed by choosing

Classes 

Options. See [Controlling class display in the Class Browser](#) 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.

 See Also

**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 Hierarchy  View Imports 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.

1. Select or click in a class, then choose Search  Go to Definition, 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.


 See Also

[Details on using the Class Browser](#)

#### To go to a class definition from the Class Browser

You can view source code in the Source pane of the Class Browser or in a Source window. The Source pane has the same editing options as the Source window.


- To display source code in the Source pane, select a class in the Classes pane then a member in the Members pane.

- To display source code in a Source window, from the Classes pane, select a class then choose Classes  Go to Source, or right-click a class and choose Go to Source.


 See Also

[Details on using the Source window](#)

#### To go to a class definition from the Hierarchy Editor

- To view a class in the Source window, select a class then choose Hierarchy  Go to Source, or right-click a class and choose Go to Source.
- 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**

- Double-click a class in the Packages or Files view.
- Select a class in the Packages or Files view, then choose Object  Edit Source, or right-click and choose Edit Source.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

The Class Wizard makes creating new Java classes and interfaces easier and more foolproof by setting up a complete prototype for you. You can also add classes manually.

**To add a new class with the Class Wizard**

- Launch the Class Wizard from the Insert menu, Class Browser, or Hierarchy Editor.

 See Also

[Details on using the Class Wizard](#)

**To manually add a new class from the Source window**

- Type the Java code that creates a class.

**To add a class from the Project window**

- Adding a [top-level component](#) to the Project window (Objects view) creates a new class and Java source file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

The Class Wizard makes editing Java classes and interfaces easier and more foolproof. You can also edit classes manually.

#### **To edit a class with the Class Wizard**

- Launch the Class Wizard from the Class Browser or Hierarchy Editor.

 See Also

[Details on using the Class Wizard](#)

#### **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**

- Select a class in the Classes pane then a member in the Members pane, and edit the class in the bottom Source pane.

#### **To change the inheritance hierarchy from the Hierarchy Editor**

- To change the inheritance hierarchy, click the line between a parent and base class, then drag the line by its anchor to another base class.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

 See Also


The 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.

1. To define a new class or interface with the Class Wizard, do one of the following:

- Choose Insert  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 in the Environment Options, you can press INSERT from the Class Browser or Hierarchy Editor.

If you select a class, that class becomes the default class to inherit from.

To edit a class or interface with the Class Wizard, do one of the following:

- Select a class or interface in the Classes pane of the Class Browser, then choose Classes  Edit Class, or right-click and choose Edit Class.
- Select a class or interface in the Hierarchy Editor, then right-click and choose Edit Class.

2. On the first page of the wizard, specify the options you need:

Option	Description
Type	Select Class if you are creating a <a href="#">class</a> or Interface if you are creating an <a href="#">interface</a> .
Name	Type the name of the class or interface.
Source	Type the complete path to the Java file. Click the Edit button to browse. In the Edit Class File dialog box, you can specify the package and Java file name, and the file path is displayed for you.
Package	Choose a <a href="#">package</a> to add the class or interface to, or None to not add it to an existing package.
Extends	If you are defining a class, choose a class to extend from.
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 <a href="#">inner classes</a> only.
Final or Abstract	Select Final class or <a href="#">Abstract class</a> .
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 <a href="#">bean</a> are that it can be instantiated (it is not an abstract class or interface) and it has a class constructor method that takes zero parameters.

3. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can choose the interfaces to implement.

4. From the Available interfaces list, select the interfaces you want to implement and click the downward-pointing arrow.

To move an interface from the lower list box to the upper one, select the interface and click the upward-pointing arrow.

5. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can choose the [methods](#) to override. Required methods already appear in the Override these methods list box.

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 cannot move methods that are required.

7. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can work with [properties](#).

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.
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 other objects get notified when the property changes value.
Constrained	Selecting this option generates code that takes care of creating notifications so 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 continue by clicking Next.

The next page of the wizard appears, where you choose [events](#) to generate.

10. From the Available events list, select the events 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 previous pages of the wizard.

12. Click Finish when you are finished with the definition.

The new class is inserted in the active project.

13. Complete the definition of the class or interface in the Source window or Class Browser.



[Details on using the Source window](#)





[Details on using the Class Browser](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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 Source Editor windows, including the Source window and the Class Browser Source pane. The Source pane is at the bottom of the Class Browser.

- Select code, then choose Edit  Cut or Copy, or right-click and choose Cut or Copy.
- Position the cursor where you want to paste, then choose Edit  Paste, or right-click and choose Paste.
- Drag a block of selected text to move it.
- Press CONTROL while dragging a block of selected text to copy the text. (A + appears over the cursor when a copy operation is being performed.)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The 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.

#### **To rename a class with the Class Wizard**

- Launch the Class Wizard from the Class Browser or Hierarchy Editor.



[Details on using the Class Wizard](#)

#### **To rename a class in the Project window**

- Renaming a [top-level component](#) in the Project window (Objects view) 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 constructor is renamed for you.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can delete a class from the Source window.

- Select the code, then press DELETE or choose Edit  Delete.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

 See Also

You can quickly change or delete the inheritance relationship between a class and its parent in the Hierarchy Editor and Class Browser.

#### **To change the inheritance hierarchy in the Hierarchy Editor**

- 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 its source code.

#### **To delete a class inheritance from 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.
3. Choose Delete Inheritance.

The class now inherits directly from `java.lang.Object`.

#### **To delete a class inheritance from the Class Browser**

1. In the Classes pane, choose Classes  Options, or right-click and choose Options.
2. In the Group/Sort tab, specify a Class Grouping of Hierarchically, then click OK.

When you are showing classes by 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 is 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`.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can control how classes display in the Classes pane.

1. In the Classes pane, choose Classes  Options, or right-click and choose Options.
2. In the Class Options dialog box, set the options you want:
  - In the Group/Sort tab, specify whether you want to group classes alphabetically, hierarchically, or by package.
  - In the Filter tab, specify whether you want to show imported classes.

 See Also


**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can enable and disable viewing imports in the Hierarchy Editor.

- While the Hierarchy Editor is the active window, choose Hierarchy  View Imports or right-click and choose View Imports to toggle the display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



### Class Options Filter tab

Use this tab to enable filtering of classes and members. This filtering results in a refined display in the Classes and Member panes of the Class Browser.

#### Show these members

Defines the type of members to display in the Member pane. Options include [access type](#) and method types. There must be at least one option selected in each group.

#### Show member types

Select this options to include the method's arguments in the listing.

#### Show imported classes

This is the single option for classes. Clearing this option prevents imported classes from displaying in the Class and Member panes.

#### Final, Static, Regular

These are Java method types. Refer to your Java documentation for more information.



**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



## Class Options Group/Sort tab

Use this tab to change the ordering of classes and member elements in the Class Browser.

### Group Classes

Specify how the classes in the Classes pane are to be grouped.

### Sort Members

Specify how the members in the Members pane are to be sorted in their group. Grouping of members is controlled with Group Members. The None option sorts the elements based on the order in which they were created.

### Group Members

Defines the grouping of members in the Member pane. The By Kind option groups the elements as Methods or Data. The By Access option groups the elements by their [access type](#).



**Tip** Press Shift then + or - to expand or collapse all entries in the tree view.

## Members

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

You can quickly locate methods (including [event handlers](#)) and data variables in the Source window and Class Browser, and view their definitions.

#### To go to a method in the Source window

1. In the Objects drop-down list of the Source window, 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 is not bold, it is created for you.

 See Also

**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 Source pane of the Class Browser.

1. Select or click a member, then choose Search  Go to Definition, 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.

 See Also

[Details on using the Class Browser](#)

#### 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 Search

 See Also

Find to look in the current file.

- Choose Search  Find in Files to look in multiple files.

 See Also

[Details on this task](#)

#### To locate a method or data variable in the Class Browser

1. Select a class in the Classes pane.

 See Also


[Details on this step](#)

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 are 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).

 See Also

**Note** If the method or data variable you want is not displayed in the pane, you might need to change what members are displayed by choosing Classes  Options. See [Controlling member display in the Class Browser](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also


You can conveniently add members from the Class Browser and Source window.

#### To create a new member from the Class Browser

1. In the Class pane, select the class you want to add a method or data variable to.

 See Also

[Details on this step](#)

2. Right-click in the Members pane and choose Insert Member. Or while the Class Browser is the active window, choose Insert  Member.

By default in the Environment Options, you can also press INSERT.


The Insert Member dialog box appears.

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 Source pane.

 See Also

**Note** If the method or data variable you want is not displayed in the pane, you might need to change what members are displayed by choosing Classes  Options. See [Controlling member display in the Class Browser](#) for more information.

#### To add an event handler from the Source window

[Event handlers](#) and other source code can be added in the Source window. 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.addtypeListener` or `object.addtypeAdapter`) 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, plus the interaction specified in the wizard is generated in the event handler. See [Using the Interaction Wizard](#) for more information.

To add an event handler from the Source window:

1. In the Objects drop-down list of the Source window, 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 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.

#### To manually add a new member from the Source window

- Type the Java code to add the member.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also


You can conveniently edit members from the Class Browser and Source window.


#### To view or edit the source code of a member

You can view source code in the Source pane of the Class Browser or in a Source window. The Source pane has the same editing options as the Source window.

- To display source code in the Source pane, select a class in the Classes pane then a member in the Members pane.

 See Also

**Note** If the class or member you want is not displayed in the pane, you might need to change what classes and members are displayed by choosing Classes  Options. See [Controlling member display in the Class Browser](#) for more information.

- To display source code in a Source window, from the Classes pane, select a class then choose Classes  Go to Source, or right-click a class and choose Go to Source.

 See Also

[Details on using the Source window](#)

#### To view attributes of a member from the Class Browser


1. In the Class pane, select the class that contains the member.


 See Also

[Details on this step](#)

2. Select the member in the Members pane.

 See Also

**Note** If the method or data variable you want is not displayed in the pane, you might need to change what members are displayed by choosing Classes  Options. See [Controlling member display in the Class Browser](#) for more information.

3. Right-click in the Members pane and choose Member Attributes. Or while the Class Browser is the active window, choose Classes  Member Attributes.

The Member Attributes dialog box displays.

4. Change the access type if needed, then click OK.

#### To edit a member in the Source window

- Find the member and edit the Java code.



 See Also

[Details on finding a member](#)

{ewc\_HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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 Source Editor windows, including the Source window and the Class Browser Source pane. The Source pane is at the bottom of the Class Browser.

- Select code, then choose Edit  Cut or Copy, or right-click and choose Cut or Copy.
- Position the cursor where you want to paste, then choose Edit  Paste, or right-click and choose Paste.
- Drag a block of selected text to move it.
- Press CONTROL while dragging a block of selected text to copy the text. (A + appears over the cursor when a copy operation is being performed.)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can conveniently rename members from the Class Browser and Source window.

#### **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.

 See Also

[Details on finding a class](#)

 See Also

[Details on finding a member](#)

2. When an edit box appears, type the new name or edit the existing name.

#### **To rename a member in the Source window**

- Find the member and rename it, then search for its name and rename all occurrences.

 See Also

[Details on finding a member](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

You can delete a member from the Class Browser or Source window.

 See Also

**Note** Deleting an [event-handler](#) is the same as deleting an [interaction](#). See [Using the Interaction Wizard](#) for more information.

#### To delete a member from the Class Browser


1. In the Class pane, select a class that contains the member.


 See Also

[Details on this step](#)

2. Select a member or shift-click multiple members in the Members pane.

 See Also

**Note** If the method or data variable you want is not displayed in the pane, you might need to change what members are displayed by choosing Classes  Options. See [Controlling member display in the Class Browser](#) for more information.

3. Right-click in the Members pane and choose Delete Member. Or while the Class Browser is the active window, choose Classes  Delete Member.

The member is deleted from the class.

 See Also

**Note** You get a deletion confirmation dialog box only if it is enabled in the Environment Options. See [Specifying code editing options](#) for more information.

#### To delete a member from the Source window

- Select the code, then press DELETE or choose Edit  Delete.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can control how members display in the Members pane.

1. In the Members pane, right-click and choose Options. Or while the Class Browser is the active window, choose Classes



Options.

2. In the Class Options dialog box, set the options you want:
  - In the Group/Sort tab, specify whether you want to sort members by access (public, private, protected, and package), alphabetically, or not at all.
  - In the Group/Sort tab, specify whether you want to group members by kind or by access.
  - In the Filter tab, specify what members you want to show (choose from public, private, protected, package, final, static, and regular) and if you want to display their types.
  - In the Inheritance tab, specify whether you want to show inherited methods, and if so, if you want to display full method names or show overridden methods.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

**Class Options**  **Inheritance tab**

Use this option to toggle the display of inherited methods in the Class Browser's Members pane.

**Show inherited methods**

Toggles the display and activates two suboptions.

**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**

Adds methods that are overridden by methods in the class to the Member listing.

## Events

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



[Event-handler](#) code is automatically added 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.addtypeListener* or *object.addtypeAdapter*) 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, plus the interaction specified in the wizard is generated in the event handler. See [Using the Interaction Wizard](#) for more information.

To add an event handler from 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

 See Also

You can conveniently edit [event handlers](#) from the Class Browser and Source window.

#### To edit an event handler from the Class Browser

- Select a class in the Classes pane then a member in the Members pane, and edit the member in the bottom Source pane.

 See Also

[Details on finding a class](#)

 See Also

[Details on finding a member](#)

#### To edit an event handler in the Source window

1. In the Objects drop-down list of the Source window, 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 is not bold, it is created for you.

3. Edit the Java code.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

Deleting an [event handler](#) is the same as deleting an [interaction](#). See [Using the Interaction Wizard](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also

Visual Cafe has a command that automatically converts a Java source file from the 1.0 event model to the 1.1 event model.

 See Also

**Note** If you add a Visual Cafe version 1.0 Java file to a Visual Cafe version 2.0 project without first opening its 1.0 project, the Java file will not be set up to use the new Visual Cafe components (specifically, the INIT portion of the code). You should first open the 1.0 project in the newer version of Visual Cafe so that the file is updated for you.

1. Open the file in the Source window.

2. Choose Project  Migrate.

Visual Cafe parses the `handleEvent` and action methods. From these methods, new code is generated to handle the events in the 1.1 event model. This includes generating the required [adapters](#) and [listeners](#), and registering the listeners. For menus with menuitems 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



If you want to use `removeActionListener` or `addActionListener` within another method, you need to create code to allow access to the implemented [listener](#) from any part of the class file.

After Visual Cafe creates a listener in your Java source code, you will find the comment tag `//{{REGISTER_LISTENERS,` followed by the instance of a listener. For example, in an `init` method for an applet with a button action listener, you would see the following code:

```
//{{REGISTER_LISTENERS
SymAction lSymAction = new SymAction();
button1.addActionListener(lSymAction);
//}}
```

The `ISymAction` listener is not accessible outside of the `init` method. To get access, you should create a global reference to this instance.

To do this, after the close of the register listeners code block `( // }} )`, add the following line of code:

```
glSymAction = lSymAction;
```

Then locate the comment tag `//{{DECLARE_CONTROLS` in the body of the class. After the end of this code block `( // }} )`, add the following line:

```
SymAction glSymAction;
```

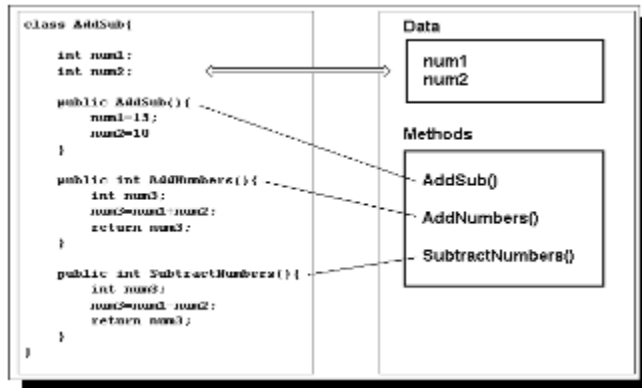
Now you will have access to the implemented listener from any part of the class file by using the `glSymAction` variable.

## Obsolete Topics

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

Much like DNA determines the structure of a living object, classes determine the way [objects](#) look and behave. Classes contain source code, which includes instructions for object creation or [instantiation](#).

A class contains the code to define a collection of data variables and [methods](#). The following diagram illustrates what the code for a class looks like:



*The AddSub class definition*

This is the code for a simple class called `AddSub`. Note that the `AddSub` class listed above is not a complete Java program that you can run. A Java program has a **main** method or [extends](#) `java.applet.Applet`. Classes containing a **main** method or extending the `Applet` class can be compiled into programs. Other classes, like `AddSub`, are only a class definition. They simply specify a class name and define what [data](#) and methods the class contains.

In this example, the `AddSub` class has two variables as its data: the integers `num1` and `num2`. The `AddSub` class also has three methods: `AddSub`, `AddNumbers`, and `SubtractNumbers`. The first method defines the [constructor method](#) because it has the same name as the class, `AddSub`. The other two methods are called `AddNumbers`, and `SubtractNumbers`. These two methods change the data and return a result.

Once a class has been defined and compiled, it can be included in your Java programs. Note the following:

- A Java source file (with a **java** extension) can contain more than one class.
- When a source file is compiled, one or more class files (with a **class** extension) are generated — one class file per class.

<<The rest down looks wrong. Needs to be fixed.>>

However, to use any of the data or methods defined in a class, you must write a program that creates components based on the class. The process of creating a component based on a class definition is called creating an instance of a class.

You can access the source for a class component by double-clicking the source file in the Packages view of the Project window. Like forms, when you open the component, the primary editor is invoked. In the case of class files, the primary editor is the Source Editor, which is a text editor.

When you add a [top-level component](#) to your [project](#), a Java source file is generated in your project. This Java source file contains the form [declaration](#) and, when components are added to the form, [components](#).

### Understanding components

When you want to make use of the code in a class definition, you need to create an instance of that class. An instance of a class is a variable that contains references to all the data and methods of the class on which it is based. Just as an `int` variable is based on the type integer, instances of a class store the data and can access the methods of the class on which it was based.

A class is a blueprint that defines basic data and methods of an object. The class definition is used to define templates of components that can be instantiated and then used in your programs. You can create multiple instances of the same class just as you can create multiple variables of the type `int`. Once you have created a component, that component contains its own data and provides the methods to access that data. You can use an instance of a class as a starting place for your own component design. You can also add to or redefine existing methods and variables.

For example, suppose you want to write a program that draws a rectangle. First you must think about the characteristics of the rectangle component you want to model. Some properties of a rectangle are width and height. A behavior of a rectangle is that it can be drawn.

The data for this component could be two integer variables to hold the width and the height, and a method to draw the rectangle.

Defining the data and methods together in a component or class encapsulates the properties and behaviors into a single unit. Once the code for the rectangle has been created, you or anyone else can reuse it whenever you want. Encapsulation and reusability are two of the most beneficial characteristics of an object-oriented programming language.

When you work with components in an object-oriented programming language, you work with the features of that component to make it behave a certain way, or to interact with other components.

Sorting classes

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The Classes pane of the Class Browser displays a list of your project's Java classes. This view also provides an outline in which subclasses display indented and below their parent. A class that implements multiple interfaces displays below each interface class.

You can change the ordering of the classes by using the Options command in the pop-up menu. This command opens the Class Options dialog box Group/Sort tab. By default, the classes appear alphabetically by name.

To add and define a class or interface, use the pop-up menu Insert Class command. This command is the same as Insert



Class off the main menu.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The Members pane of the Class Browser displays a list of methods and data elements in the selected class. Clicking a member causes its associated Java source code definition or declaration to display in the Source 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 = public, yellow = protected, blue = package, and red = private).

You can change the display of elements in the Members pane by using the Options command in the pop-up menu. This command opens the Class Options dialog box Group/Sort tab.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


The Class Browser and Hierarchy Editor are Java class editors. These tools let you work with your source code in an object-oriented fashion, rather than in the traditional file-oriented manner.

The Class Browser and Hierarchy Editor can perform similar operations on classes and hierarchies; they differ mostly in their interface. The Class Browser's interface emphasizes member editing; the Hierarchy Editor's interface, through its graphical display, emphasizes inheritance relationships. You can use one or the other, or both simultaneously, according to your preference.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

The Insert Class Wizard makes creating new Java classes and interfaces easier and more foolproof by setting up a complete prototype for you.

1. To define a new class with the Insert **<<Create?>>** Class Wizard, do one of the following:

- Choose Insert  Class.
- Select a class in the Classes pane of the Class Browser, then right-click and choose Insert Class.
- Select a class in the Hierarchy Editor, then right-click and choose Insert Class. **<<Are you going to implement this?>>**
- Select a class in the Project window (Packages or Files tab), then right-click and choose Insert Class. **<<Are you going to implement this?>>**

If you select a class, that class becomes the default class to inherit from.

2. On the first page of the wizard, specify the options you need:

Option	Description
Type	Select Class if you are creating a <a href="#">class</a> or Interface if you are creating an <a href="#">interface</a> .
Name	Type the name of the class or interface.
Source	Type the complete path to the class file. <b>&lt;&lt;I don't get the Edit Class File dialog box.&gt;&gt;</b>
Package	Choose a <a href="#">package</a> to add the class or interface to, or None to not add it to an existing package.
Extends	If you are defining a class, choose a class to extend from.
Access	Select whether you want to make access to your class through Package, or Public, Private, or Protected. <b>&lt;&lt;Is this working right?&gt;&gt;</b>
Final or Abstract	Select Final class or <a href="#">Abstract class</a> .

3. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can choose the interfaces to implement. Required interfaces already appear in the Implement these interfaces list box.

4. From the Available interfaces list, select the interfaces you want to implement and click the downward-pointing arrow.

To move an interface from the lower list box to the upper one, select the interface and click the upward-pointing arrow. You cannot move interfaces that are required.

5. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can choose the [methods](#) to override. Required methods already appear in the Override these methods list box.

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 cannot move methods that are required.

7. If you want to review or change part of the definition, click Back.

You can go back to previous pages of the wizard

8. Click Finish when you are finished with the definition.

The new class is inserted in the active project.

9. Complete the definition of the class or interface in the Source window or Class Browser.



[Details on using the Source window](#)



[Details on using the Class Browser](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

As an alternative to using the Insert menu, you can add a class within the Class Browser.

1. Select a class as the base class of the your new class.
2. Right-click in the Classes pane to display the pop-up menu.
3. Select the command Insert Class
4. Complete the Insert Class wizard.



[Details on this step](#)

The new class name appears in classes pane. The source pane shows the class declaration.

**Note** To delete a class, select the class and press DELETE.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

As an alternative to using the Insert menu, you can add a class within the Hierarchy Editor.

1. Select the parent/base class.
2. Drag the selection and release it in any whitespace of the window background.
3. Complete the Insert Class wizard.





See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

To add classes to an existing package from within the Project window:

1. In the Project window, click the Packages tab.
2. Select the package to which you want to add the class.
3. Choose Insert  Class.
4. Complete the Insert Class wizard.  
 [Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

The Visual Cafe Source window is where you can create, display, and edit code. It is a full-featured text editor with many options and features particularly useful when editing Java source code. It provides for full Java [syntax](#) highlighting and flexible navigation tools. 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.

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 next to the source pane. 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 text box.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can open a component's Java source file to enhance the component behavior with custom Java code.


1. In the Form Designer, double-click the component.  
**or**  
Select the component in the Source window or Form Designer, then choose Edit Source from the pop-up menu.
2. In the Source window, add your custom code to the file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can move the insertion point in a file by using the mouse or keyboard in the standard Windows text-editing manner. In addition, you can jump to specific points in a file using commands from the Search menu.


The following are common tasks that you do while editing. Other "Go To" functionality is available on the Search menu.

### Jumping to a matching delimiter


A common problem in source code is parentheses ( `()` ), brackets ( `[]` ), and braces ( `{}` ) that don't match. To find the other half of a pair of these delimiters, position the insertion point in front of one of the delimiters and choose Search  Go to Matching Delimiter. The insertion point moves to the other half of the pair.

Delimiter checking can also be done automatically using the source format options. The Source Editor looks for any parenthesis, bracket, or brace, including text in strings and comments.

### Jumping to a specific line

To jump to a specific line, choose Search  Go To Line. The Go To Line dialog box opens. Type the line number in the text box and click OK. The editor moves the insertion point to the beginning of the specified line.


### Jumping to a function

To jump to a specific function, choose Search  Go to Function. The Go to Function dialog box opens. You can select a function name from the scrolling list or type in a function name. The insertion point then moves to the beginning of the specified function.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

The global find feature of the Source Editor provides a powerful means of locating a string in any set of files.

#### **To search for a string in multiple files**

- Choose Search  Find in Files. You can choose to search:
  - All source files in the current project
  - All files listed in the Search window (which opens after the first search)
  - All files matching the criteria you specify, including filename, directory, date, time, and file attributes

#### **To compare two files**

- Choose Tools  Compare Files.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

**Classes**  **Class Attributes**  
**Hierarchy**

 **Class Attributes**

Use this dialog box to change the attributes for the selected class.

You can change the class name and its base class.

<< Opens the Edit Class Wizard, where you can change the definition of a class or interface???>>

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

Event handlers and other source code can be added directly in the Source window:

1. Open the Source window for the form.
2. In the Source window, select an event handler from the Events/Methods drop-down list.  
The appropriate event handler logic is added automatically to the source code.
3. Replace the placeholder text `/// to do: place event handler code here` with appropriate Java code.
4. Save the file.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

You can add methods to a component from the Class Browser or Source window.

[Adding a method from the Class Browser](#)

[Adding a method from the Source window](#)Adding\_methods\_SourceEditor\_howto

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

The purpose of the Visual Cafe Source Editor is to 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. The Source Editor is designed to work in concert with other Visual Cafe tools.

The Visual Cafe Source window and the Class Browser Source pane both share the same editing functionality. The editor uses standard Windows editing commands and has special features that make working with Java files easier. For example, the editor can automatically indent or unindent after braces and can check delimiters.

In addition, the Source Editor can display keywords, preprocessor directives, and comments in special font styles and colors. This technique helps track errors in source code while you are 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 and preprocessor directives are easier to spot when they are in a different color or font style. Misspelled keywords can be caught immediately when they remain displayed in the default font.

Source windows are an integral part of the Visual Cafe environment and work together with other Visual Cafe windows to make application development easier. For example, 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 an error message, Visual Cafe opens a Source window on the corresponding source file, if necessary, and then jumps to the line in the source code that caused the error.

The Source Editor supplies standard Windows functions for cutting, copying, pasting, and deleting text. These functions can be accessed through either the Edit menu or the pop-up menu.



[Click here for guidelines on adding custom code](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

With Visual Cafe's extensive [Component Library](#), you can create many applets and applications without writing any Java code. However, there may be specific behaviors that require [custom code](#) to be added to the Visual Cafe source file.

You may want to add or modify Java code for a [component](#) to:

- add event behavior
- perform exception handling
- implement runtime security for applications
- extend a method

Before you add custom code, you should create your project and use Visual Cafe's powerful visual tools to create your forms. You can add components to your project, arrange components in the Form Designer, modify the look of individual components with the Property List, and add interactions between components or a component and itself with the Interaction Wizard. It is a good idea to do as much as you can with the Visual Cafe tools before you add your custom code.

When you are visually satisfied with your project, you can then add any necessary code enhancements using the Source window or Class Browser.

**Caution** Visual Cafe only maintains the code that it automatically generates. This means that when you add code to an component and later delete the component from the form or project, only the automatically generated code is removed during the component deletion. You must edit the appropriate Java source files to remove your own code references.

{button ,JI('VCAFE.HLP','Adding\_Code\_to\_a\_Java\_Source\_File\_howto')}} [Click here for more guidelines](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can use standard graphical user interface (GUI) techniques for dragging and dropping objects into and among other objects.

<<Any drag/drop capabilities for Class Browser or Hierarchy Editor? I assume the source pane has the same drag/drop characteristics as the Source window. Also, is the table below complete?>>

<u>Dragging</u>	<u>Into</u>	<u>Effect</u>
Code in the Source window	Windows operating system	Creates a <b>src</b> file containing the selected code. <<This doesn't appear to work.>>
Code in a Source window	Event prototype in a Source window	Moves just the event code into the new event. Press CONTROL to copy. <<True? Better explanation?>>
Event or method item	Source window	Inserts Java code that calls the event and enters the method name in the new event prototype. <<True?>> Press CONTROL to copy.

## Intro/Marketing - Overview Section

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Symantec Visual Cafe is a visual Rapid Application Development (RAD) tool designed exclusively for the Java programming language. Visual Cafe is a complete form-centric development environment that provides the tools and components necessary to develop, debug, and deploy high-performance Web applets and stand-alone Java applications.

Visual Cafe provides you with a rich set of What-You-See-Is-What-You-Get (WYSIWYG) tools that you use to assemble and debug your applets and applications.

Visual Cafe comes in a Standard Edition, Professional Edition, Database Edition, and Enterprise Suite. If a feature applies to the Professional Edition, but not the Standard Edition, it is noted; for example "(Professional Edition)." Professional Edition features also are included in the Database Edition and Enterprise Suite.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



This help system describes how to use Symantec Visual Cafe to develop Java programs. It has the following kinds of topics:

**Topics with  
this image...**



**Tell you this ...**

How to perform high-level processes, such as creating applets, applications, beans, and servlets



Conceptual information that helps you understand Visual Cafe features



How to perform tasks



How to use JavaBeans components delivered with the Visual Cafe environment

You can access help by choosing an item in the Help menu or by selecting an item in the environment and pressing F1.

You can obtain help while coding by positioning your cursor in the code and pressing F1, or by right-clicking and choosing View Javadoc. You can also enable the Code Helper and Syntax Checker to help you code.

In the Help menu, you can start a Visual Cafe demo that includes videos to help you quickly become familiar with Visual Cafe features.

Be sure to take advantage of the extensive Visual Cafe help system, which is designed to assist you during every phase of your development process.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Symantec Visual Cafe is a complete Rapid Application Development (RAD) environment for Java. It provides an easy-to-use and seamlessly integrated visual development environment featuring a complete professional toolset. Visual Cafe allows you to create and debug Web-hosted Java applets, stand-alone Java applications, and libraries.

Featuring a sophisticated set of high level, [two-way tools](#), Visual Cafe is a leading Java development environment:

- The form-centric development environment has a powerful integrated form design tool.
- A configurable widget toolbar contains your favorite form components.
- The Component Library contains a repository of components for form development.
- Add-in third party components are easily made available through the extensible Component Library.
- The Interaction Wizard allows you to specify actions based on [events](#).
- Form and property list modifications generate real-time source code.
- A debugging window incorporates a tree view to collapse and expand the various scopes.
- Utilities for deployment, localization, version control, and more assist you in performing tasks faster.

The Symantec Java compiler that comes with Visual Cafe was written specifically for the Java language. Instead of typing a string of commands at a command prompt, you click a single icon to compile and execute a program. Visual Cafe provides useful, meaningful messages indicating how the compiling process is progressing.

In addition, Visual Cafe provides many Windows-based development tools not found in the JDK. Visual Cafe's graphical development environment makes Java programming easier and faster.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe provides several tools to help you create [forms](#). With Visual Cafe, you can add visual [components](#) to your forms to quickly assemble [applets](#) and [applications](#).

Each visual component has a set of properties, a [visual element](#), a set of methods and associated Java code, and may have one or more [interactions](#) and [events](#).

Some Visual Cafe utilities are:

- |   |  |
|---|--|
| • Visual Cafe window                    | • Component Palette                        |
| • Project window                        | • Component Library                        |
| • Form Designer                         | • Property List                            |
| • Source window                         | • Menu Designer                            |
| • Class Browser                         | • Interaction Wizard                       |
| • Class Wizard                          | • Hierarchy Editor                         |
| • JavaBean Wizard                       | • Deployment utility                       |
| • Code Helper                           | • Debugger                                 |
| • Syntax Checker                        | • Pluggable VMs                            |
| • Servlet Wizard (Professional Edition) | • Localization Tool (Professional Edition) |

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



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.

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 together the windows that have related functions. Workspaces provide a means of organizing the multiple windows of the development environment into logical groups. For example, when you edit source code you could use a workspace that displays the text editor, the Messages window, and the Project window. When you are debugging your program, you could use another workspace that displays windows for the different debugging tools.

Visual Cafe supports the extended mouse functions of the Windows 95 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 the different parts of the screen. You may discover some useful shortcuts!

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Symantec Open APIs offer a broad range of functionality. In general, the Open APIs are designed to give third-party vendors flexible ways to integrate their tools into Visual Cafe for Java without imposing any unnecessary restrictions. For technologies requiring a native interface, third parties can use any C, C++, or Pascal environment to implement the integration. Here are some features of the Open APIs:

- The Project Templates API offers vendors the ability to register and add new project templates to Visual Cafe for Java, and supports both dynamic (interactive wizard) and static project-template creation.
- The Source Code Manipulation and Integration API provides a way for third-party software developers to ensure that interactive source code changes can be made with Visual Cafe, allowing for live integration with source code editing and modeling tools. Vendors can use this API to add their own classes and components to Visual Cafe for Java and ensure that when a user is working with advanced tools changes made in the source code are reflected across all files.
- The Symantec BeanInfo Extension APIs provide visual development environments with information missing from the BeanInfo specification for the placement and attributes of JavaBeans within a development environment, and to provide code generation between objects within that environment.
- The Menu Hooks API provides a way to add menu items to the Visual Cafe for Java menu bar.
- The Project Windows Extension API lets vendors add columns to a Project Window, notify owners or users of specific events, and customize the Project window with their own contextual help messages.
- The CASE API enables third parties to receive notification of project-level events, query, and set component property values, thus seamlessly integrating developers' use of CASE tools with Visual Cafe for Java.
- The Version Control API allows for vendor implementation of their version control system's user interface into Visual Cafe, allowing end users to install and work with multiple version-control systems on a per-project basis.
- The Data Binding API helps third-party component developers provide their users with a way to develop the exchange between Java UIs and databases without editing code, and offers a convenient way to bind visual components to databases.

See the Open API documentation for more information. (Currently, the latest revision of this documentation is available at <http://cafe.symantec.com/>.)

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

An applet can add dynamic, interactive functionality to your Web page. Visual Cafe makes it easy for you to develop an applet. You can follow these basic steps:

1. Create a new project.

To get started quickly, you can use an Applet project template.

 See Also

[Details on this step](#)

2. Design the user interface.

 See Also

[Details on this step](#)

3. Enhance Java source code.

 See Also

[Details on this step](#)

4. Set project options.

If you started with an Applet project template, you probably do not need to set project options.

 See Also

[Details on this step](#)

5. Run the applet in Visual Cafe.

 See Also

[Details on this step](#)

6. Debug the applet, if needed.

 See Also

[Details on this step](#)

7. Add the applet to your HTML page.

 See Also

[Details on this step](#)

8. Run the applet in the HTML page.

 See Also

[Details on this step](#)

9. Deploy the applet.

 See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe provides powerful tools for assembling an application quickly. In general, the development cycle is made of these basic steps:

1. Create a new project.

To get started quickly, you can use an Application project template.

 See Also

[Details on this step](#)

2. Design the user interface.

 See Also

[Details on this step](#)

3. Enhance Java source code.

 See Also

[Details on this step](#)

4. Set project options.

If you started with an Application project template, you probably do not need to set project options.

 See Also

[Details on this step](#)

5. Test run the application in Visual Cafe.

 See Also

[Details on this step](#)

6. Debug the application, if needed.

 See Also

[Details on this step](#)

7. Deploy the application.

 See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe makes it easy for you to develop a [servlet](#) by supplying a Servlet Wizard that gets you started. (This feature is not available in the Standard Edition.) You can follow these basic steps:

1. Create a new project.

To get started quickly, you can use a Servlet project template, which brings up the Servlet Wizard.

 See Also

[Details on this step](#)

2. Enhance Java source code.

 See Also

[Details on this step](#)

3. Set project options.

If you started with an Servlet project template, you might not need to set project options.

 See Also

[Details on this step](#)

4. Run the servlet in Visual Cafe.

 See Also

[Details on this step](#)

5. Debug the servlet, if needed.

 See Also

[Details on this step](#)

6. Deploy the servlet.

 See Also

[Details on this step](#)

## Obsolete

The Symantec Java compiler that comes with Visual Cafe was written specifically for the Java language. Instead of typing a string of commands at the command prompt, you click 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, meaningful messages indicating how the compiling process is progressing.

In addition, Visual Cafe provides many Windows-based development tools not found in the JDK. Visual Cafe's graphical development environment makes Java programming easier and faster.

### **Sun Microsystems' Java compiler and the 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 along with any other required parameters.

Sun's Java compiler also comes with the Java Developers 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

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, coordinating text messages, and mapping functions to visual components and messages. You can do most of these tasks with the [Property List](#) by setting [properties](#) to define, refine, and control the appearance and behavior of your visual components.

Visual Cafe creates and updates the source code as you create the [layout](#). This feature of Visual Cafe allows you to concentrate on designing and allocating other resources for your applet or application. It is possible to design, develop, and build an [applet](#) or [application](#) without having to write a single line of source code.

The Visual Cafe tools help you find and fix problems, reference the rules of the Java language, and organize the entire process of creating Java [applets](#) and [applications](#). Visual Cafe facilitates these tasks by providing an Integrated Development and Debugging Environment (IDDE). In other words, it is a single application that provides tools for the creation and development of a computer program.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

When a development environment is said to be integrated, this means that the tools in the environment work together. For example, the compiler might find an error in the [source code](#). In addition to displaying the error, you can double-click the highlighted error so that Visual Cafe opens the source file in the source editor and jumps to the exact line in the source code where the error occurred. Creating programs is faster and easier using tools that work together.

A large part of [applet](#) and [application](#) development involves adding and arranging [components](#) on [forms](#) and applets. Visual Cafe provides tools to make designing forms a simple process.

What is a visual component in Visual Cafe? Any visual component, such as a form, applet, or button, has:

**A set of properties**

Governs how components display and how they behave. Properties are accessible from the Property List.

**A visual element**

Shown as an icon in the Project window, and directly editable by double-clicking and opening the component.

**A set of event handlers**

Code associated with any number of its [event handlers](#) are accessible from the Source window.

**One or more interactions**

When provided, Visual Cafe translates into code. Interactions are *not* an integral part of all components.

Each of these component attributes has its own editor, from the Form Designer to the Interaction Wizard, making it easy to manipulate visual components.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

Developing a Visual Cafe [applet](#) or [application](#) happens in two stages: creating the program and developing it. In the first stage, you design, create, and implement the graphic user interface (GUI) your program will need. You also make a very simplistic arrangement of all the components needed for this program. The second stage is when you bring your project to life by adding and modifying source code, debugging, redesigning, and testing your project.

### **Step one: creating your applet or application**

When you start a new project in Visual Cafe, Visual Cafe creates the basic [project](#) files required for a Visual Cafe applet or application, including source files and the Visual Cafe project file. Visual Cafe then loads these project files, and you can immediately begin working on your project.

To begin working on the GUI of your applet or application, lay out the various components and controls by dragging them from the Component Palette onto the Form Designer.

When you save your work, Visual Cafe saves the whole project as a .vep file along with all the changes and other files you may have added to your project. The .vep file is very much like a file folder for storing all of your work.

### **Step two: developing your applet or application**

The process for developing an applet or application involves changing or adding source code if necessary, adjusting the properties in the [Property List](#), working with the Hierarchy Editor and Class Browser, then compiling, linking, testing, and debugging. These steps are iterative, meaning that they are repeated constantly throughout the standard software development process.

### **Common tasks**

Here are procedures that are common to developing both applets and applications.

The first step in developing your program is to create the [graphic user interface](#) (GUI) using the Form Designer. Then you draw the components that make up the interface onto the forms you create by dragging and dropping the visual components from the Component Palette.

Next, you set properties for the components you placed on the Form Designer. The Property List provides a dialog box for you to set properties for all the components on a form. The Property List is also where you specify the text for all labels.

Now you create [event handlers](#). Event handlers contain code that is executed when an event occurs, such as when a user clicks a mouse button or enters information through the keyboard. You can automatically create event handlers (in other words, bind events) from the Source window or the Interaction Wizard.

Finally, you either run the program directly, otherwise run it in the debugger to debug your application.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

Visual Cafe [applets](#) and [applications](#) are cross-platform Java programs that you design, develop, and build using the drag-and-drop features of the Windows operating system through Visual Cafe. The program code used to create applets and applications are fundamentally similar. Both applets and applications are created using the same basic programming concepts.

Because the Web browser is responsible for running an applet, program instructions for applets have a different organization than the instructions for applications.

### More about applets

Java applets, like all Java programs, are made of source code that is 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 displays it.

Another feature of applets is that when they are executed within a Java-capable browser, the applet adopts the look and feel of the client machine's operating system and native interface controls. This means that a Java applet will look and feel like it was written for the Macintosh when it runs in the Macintosh environment, Windows applets will have the Windows look to them, and UNIX applets will reflect the look of the various flavors of UNIX.

Applets, however, cannot read from or write to a user's local hard drive. To add this capability to your applet, Symantec has created the database solution **dbAnywhere**.

When you create an applet with Visual Cafe, you create a subclass of the class Applet in the java.applet package. This applet class enables your applet to work within a Web browser and to utilize the capabilities of the Abstract Window Toolkit. The AWT allows your applet to include user interface (UI) elements, to handle mouse and keyword [events](#), and to display to the screen. Although your applet can utilize as many [classes](#) as it needs, the main applet class triggers the execution of the applet. It's signature is as follows:

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

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

### Applets: advantages and disadvantages

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.

Standalone applications have [overhead](#) that must be written to make the program run properly; a standalone program must be able to start and stop the program, perform memory management, and handle display functions. Although this requires additional programming work, applications do not depend on the presence of a Web browser, nor do they have limitations on the kind of operations they can perform.

Because the browser software provides this functionality for applets automatically, applets are an attractive type of Java program to work with.

### Applet limitations

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 cannot cause damage to your system. To prevent applets from being destructive, Java enforces the following limitations:

- Applets cannot 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 cannot run any programs (or parts of programs like [DLLs](#) or shared files) on the viewer's computer. This prevents an applet from calling destructive programs that do not have the limitations of the applet.
- Applets can only establish connections between the server computer where the applet is stored and the viewer's computer. This prevents the applet from connecting the viewer's computer to another server without the viewer's knowledge.

Java applications do not have these limitations and can be used to build fully functional software programs.

### More about applications

An application is a Java program that runs with a [Java Virtual Machine](#) (Java VM) or a Java-compatible browser that is installed on the client system. An application is not displayed on a Web page. Java applications can be built because the Java language includes useful features that are not found standard in other languages. For example, Java comes with existing libraries of program code that make networking and graphics operations easier than ever.

Java applications, unlike applets, can read from and write to the client machine's hard drive. Applications can create their own frames, title bars, and menus.

Since applications are Java programs that run on their own, applications can be as large or as small as you want or need them to be. The only class that is needed to run an application is the **main** method. When you run your compiled Java class with the Java interpreter in Visual Cafe, the **main** method is called first.

The signature for the **main** method looks like this:

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

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](#). The main method of your application must be declared public for the application to run.

The keyword static means a storage class.

The keyword void tells the main method to not 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](#).

## Environment Options

See Also

See Also

See Also

See Also

See Also

See Also

See Also



 See Also

See Also

```
{ewc HLP25632,HLP256_UPPER_LEFT,can.bmp}
```



You can customize the Visual Cafe development environment by:

[Setting environment options](#)

[Setting project options](#)


[Modifying workspaces](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

A large number of views are available in the Visual Cafe environment. Visual Cafe lets you save a configuration of windows as a unique [workspace](#). A workspace provides a convenient way to switch from one screen layout view to another.

Workspaces are task-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 Window  Workspaces.

Visual Cafe supplies you with two workspaces: Edit and Debug. The Debug workspace automatically loads when you run your application and opens necessary debugging windows. When the processing 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 while running a project and when you stop the project. They are saved dynamically. For example, if the Property List is open in the Edit workspace and you close that window while in the Edit workspace, the change is saved to the workspace. This feature ensures that as you run and stop your programs, the window state remains as you last left them.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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:


- Choose Window  Workspaces

 See Also

*Workspace-name.*

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

### To save your current Visual Cafe 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. Choose Window  Workspaces

 See Also

*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 Workspace field. The workspace is also added to the Workspace list.

### To rename a workspace

1. Choose Window  Workspaces

 See Also

*Rename.*

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 Workspace field. The name also changes in the Workspace list.

### To delete a workspace

**Caution** Deleting a workspace immediately deletes the workspace named in the toolbar Workspace field.

 See Also

**Note** You cannot delete the last remaining workspace.

- Choose Window  Workspaces

 See Also

*Delete.*

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

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

## Tools Environment Options

Visual Cafe allows you to set options that are global to all of your projects.

 See Also

**Note** To customize a single project, use the Project Options dialog box.

### Tasks

[Setting environment options in the General tab](#)

[Setting debugging options for the environment](#)

[Specifying text formatting for Visual Cafe windows](#)

[Mapping Visual Cafe commands to key sequences](#)

[Customizing the font and color of Visual Cafe windows](#)

[Specifying backup and save options](#)

[Specifying code editing options](#)

[Using the Component Palette tab](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

From the General tab of the Environment Options dialog box, you can specifying the following environment options:

[Defining the Visual Cafe startup mode](#)

[Finding Java source files](#)

[Defining the Help file set](#)

[Automatically opening the Messages window](#)

[Choosing the SDI or MDI window system for the development environment](#)

[Converting between native and ASCII](#)

[Setting the default version control user name](#)

**To open the Environment Options, General view**

- Choose Tools  Environment Options

 General tab.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The startup mode defines what processing you want done when you start a new Visual Cafe session.

1. Choose Tools  Environment Options

 General Tab.

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

Select...	To specify this...
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 Visual Cafe exited; if there is none, a new project is created. (This is the default.)
Do nothing	Specify that no project opens and that you will select an appropriate action after Visual Cafe starts.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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

 See Also

**Note** You can set the source 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.

1. Choose Tools  Environment Options

 General tab.

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

 See Also

**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+ specifies that Visual Cafe searches in c:\myapps and all of its subdirectories.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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

1. Choose Tools  Environment Options

 See Also General tab.

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

 See Also


**Note** These directories are automatically searched by Visual Cafe: the installation directory, and the release \bin and \help directories.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

When this option is selected, when an error message occurs and the Messages window is closed, the window opens.

1. Choose Tools  Environment Options

 See Also General tab.

2. Select or deselect the Automatically Open Messages Window option.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe Professional Edition 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.

1. Choose Tools  Environment Options

 See Also

2.  General Tab.  
Select MDI Development Environment to enable MDI, or deselect it to enable SDI.

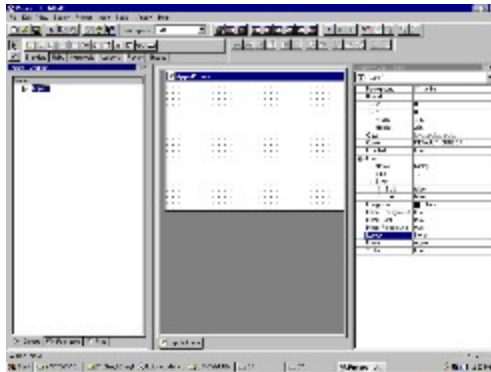
{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

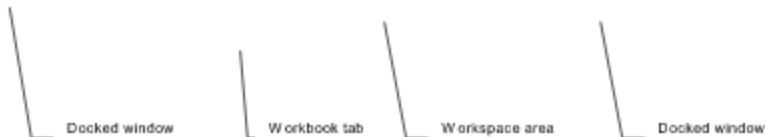
 See Also

Visual Cafe Professional Edition 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 is an example of what MDI looks like.



Expand or collapse  
a docked window  
with this button



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. Here are some of the dockable windows:

Project window	Breakpoints
Property List	Watch
Component Library	Variables
Messages	Calls
Find in Files results	Threads


A dockable window can be docked only if Docking View is enabled.

 See Also

**Note** Docked windows have the same highlight color in the title bar as a regular window that is selected. To make sure that a docked window is the active window, click it.

When using MDI mode, some window positions are saved with the workspace and the project. For more information on setting your workspace options, see [Visual Cafe workspaces](#).

#### To toggle Docking View for a dockable window

- Right-click the window then choose Docking View.
- Activate the window then choose Window  Docking View.


If the window was previously docked and you enable Docking View, the window might dock again.

#### **To dock a window floating in the workspace area**

- While Docking View is enabled, drag the window to the edge of the workspace.

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

#### **To undock a window**

- Double-click the title bar.
- Drag the window into the workspace area.
- Right-click the window body and choose Docking View.
- Activate the window then choose Window  Docking View.

#### **To change the size of a docked window**

- Move the cursor over the edge of the window until the cursor changes, then drag. You can drag the edge facing the center of the main window.

#### **To prevent a dockable window from docking while dragging it**


- Press CONTROL while dragging.

#### **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.

#### **To change the display of windows in the workspace area**

The workspace area contains regular windows, such as the Form Designer or Source window, and dockable windows that are not docked. These dockable windows can have Docking View enabled or disabled.


- Choose Window  Cascade, Tile Horizontally, or Tile Vertically.
- Drag the windows within the workspace area.

#### **To toggle the display of Workbook tabs**

At the bottom of the workspace area, you can display Workbook tabs — one for each open window (either regular windows or dockable windows with Docking View disabled). Clicking a tab activates that window in the workspace area.

- Choose View  Workbook.

#### **To activate a window in the workspace area**

- Click it.
- Click a Workbook tab.
- Choose Window  Next or Previous.
- Choose the window from the Window menu.

#### **To toggle the display of the status bar**

- Choose View  Status Bar.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

There are several Visual Cafe debugger options that apply to the entire environment.

#### To set environment options for debugging

- Choose Tools  Environment Options, then click the Debugging tab.

#### To enable or disable the run-time editing option


Visual Cafe Professional edition offers [incremental debugging](#) (also called run-time editing). While your program is running in the Visual Cafe debugger, you can edit your Java program and immediately see the effects of the code change.

- Select the appropriate option:
  - 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 the same as the first option, except that you are prompted first, so 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 enable or disable **ValueTips** at debug time

1. Select or clear the Enable ValueTips option.
2. If selected, set the delay time before a tip displays.

#### To switch to the Debug **workspace** when running in the debugger


- Select this option to go to the workspace named Debug when you choose Project  Run in Debugger. Deselect this option if you want to remain in the current workspace. The default is selected.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can define a format style for different file types displayed in the Visual Cafe windows. The initial file types that are available are Java and HTML.

To change format options for the current file, choose Source  Format Options.

#### To open the Environment Options Format view

- Choose Tools  Environment Options

 Format tab.

#### To set format options for files with a certain extension

1. 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 it then click Delete.

<UNTITLED> is the same type you get after choosing File  New File.

<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.

2. Set the options you want to apply to files with this extension. To highlight language keywords, choose the language from the Language keywords drop-down list.

Option	Description
Word wrap	Enables word wrap. While typing, lines that extend beyond the right margin are automatically broken at the last word boundary before the margin.
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 is displayed in the status bar.
Indent after brace	If the last character typed 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 in the buffer.
Indent automatically	Automatically indent on newline. When you press ENTER, the editor positions the cursor directly below the first character in the previous line.
Change tabs to spaces	Tabs are inserted into the text as an appropriate number of spaces, rather than as tab characters.
Remove trailing spaces	Trailing spaces and tabs are removed from the end of each line when a file is saved.
Enable custom keywords	You can maintain a set of custom keywords that highlight in a specified manner within the edit window. There is one custom keywords list that applies to all file types that have the Enable custom keywords option. Click Edit Custom to specify custom keywords. To set highlighting for custom keywords, click the Display tab.
Indent comments at	If selected, the editor automatically indents the comment to a specified alignment column when you type "/" or "/" to start a comment. You can specify the alignment column in the adjacent

	text box.
Language keywords	Specify Java, HTML, IDL, MFC, C++, or none.
Tab width	Specifies the number of columns between tab stops (1-16). It is the width of the ASCII tab character. The default is four character widths. This value may be overridden locally in each buffer.
Indent width	Specifies how much text is indented when you press the Tab key or when autoindenting occurs. (Note that it effects editing actions, not the static text.) By default, the tab width and the indent width are the same. However, it is quite common to have them different, for example, an indent width of 4 and a tab width of 8. When the indent width is different from the tab width, code is indented using the most efficient combination of spaces and tab characters.
Right margin	Specifies the column that acts as the right margin for word wrapping. (1-512). This value may be overridden locally in each buffer.

3. Click Apply or OK to save the change.

The change takes effect immediately for the next file you open.

### **To specify custom keywords**

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 Display tab. There is one custom keywords list that applies to all file types that have the Enable custom keywords option.

1. Click Edit Custom.

The Custom Keywords dialog box opens so that you can enter the new keywords.

2. Click Apply or OK to save the change.

The change takes effect immediately for the next file you open.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Environment Options  Format tab

 See Also **New**

Enter the file extension that you want to add to the format type list.

When the extension is available from the list, you can define a format style for the file type. Two predefined file types are Java and HTML.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

**Environment Options**  **Format tab**

 **Edit Custom**

Use the Custom Keywords dialog box to add keywords to the list of words recognized in the Source window and pane.

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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe lets you define a custom keystroke sequence for many Visual Cafe editing operations and macros. Macros are stored in the command list as *macrofilename*. ScriptMaker macro files, stored in \Bin\Macs\Src, appear in the list.


Your settings can be saved in a key file, stored in the \Bin\Keys directory. Saving to a file allows you to reload or distribute the key assignments. In version 2.5 of the Professional edition, a VJ key file was added so you can use Visual J++ keyboard commands.

 See Also

**Note** Help on the commands is available in the Macro help. This help is available from the Help menu and by pressing F1 when you are editing macros.

In the Keyboard view of the Environment Options, you can also specify editing options for the Source Editor.

#### To access the Environment Options Keyboard view

- Choose Tools  Environment Options

 Keyboard tab.

#### To choose a key file to use in the Visual Cafe environment

- Choose a file in the File field.

#### To add a key file to the list

- When you start modifying the assignments, Untitled appears in the list. You can save it under another name.

#### To delete a key file

- Choose a file in the File field, then click Delete.

The file is deleted from your hard disk.

#### To map a command to a key sequence

1. Choose a key file.
2. From the command list, select a command.

If an item has more than one key assignment, the item appears multiple times in the list.

 See Also

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

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

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

 See Also

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

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

The assignments are automatically saved in an untitled file.

5. 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 directory.

#### To delete the key assignment for a command

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

#### To copy the command assignment list as text

- Right-click in the Command list and select Copy All.

You can paste the list into another window, such as the Source window. This is a good way to get a list of your key assignments.

**To modify what commands are shown**

- Right-click in the Command list and choose Unbound commands, All, Member, Text, or Class.

Unbound commands toggles the display of commands without key assignments.

Member, Text, and Class toggle display of commands that start with these words.

**To specify key editing options for the Source window and pane**

- Click More and select options in the dialog box.



[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

These settings apply to all Source windows and panes.

1. Choose Tools  Environment Options

 See Also Keyboard tab.

2. Click More.
3. Select the options you want:

Option	Description
Virtual cursor	When checked, you can position the cursor anywhere in the file, regardless of the placement of the line endings. When unchecked, you cannot position the cursor past the end of a line. This box is unchecked 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.
Brief-compatible selection	If you choose this option, the editor stays in selection mode when you use the arrow keys.
Typing replaces selection	When checked, the source editor follows the Windows standard convention of replacing any selected text with the first character typed or pasted. When unchecked, typing or pasting inserts the text to the left of the current selection. This box is checked 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.
Cut and copy line without selection	When checked, you can quickly cut or copy the current line using the standard cut or copy keystrokes (without having to first select the line). When unchecked, you can cut or copy a line by using the keys assigned to the EditorCutLine or EditorCopyLine functions. This box is unchecked by default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Environment Options  Keyboard tab

 See Also Save

Use the Save Key Binding dialog box to save a **key** file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

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

#### Note

- When you select All Windows, the Color and Style list displays a set of common components. Common component changes apply to all objects that contain the component.
- There is only one Text entry; changing the color and style for it changes the text color for all windows.

#### To access the Environment Options Display view

- Choose Tools  Environment Options

 Display tab.

#### To set the font or font size

1. In the Category area, select the item whose appearance you want to modify.
2. Choose a value for font or font size.

#### To set the color and style

1. In the Category area, select the item whose appearance you want to modify.
2. Select an item in the Color and Styles 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.

Here are explanations of some items:

Item	Description
Text	Text other than the type defined below.
Selected text	The text and background of selected text.
Errors	Lines where compiler errors are found.
Comments	Java comments.
Keywords	Java keywords.
Current Line	The line that contains the insert point.
Preprocessor	Java preprocessor directives.
Custom keywords	Special keywords that you define.
Execution	During debugging, the current execution line.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides several backup and save options:

[Saving files for recovery purposes](#)

[Automating source file backups](#)

[Setting the scope of the Undo Command](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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

1. Choose Tools  Environment Options

 See Also 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 are not 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 08-12-95 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can control whether files in a project are automatically backed up each time that a save is performed. You can also specify the location and name of the backup files.

 See Also

**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. Choose Tools  Environment Options

 See Also

Backup tab.

2. Select Backup files on Save.
3. Select the location and name of the backup files:

Select...	To specify this...
Create BAK file	Create one or more backup files that are named <i>file.bak</i> .
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 BackupFile script	Run your own macro, created with the Visual Cafe macro utility and containing a <b>BackupFile</b> method. All macros are placed in the <i>\VisualCafe\Bin\Macs\Src</i> directory.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The Undo command is available from the Edit menu and has standard Windows functionality. The default number of undoable actions is 100.

1. Choose Tools  Environment Options

 See Also

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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe allows you to change standard editing functionality for Source windows, the Class Browser, and the Hierarchy Editor.

#### To access the Environment Options Editing view

- Choose Tools  Environment Options

 Editing tab.

#### To set the cursor style

In the 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.

- Set the options you want for 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 confirm and multiple selection options for the Class Browser and Hierarchy Editor

- These options set functionality for both the Class Browser and Hierarchy Editor.

Option	Description
Confirm Delete member	Requires confirmation of member deletions.
Confirm Inheritance change	Requires confirmation of inheritance changes.
Multiple selection	Specifies if 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.

#### To set other editing options

- Select the options you want:

Option	Description
Show horizontal scrollbars	Select this option to display a horizontal scroll bar at the bottom of Source windows and panes. (Default: selected)
Show source navigator	Select this option to show the Objects and Events/Methods drop-down lists at the top of the Source window. (Default: selected)
Automatically show code helper	Select this option to make the Code Helper automatically appear when appropriate in the Source Editor. For example, the Code Helper would appear after typing the words import, extends, implements, and new (when followed by a space), and when typing a period. When the option is not selected, the Code Helper appears only when pressing Ctrl+J or by choosing a menu item. (Default: selected)

Enter code helper selection automatically	When this option is deselected and you type something that does not match the current selection in the Code Helper, the Code Helper is canceled without inserting text. If this option is selected, the Code Helper inserts the selected item before canceling. (Default: deselected)
Highlight syntax errors while editing	When selected, basic syntax errors in the code are highlighted in red. Placing the mouse cursor over the red token shows a tooltip with the error message. (Default: selected)
Show full compiler errors	When both this option and Highlight Syntax Errors While Editing are select, the Source Editor shows all the errors in red that would be found when compiling. This option is more useful than just showing syntax errors, but it takes more time and can slow down the editor. (Default: deselected)
Undo cursor movement	When selected, any mouse clicks and cursor movements are saved for the Undo command. When deselected, only editing commands are saved. (Visual Cafe version 2.0 saved cursor movements for the Undo command, while the version 3.0 default is deselected.)
Undo individual keystrokes	When selected, every character you type is undone separately — you must select the Undo command once for each character that is typed. When deselected, all typing since the last cursor movement or other command is treated as one operation and is undone by selecting the Undo command only once. (Default: deselected)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can set the cursor style for the insert and overwrite cursor.

1. Choose Tools  Environment Options

 Editing tab.

2. In the Insert or Overwrite definition area, select or clear the Blinking option.
3. Select the cursor style: Block, Underline, or Vertical Bar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



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

**To float a toolbar**

- Drag the toolbar from the top of the Visual Cafe window onto your desktop.
- Double-click somewhere in the toolbar background.

**To dock a toolbar**

- Drag the toolbar to the top of the Visual Cafe window.
- Double-click somewhere in the toolbar background.

**To hide/show a toolbar**

- Right-click at the top of the Visual Cafe window and select the toolbar name.
- Click the close box on the toolbar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

Visual Cafe lets you use different virtual machines (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.


 See Also

**Warning** 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 Project  Run in Debugger, 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.

For important information on debugging with different VMs and JDKs, see [Tips for debugging different virtual machines and JDKs](#).

#### To access the Environment Options Virtual Machines view

- Choose Tools  Environment Options

 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.
2. Specify the parameters you want. Only the VM Name and the VM Executable parameters are required, and it is recommended that you also supply a VM JDK Source Path.

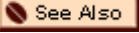

Parameter	Description
VM name	A unique name you want to use for the virtual machine.
VM manufacturer	The name of the creator of the virtual machine. Currently used for informational purposes only.
Compile classpath	Location of the files your program depends on for compilation, for example, the class files for JFC Swing components. The compile classpath is appended to the classpath for the project. If the Compile Classpath field is left blank, the classpath specified in the sc.ini file is used.

 See Also

**Note** You need to add symbeans.jar to this classpath 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 classpath.

VM executable	The virtual machine executable and the path to it.
VM executable	Command-line specifier for the classpath,

classpath prepend string	usually <b>-classpath</b> . The Microsoft virtual machine may use <b>/cp:a</b> (or <b>/cp:p</b> or <b>/cp</b> ). Specify the classpath in the VM Executable Classpath field.
VM executable classpath	The classpath to be appended to the VM Executable Classpath Prepend String parameter.
	 <b>See Also</b>
	<p><b>Note</b> You need to add symbeans.jar to this classpath 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.</p>
VM path	<p>The search path for preparsed JDK information and a <a href="#">Javadoc</a> executable. This location is typically the directory with java.exe.</p> <p>Visual Cafe ships with the standard Java package imports from the Sun Microsystems Java Developers 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.</p> <p>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.</p> <p>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.</p> <p>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.</p>
VM other execution options	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. For example, you might be able to use \JDKx.\bin\JDB.exe, which is a part of the JDK.
VM 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 on the VM path. If the executable is not found, the default Javadoc.exe is used.

Supports direct debugging	<p>Select this option if this virtual machine can be used with a debugger currently available from the Visual Cafe environment or if you want to use the Visual Cafe remote debugging feature to attach to a waiting VM. Then specify a debugger in the Run In Debugger field.</p> <p>If you deselect the Supports Direct Debugging option, when you choose Project  Run in Debugger, Visual Cafe simply executes the Java program in the virtual machine you specified, without launching a debugger.</p> <p>Instead, you can deselect this option and specify another debugger in the Use Debugger Executable field. In this case, when you run in the debugger, Visual Cafe launches this debugger as a separate process. When you choose Project  Run in Debugger, 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.</p>
Run in debugger	<p>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.</p> <p>If you select Default, you can use remote debugging to debug your Java program using the VM you specified. See <a href="#">Debugging remotely</a> for more information.</p> <p>If you specify the default value and run an application, the Applet Viewer is used. If you specify Netscape Navigator and run an application, Visual Cafe instead uses the default virtual machine.</p> <p>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.</p>
VM JDK source path	<p>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.</p>

- Click OK in the Virtual Machine Settings dialog box, select the VM you added, then click OK in the Virtual Machines view.  
The change takes effect immediately.

#### To edit the settings of a virtual machine

- Choose the virtual machine from the drop-down list.
- Click Modify.
- Specify the options you want.
- Click OK in the Virtual Machine Settings dialog box.

The change has been made.

5. When you are finished making changes, in the Virtual Machines view, choose the virtual machine you want to use from the drop-down list, 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

Visual Cafe lets you use different virtual machines (VMs) so you can execute, compile, and debug with them from within the Visual Cafe environment. Here are some debugging tips.

### Debugging JDKs other than the default VM

You can use the remote debugging feature to use the Visual Cafe debugger, as described in [Debugging remotely](#). Or you can execute in an external debugger.

1. Choose Tools  Environment Options

 Virtual Machines tab.

2. Select a VM, then click Modify.
3. Specify options.

Parameter	Description
Supports Direct Debugging	Deselect this option to launch an external debugger, specified in the following field
Use Debugger Executable	You can specify <code>\JDKx\bin\JDB.exe</code> , which is a part of the JDK.
Other Command Options	Specify the remote debug option. You can use <b>java.exe -help</b> to display the remote debugging option.
VM Executable	For JDKs before 1.2 (such as 1.1.x), you need to specify <code>java_g.exe</code> as the VM executable to allow JDB to connect to the VM. If you need to execute your applications with <code>java.exe</code> , you can set up two profiles, one for executing and one for debugging.

4. Click OK in the Virtual Machine Settings dialog box.  
The change has been made.
5. When you are finished making changes, in the Virtual Machines view, choose the virtual machine you want to use from the drop-down list, then click OK.

6. Open the project, and choose Project  Run in Debugger.

`java.exe` launches, then the remote debugging password appears in the console window. Next, the Debugger Options dialog box appears (make sure it is not hidden behind the console window).

7. In the dialog box, type the following then click OK.

**-password xxxx -host 127.0.0.1**

127.0.0.1 is your localhost IP address. If you specified JDB as your debugger, the JDB console window appears. It is connected to the VM that is running your Java program. If the window does not appear, you might have typed in the wrong execution parameters. See the JDK documentation for information on using JDB.

 See Also

#### Note

- Visual Cafe only launches the VM and the debugger executable. If you see errors in the console, it is coming from one of those processes, and not Visual Cafe. When exiting a JDB session, you might see these VM connection exceptions.
- Because JDB and the VM do not always exit cleanly, if Visual Cafe displays “executing” in the status bar, even though there are no apparent VMs running, go to the Windows Task Manager and terminate any `java_g.exe` process that you know is not supposed to be running.
- With some VMs, such as in JDK1.2b4, you might need to turn off the JIT. See the JDK documentation on how to do this.

## Using JDK 1.0.2

Here are a few things to keep in mind:

- You should turn off RAD because the automatic code generation will be in the JDK 1.1 style.
- Remote debugging does not support JDK 1.0.2. Use Visual Cafe version 1.0 to debug.
- The compiler in Visual Cafe enforces the Java specification that requires classes to be in their respective packages/directories. It also will give a compilation error if it comes across a "private protected" declaration, which is no longer valid in JDK 1.1.x.
- If you need full 1.0.2 support, you can use Visual Cafe 1.0, which is on the Visual Cafe CD.

## More pluggable VM tips

Other notes for debugging with a pluggable VM:

- The default debugger uses the default parse information. So when you step into a JDK package, such as `java.awt.Component`, the source for JDK 1.1.5 will show up.
- The "Other Command Options" (in the Virtual Machines tab of the Environment Options) will not be passed into the VM when debugging with the default debugger.
- If you notice that files from the JDK are constantly getting parsed in the Messages window, this means that there was no JDK source directory specified when the parse information was created. See a following section for information on creating the parse information.
- When installing a new JDK, select the option to install the JDK source. You will need it for the preparsed information.
- When switching JDKs, the next time you launch an application or applet you might see the JDK copyright screen. The copyright text is being read from the JDK in the Visual Cafe installation, so it will always read the same version. The VM you specified will properly launch, and has nothing to do with that version.
- Compiling native Win32 only works with the Visual Cafe default VM. If you try to compile with a JDK other than the default, you will get errors.

## Tips on setting up a VM Profile

Here are some tips when setting up a profile:

- Put the `classes.zip` of the JDK/SDK for the profile on the front of the compiler and VM executable classpaths.
- Put `swingall.jar` on the classpath if you use JFC beans.
- The Microsoft VM cannot read jars, so you must extract the classes into a directory, and put that directory on the classpath for the VM executable. For example, you will need to do this if you use any Symantec components from `symbeans.jar`. You can have jars for the compiler classpath since Visual Cafe can read them.
- In the VM profile, do not forget to put in the location of the JDK source path. The next time you launch Visual Cafe, this location will be filled for you in the preparsed information dialog.
- Since the compiler and executable classpath will be the same in most cases, to save time you can copy the entire line from one and paste to the other after making a change.

## Discovering the classpath

If you want to see exactly what the VM uses as the classpath at run time, include this line in your code that will get executed, such as in `init()`:

```
symantec.itools.qa.PropFinder.printDefaultProperties();
```

To see all the system properties, paste this line in your code:

```
symantec.itools.qa.PropFinder.printAllProperties();
```

`\Visual Cafe\java\lib\symclass.zip` must be on your classpath to use this.

To find out what classpath the compiler uses, look in the Messages window after doing a Compile to see what the classpath option is set to.

## Creating preparsed information

You can easily include any additional source files to the preparsed file that is used to display classes in the Class Browser, Insert Class Wizard, and Hierarchy Editor.

When Visual Cafe is launched, it searches for the files `JDK.vep` and `JDK.ve2` in the directory specified in the VM Path field of your Virtual Machine Environment Options. If it cannot find these preparsed files, a dialog box will appear. Here you can specify the locations of the source files you want parsed. Among the directories you need to include is the location of the JDK source you are working with.

If you want to add source files to the preparsed information, just delete `JDK.vep` and `JDK.ve2` from the VM path directory, and relaunch Visual Cafe.




{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you install the Database Development Edition of Visual Cafe, a Database tab is added to the Environment Options dialog box. You use this view to set the default data types for components in the dbAWARE Project and Table Wizards (Choose Database Columns page) and when you drag a column to a project from the dbNAVIGATOR window.

#### **To access the Environment Options Database view**

- Choose Tools  Environment Options



Database tab.

#### **To set a default data type for a component**

1. Click the Component field of any data type to see the drop-down list of component data type options.
2. Choose an appropriate data type for your component.

#### **To return to the default data types**

You can return to the defaults that shipped with Visual Cafe Database Development Edition.

- Click Set Defaults.

Obsolete

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



<<Add this back for NextGen. Need to rework.>> You can customize the toolbars in your Visual Cafe workspaces. This is helpful if, for example, you primarily use certain toolbar buttons when debugging, and use different toolbar buttons when you are designing.

You can either rearrange or remove buttons from current toolbars, or create a new one with your own combination of toolbar buttons. You can customize all toolbars, except for the Workspaces toolbar and the Component Palette. Use the Customize dialog to make changes to your toolbars, or edit them right from the desktop.

To customize your toolbars, you can:

- access the Customize dialog box
- view or hide a toolbar
- create a new toolbar
- add buttons to a toolbar while in the Customize dialog
- add buttons to a toolbar from the desktop
- rearrange buttons on a current toolbar
- set your toolbars back to their defaults


#### To customize toolbars from the Customize Toolbars dialog box

- Choose View  Toolbars
-  Customize.
- Right-click a toolbar and choose Toolbars.

#### To view or hide a toolbar

1. In the Customize dialog box, select the Toolbars pane (if it isn't already selected).  
A listing of available toolbars displays.
2. To make a toolbar visible in the workspace, click the box to the left of the toolbar name. A checkmark appears next to the name of a visible toolbar.  
To hide a toolbar in the workspace, click the box to the left again. The checkmark disappears, and the toolbar hides from view.

#### To create a new toolbar

1. Click View  Toolbars.  
The Toolbars dialog displays.
2. Click New.  
The New Toolbar dialog displays.
3. Enter a name for your toolbar, and click OK.  
The Customize dialog displays with your new toolbar listed in the Toolbars list.  
Another way to create a new toolbar is from the Customize dialog. From the Toolbar pane, click New, and enter the name for your new toolbar. Once you click elsewhere in the dialog your new toolbar shows in the toolbar list.  
You can add buttons to a toolbar by one of two ways. You can add buttons from the Customize dialog, or you can drag buttons from existing toolbars to your new toolbar.

#### To add buttons to a toolbar while in the Customize dialog

1. Open the Customize dialog box for toolbar editing.
2. Make sure your toolbar is visible in the workspace area. You may want to move it closer to the Customize dialog to facilitate editing.
3. In the Customize dialog, click the Commands pane.  
The Commands pane displays a list of standard toolbar button categories.
4. Click one of the four standard toolbar names to access toolbar buttons available in those categories. Click a desired button, and drag it to your toolbar. Your standard toolbars will remain intact.

#### **To add buttons to a toolbar from the desktop**

1. Make sure your toolbar is visible in the workspace area.
2. Alt-click a button to drag it to another toolbar.



**Note** This action moves, not copies, a toolbar button to another toolbar. The toolbar button is removed from the original toolbar.

#### **To rearrange buttons on a current toolbar**

- Alt-click and drag to rearrange buttons right on the toolbar itself. You can do this with standard toolbars and also ones you've created.


#### **To set your toolbars back to their defaults**

- In the Customize dialog box, click Reset.

Your toolbars go back to what they looked like when you first installed Visual Cafe. Any customizations to your toolbars, such as rearranged, removed, or added toolbars, are nullified.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

ValueTips display the value of a variable in the Source window when you place your cursor over the variable at debug time.

1. Select Tools  Environment Options.
2. Click the Debugging tab.
3. Select or clear the Enable Value Tips option.
4. If selected, set the delay time before a tip displays.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can enable multiple component selection and select confirmation options for both the Class Browser and the Class Hierarchy.

1. Choose Tools  Environment Options

 Editing tab


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

Option	Description
Confirm Delete member	Display a confirmation message before deleting a class member.
Confirm Inheritance change	Display a confirmation message before applying inheritance changes.
Multiple selection	Display a confirmation message before applying a change to multiple selected objects. Yes - allow multiple select. No: disallow multiple selects. Confirm: allow but confirm changes to multiple select.



**Note** You can also enable a horizontal scroll bar in the Class Browser source pane by selecting the ShowHorizontal scroll bar option in the Source window area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can change the behavior of the Visual Cafe editors to suit your individual work needs. Global changes can be made with the Environment Options dialog box, Tools  See Also Environment Options. Local changes can be made to an editing session by using the Format Options dialog box, Source

 See Also Format Options.

The following items can be customized:

#### Font Style and Size

Sets the appearance of text in Source windows.

#### Colors

Both foreground and background color for the selected window can be customized. You can also preset the colors for certain text items. The items for which you can change the color include: breakpoint text, code window text, comment text, current statement text, custom keyword text, error text, identifier text, keyword text, next statement text, and selection text.

#### Tab to Space Conversion

Converts tab characters to spaces as you type. You can also preset the number of spaces used for each tab character.

#### Automatic Source Formatting

Automatically formats code as it is entered, indenting lines of code and adding closing braces and parentheses.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can use the 2.1 version of the Visual Cafe menus instead of the latest menus. For example, Visual Cafe version 2.1 did not have a View menu, whereas later versions of the Professional edition do have one. Some items from the Window menu were moved to the View menu (menu items that opened a specific type of window, such as the Component Library menu item). You can specify that you want to use the old menu system, so that all View menu items appear in the Window menu instead. One reason you might want to do this is if you have scripts that rely on the older version of the Window menu.

1. Choose Tools  Environment Options

 General tab.

2. If you want to use an earlier version of Visual Cafe menus, select Use 2.1 Menus. Otherwise, deselect it.

## About Optlink Error Messages

This section lists warnings, plus non-fatal and fatal error messages generated by OPTLINK. When appropriate, any probable cause and/or remedy is briefly discussed. The DOS return code and segmented `.exe` error flag is set upon OPTLINK encountering either a Non-Fatal or Fatal error. Warnings have no effect on the code or flag. See also `/ONERROR` and `/DELEEXECUTABLE`.

## How OPTLINK Errors are Displayed

Each warning and error message, with added information such as the symbol name or file name involved, is displayed to the STDOUT device and added to the `.map` file (if selected) when its triggering condition is detected. If a log of errors is desired, DOS redirection may be used.

The following rules apply when displayed: If the file name is known, it is printed. If the module name is known, it is printed (useful especially if the file name is `.lib`). If the error occurs at a known offset in the file, the offset (in hexadecimal) is printed. If a specific object record is being worked on, its record type (a number in hex corresponding to the standard Intel OMF or some extension) is printed. And in every case, an error number (not significant since these numbers may vary between OPTLINK versions and/or other linkers) is printed followed by the text.

## Warning Messages

OPTLINK provides several warnings, which alert the operator to an inconsistency rather than diagnosing a known error. These messages indicate code that may not execute as you expect but that does link. OPTLINK completes its operation and the output file may be usable.

### ALIAS Previously Defined

An attempt to alias an already-defined symbol was detected. The `ALIAS` directive is ignored.

### Bad Checksum

A checksum error was detected in an input file. Perhaps the file has been corrupted.

### Below 100H Cannot Be Initialized

While creating a `.com` file, an attempt to initialize data areas at an address lower than 0100h was detected.

### Cannot Allocate EMS Blocks

An EMS allocation call failed. The EMS driver has provided inconsistent information. OPTLINK will use alternative memory space instead.

### Cannot Generate Segmented .COM or .SYS File

A `.com` or `.sys` extension was specified as the output file name when a `.def` file had also been supplied.

### COMMON Combine type overrides others

Multiple combine types were specified for a given named segment. In such a conflict, the `COMMON` combine type overrides all others, and the multiple segments will be combined into a single segment.

### Duplicate RESOURCE, Ignored

A second resource was found within the `.res` file with the same name and type as one already processed. The second resource is ignored.

### File Not Found

The specified `.lib` file could not be found in order to complete command operations. If referring to an `.obj` file or an indirect command file, this indicates a fatal error, described later in the fatal error section.

### Heap without DGROUP

Heapsize was defined with no `DGROUP` or zero length `DGROUP`. Causes Windows to crash.

### Grouped Segments Have Conflicting Flags

Segments in a single group have different flag settings declared via the `SEGMENTS` directive in the `.def` file. Since a group is a single physical segment in protected mode, all segments that make up the group must have the same flags.

This warning may be caused by the fact that any non-default flags override the default settings. That is, a `MOVABLE` segment overrides a `FIXED` segment, and the whole group becomes `MOVABLE`.

### LINNUMs in Non-CODE Segment

A `LINNUM` statement was encountered within a data segment.

### Missing LIBRARY in .DEF File for .DLL

A `.dll` extension was specified for the output file, but no `LIBRARY` directive appears in the `.def` file. The `.dll` is created with default assumptions.

### No Stack

No stack segment was supplied. If a `.com` or `.sys` file is being generated, it is normal; in other cases, be careful.

### **No Start Address**

The end of the last `.obj` file was reached and start address was not detected. In assembler programming, start addresses are specified by putting a label after the `END` statement. In high level languages, perhaps the start up modules were not supplied.

### **Segment Already in Different Group**

OPTLINK detected a second attempt to assign a segment to a group. Each segment can belong to only one group.

### **Stack Already Declared in Module**

OPTLINK detected a second attempt to define a stack segment. Each program can have only one stack segment. The warning is followed by the name of the module that first declared a stack.

### **STACK Combine type overrides others**

Multiple combine types were specified for a given named segment. In such a conflict, the `STACK` combine type overrides all others, and the multiple segments will be combined into a single segment.

### **Start Previously Specified in Module**

More than one start address was specified for a program. Start addresses are specified by putting a label after the `END` statement. Only one module should specify a start address. If OPTLINK finds more than one, it uses the first one encountered.

### **Symbol in this LIB defined elsewhere**

Warning message generated from `/WARNDUPS`.

### **Too Many Segments, Trying PACKCODE**

The segmented `.exe` format can hold only 254 segments. This warning appears when more than 254 segments exist and

`/PACKCODE` was not specified. OPTLINK automatically invokes

`/PACKCODE` in an attempt to reduce the segment count to a number below 255.

### **Too Many Segments, Trying PACKDATA**

The segmented `.exe` format can hold only 254 segments. This warning appears when more than 254 segments exist after OPTLINK attempted to `/PACKCODE`, and `/PACKDATA` was not specified. OPTLINK automatically invokes `/PACKDATA` in an attempt to reduce the segment count to a number below 255.

### **Unknown Option**

OPTLINK encountered a misspelled, incomplete, or unsupported option switch.

### **Unsupported GRPDEF Type**

An invalid GRPDEF record was encountered. The bad record is ignored.

## Non-Fatal Error Messages

These are errors which may make the output file unusable, but did not prevent OPTLINK from completing successfully. Processing continues if any of these errors are detected, but the output file that is generated should not be used.

### **.EXE Header > 64k**

The resident portion of a .dll or .exe header approaches or exceeds 64Kb. Windows loads this header into a single segment; therefore, it must be smaller than 64Kb. The sizes of the various pieces that make up the header are listed. You probably need to import/export by ordinal to reduce name table sizes.

### **Bad COMDEF Sizes**

An size error was detected in a COMDEF record.

### **Badly Formed Segment Size**

The BIG bit was set and segment size was not zero.

### **BYTE Out of Range**

Code such as:

```
mov al, ext_abs
```

where *ext\_abs* is greater than 255, was encountered.

### **Cannot Export**

Symbol cannot be exported. It may be either undefined or a constant. Symbols must have an associated segment to be exported.

### **Cannot Reach TARGET from FRAME**

A traditional "fixup overflow" was detected by OPTLINK. Usually caused by: memory-model conflicts, a segment or group being too large, segment order or combining errors (look at the map file output to diagnose this), incorrect `ASSUME` statements, trying to access a variable when no segment register points to it, or trying to do a `NEAR CALL` / `NEAR JMP` to a `FAR` routine.

### **CEXTDEF With No COMDAT**

The end of an object file was reached and a `CEXTDEF` item remains unresolved.

### **COMDAT Continuation Mismatch**

A `COMDAT` continuation record appears before the `COMDAT` definition to which it refers. If you get this error, please contact Symantec Technical Support.

### **COMDAT Syntax**

Due to either corrupt files, translator errors, internal errors in OPTLINK, or unknown enhancements to the OMF conventions. If you get this error, please contact Symantec Technical Support.

### **Constants Must Have FRAME=0**

A constant having a `FRAME` value other than zero was detected. While not prohibited in Intel's OMF format, this is not valid in the Microsoft version.

### **CODE Directive**

A syntax error was detected in a `CODE` directive.

### **DATA Directive**

A syntax error was detected in a `DATA` directive.

### **Data Outside Segment Bounds**

OPTLINK found more data in an object module than expected. For example, an object file that indicates 5 bytes of data are to follow, and then provides 6, will cause this error. Most likely cause is that the object file has been corrupted, or that a translator error has occurred.

### **Data Overlaps Relocations**

Displayed while the `/RELOCATIONCHECK` option was in effect and an overlap was detected. Probable causes include initialization of the `COMMON` segment in more than one module, use of the `ORG` statement to back up over existing code, or a translator error. The problem must be resolved, or all other addresses referenced in the output file may be wrong since relocation information is often stored as in-place linked lists.

### **Delimited String Expected**

OPTLINK failed to find a delimited string required by the directive syntax being parsed.

### **DGROUP + Stack + Heap Exceeds 64K - 16**

The total sizes of Heap, Stack, and DGROUP exceed 65,520 bytes. Try reducing the size of the stack or the heap, or use `HEAPSIZE MAXVAL`.

### **Duplicate OLD**

A second `OLD` directive was encountered.

### **Duplicate ORDINAL Number**

The ordinal number specified for this symbol was already specified for use by another symbol. Two exported symbols cannot have the same ordinal number.

### **Duplicate RC Commands**

Both the `RC` directive in the `.def` file and the `/RC` option switch were supplied. Eliminate one of the two to correct the error.

### **Duplicate STUB**

A second `STUB` directive was encountered.

### **EXETYPE Directive**

A syntax error was detected in an `EXETYPE` directive.

### **EXPORTS Directive**

A syntax error was detected in an `EXPORTS` directive.

### **\_Export Conflicts With EXPORTS**

OPTLINK detected a symbol with a different name or parameter list size in the `.def` file than was present in the `.obj` file. They should agree.

### **FIXUPP Points Past Data Record**

OPTLINK detected a situation such as a `FIXUPP` record instructing that the doubleword beginning at byte 5 of a data record be changed, when the data record contains only six bytes (since a doubleword beginning at byte 5 would extend through byte 8). Most likely, the `.obj` file has been corrupted or a translator error occurred.

### **GROUP Cannot Be Both Relocatable and Absolute**

A segment defined as `SEGMENT AT` was placed within a `GROUP` with a normal relocatable segment.

### **Group Cannot Contain CODE and DATA Segments**

Segments in the same group have different type flag settings, via the `SEGMENTS` directive in the `.def` file or their class names. Since a group is a single physical segment in protected mode, the type for the whole group must be either code or data. It cannot contain both types.

### **Group Size Exceeds 64k**

The total size of all segments named within a Group exceeds 65,536 bytes.

### **Groups Overlap**

An overlap of groups was detected. In protected mode, groups are logical segments, and cannot be

permitted to overlap. To reorder segments in such a way that the overlap is eliminated, use `/REORDERSEGMENTS`.

### **HEAPSIZE Directive**

A syntax error was detected in a `HEAPSIZE` directive.

### **Illegal Frame On Start Address**

Detected a `FIXUPP` error on the start address. Check the module defining the start address.

### **Illegal Start Address**

An absolute address was used as a start address in relocatable mode. Correct the address.

### **IMPORT Illegal As Start Address**

The start address was specified as a symbol imported from another library. This is illegal. The start address must be in the `.exe`, not in a `.dll`.

### **IMPORTS Directive**

A syntax error was detected in a `IMPORTS` directive.

### **LOCATION Not Within FRAME**

A `CALL` or `JMP` was made and the `CS ASSUME` is not valid.

### **Library Error**

A library problem was detected. The most likely cause is a corrupted library dictionary.

### **LINSYM Before COMDAT**

A LINSYM record points to a non-existent COMDAT record. Most likely, caused by either an invalid forward reference or a translator error.

### **Multiple Descriptions**

More than one `DESCRIPTION` directive was encountered in a `.def` file.

### **Only One NAME or LIBRARY Allowed**

More than one `NAME` or `LIBRARY` directive was detected in a `.def` file.

### **Preload Segment + Relocs > 64k, Use -k**

A segment marked for preloading, plus its relocation information, exceeds 65,536 bytes in size. The fast-load section of a Windows `.exe` file cannot accept segments larger than this. You can change the segment to movable, discardable, loadoncall, or disable segment sorting with `-k`.

### **Previous Definition Different**

`PUBLIC` symbol was redefined and the second definition differs from the first.

### **PROT vs REAL Conflict**

A conflict between real mode and protected mode syntax was encountered.

### **PUBDEF Must Have Segment Index**

`OPTLINK` encountered a `PUBLIC` symbol that had no reference into a defined `SEGMENT`. Most likely cause is a corrupt object file, due either to a disk accident or translator error.

### **Relocatable Bases Not Allowed in Absolute Mode**

Reference to relocatable data was encountered while processing code in Absolute Mode. That is, the `.obj` module(s) cannot reference segments or groups, cannot do far jumps or calls, and cannot do `DD LABEL`, when generating a `.com` or `.sys` file, unless the target segment has been declared to be absolute (`SEGMENT AT`).

### **Segment >= 4G**

The total size of a 32-bit segment exceeds 4 gigabytes ("4G").

#### **Segment Size Exceeds 64k**

The total size of a segment exceeds 65,536 bytes ("64K").

#### **SEGMENTS Directive**

A syntax error was detected in a `SEGMENTS` directive.

#### **Short JMP Out of Range**

The destination of `short JMP` instruction was more than 125 bytes from the location of the instruction.

#### **Start Address Must Be 100H**

The `.com` file option has been selected and a starting address (`END start_addr`) other than 0100h was also specified.

#### **Symbol Undefined**

A symbol remained undefined after all input files, including libraries, had been processed.

#### **Too Many ENTRIES**

More than 65,536 entries were specified.

#### **Too Much DEBUG Data for Old CodeView Format**

OPTLINK detected more than 64KB of debug information per type per module, and the CodeView option has been enabled for a CodeView version prior to 4.0.

#### **Unknown COMDAT Allocation Type**

Either corrupt files, translator errors, internal errors in OPTLINK, or unknown enhancements to the OMF conventions. If you get this error, please contact Symantec Technical Support.

#### **Unrecognized data in .CFG**

Syntax error within `optlinks.cfg`.

#### **Unrecognized FIXUPP Type**

Most likely cause is the object file has been corrupted, or a translator error has occurred.

#### **Unrecognized Record**

A record in an object file cannot be recognized as one of the defined record types. The unrecognized record is skipped and linking continues.

#### **Weak EXTRN Different**

A "weak `EXTERN`" declared with a different default in another module was encountered.

#### **WeakLazyAlias Internal**

A specific internal error condition was detected in OPTLINK. If you get this error, please contact Symantec Technical Support.

## Fatal Error Messages

The following errors prevent a successful completion of OPTLINK, processing stops immediately upon detection. An executable or possibly a partial executable may have been created. This file is unusable and should be deleted.

### **.DEF Syntax Error**

An invalid `.def` file syntax could not be parsed.

### **.RES File Corrupt**

The compiled resource (`.res`) file was found to be corrupt. To correct it, rebuild the file using the `rc.exe` resource compiler and its `/r` option switch.

### **APpload Must Appear Before Any Segments**

A segment precedes the `APPLoader` directive in a `.def` file.

### **Bad FIXUPP Thread**

A `FIXUPP` thread was referenced prior to being defined. The `.obj` file is corrupt and must be regenerated. This condition can only be caused by a translator error.

### **Bad LOC Frame in Start Address**

An undefined `FIXUPP` frame was referenced in a start address. The object file is corrupt and must be regenerated. This condition can only be caused by a translator error.

### **Bad OMF Extension**

The format of a type `A0 COMMENT` record cannot be recognized.

### **Bad STUB File**

An error was detected while parsing the specified `STUB` file. Indicates an invalid DOS `.exe` file.

### **Cannot Create File**

A requested output file cannot be created. Most probable causes are the requested file name contains an invalid character, or the output disk directory is full (applicable only to root directories).

### **Cannot parse Class record**

Corrupt debug information.

### **Cannot Reopen Output File**

A file access error was encountered, probably due to a sharing violation on the output file.

### **Circular ALIAS**

A chain of aliases references itself.

### **Colon Expected**

An OPTLINK option that requires a number is requested, and the ":" or following number is not given. Try the operation again, being sure to supply all required numbers.

### **Dictionary Exceeded 512k**

Library dictionaries have a size limit of 512Kb, which is unlikely to ever be reached unless the `/IMPLIB` switch is used incorrectly.

### **Disk Full Writing**

OPTLINK failed to find enough space on the disk to write the required output (or virtual memory) file.

### **DOS Critical Error**

DOS detects a "critical error" while processing a read or write function for OPTLINK.

### **DOSSETFILEINFO Error**

An OS/2 critical error occurred.

### **EMS Error**

OPTLINK was notified of an EMS error while attempting to access expanded memory. The only known causes for this error are faults in either the EMS driver or the EMS hardware.

### **EXPORTed Names Text >64k**

The exported names text exceeds the limit. Try using the `/NONAMES` link option.

### **Filename Expected**

OPTLINK failed to encounter a required file name as input.

### **File Not Found**

Commands were given to perform an operation with a object file which cannot be found.

### **Illegal Filename**

A file name cannot be parsed into a valid format.

### **Illegal Record Syntax**

OPTLINK was unable to decipher the syntax of a record in an object file. The most likely cause is that the object file is corrupt, due either to a disk accident or to a translator error.

### **/IMPLIB Needs Larger /PAGESIZE**

Set the `/PAGESIZE` link option to a larger number.

### **Incremental Compile Error**

A record was detected that says the `.obj` file being processed is corrupt. Compilers such as Microsoft Quick C may create such records upon detecting certain errors during a incremental compile.

### **Index Range**

A record in the object file provides an index value which is outside the allowable range of values. The most likely cause is that the object file is corrupt, due either to a disk accident or to a translator error.

### **Indirect File Nested Too Deep**

The nesting level for indirect files exceeds 10.

### **Limit of 65535 Base FIXUPPs Exceeded**

Attempted to created more than 65,535 relocation entries for a real-mode executable. This limit is imposed by the structure of the `.exe` file.

### **Module or Dictionary Corrupt**

When attempting to read a `.obj` or `.lib` module, the first record was invalid. This error can be caused by passing some other file mistakenly as a `.obj` or `.lib` (syntax error). It also can be caused by invalid library dictionaries built by incompatible tools.

### **Need Larger /ALIGNMENT Value**

The segmented (Windows or OS/2) executable is larger than  $64KB * ALIGNMENT$ . Alignment is the page boundary in the `.exe` or `.dll` on which all segments will start. These segments are referred to elsewhere by a 16-bit page number. OPTLINK defaults to an `ALIGNMENT` of 16 (limits an executable to 1MB) to reduce wasted space between segments, while other linkers normally waste space by defaulting 512. To fix the problem, increase the `ALIGNMENT` size.

### **No Segments Linked!**

No input files could be processed. Therefore, no file could be created.

### **Not a Valid Library File**

A `.lib` file fails to meet the criteria for `.lib` files.

**Number Overflow**

A number greater than 65,535 (16 bits) was supplied on the command line.

**OBJ Record Too Long**

An object record exceeds 6000 bytes in length. The `.obj` or `.lib` file is probably corrupt.

**Out of Memory**

OPTLINK does not have enough memory to continue operation. OPTLINK needs to have a minimum of 200KB available. Try removing a TSR or two, and link again, or (if you are using a makefile) link from outside the MAKE program.

**Overflow 32-bit Multiply**

OPTLINK's far global data allocation routines encounter an address value which they cannot successfully calculate. While it is documented here for completeness, you are not likely to see this message.

**Selector Already Released**

As a result of an internal processing error within OPTLINK. If you get this error, please contact Symantec Technical Support.

**Swap File Full**

Virtual memory is exhausted. Delete any unnecessary files from your disk and try again.

**Token Too Long**

A symbol name or other token exceeds the 468-character limit. The `.obj` or `.lib` file is probably corrupt. The too-long symbol is appended to the message.

**Too Many Relocs to EXEPACK**

More than 8,191 segment relocations per 64KB of data were involved while performing `EXEPACK` on real-mode executable.

**Too Many Segments For Segmented .EXE Format**

The segmented `.exe` format can hold only 254 segments. Number of segments has been exceeded, even after `/PACKCODE` and `/PACKDATA` have been used. Try using `.dll`'s to contain some segments and thus reduce the size of this module.

**Too Much EXESTR Data**

More than one megabyte of `EXESTR` data (created by a MSC6.0 pragma) exists. OPTLINK supports up to one megabyte of `EXESTR` data.

**Unexpected End of File**

OPTLINK encountered the end of an input file before finding a `MODEND` record. The most likely cause is that the object file is corrupt, due either to a disk accident or to a translator bug.

**Unknown FIXUPP Frame Type**

OPTLINK was unable to determine the frame type of a `FIXUPP` record in an object module. Most likely cause is that the object file has been corrupted, or that a translator error has occurred.

**Unrecognized B2 Record**

OPTLINK found a type B2 record in its input files and was unable to determine the meaning of the record. Most likely cause is that the translator has added an undocumented extension to the B2 record usage.

**Unrecognized Communal Syntax**

OPTLINK detected an error while processing a type B0 (Communal Data) record.

**Unrecognized Record**

OPTLINK cannot recognize the type of a record in an object file.

**User ABORT**

The user pressed Control-C or Control-Break to abort the link operation.

**XMS Error**

OPTLINK was notified of an XMS error while attempting to access extended memory. The only known causes for this error are faults in the XMS driver.

## SMAKE Error Messages

This section lists and describes error messages produced by the SMAKE utility.

When SMAKE encounters an error, it prints a message to the screen describing the error and then returns to the operating system.

SMAKE error messages begin with the text "SMAKE fatal error:" followed in most cases by the name of the makefile and the current line number in the makefile, followed by one of the messages described below.

### **access denied**

DOS could not perform the action specified in a command block.

### **can't create response file *filename***

SMAKE could not find or open response file *filename* as specified on the command line (using the @ option).

### **can't create temporary file *filename***

SMAKE could not create the temporary file *filename*.

### **can't execute command**

In a command block, the specified (DOS) command could not be executed.

### **can't nest response file**

SMAKE was passed a response file (using the @ option) that included another @ command.

### **can't open error file *filename***

SMAKE could not open the error file *filename*.

### **can't open include file *filename***

SMAKE could not open the #include file *filename*.

### **can't open makefile *filename***

SMAKE could not open the makefile *filename*.

### **can't reopen *filename* for input**

SMAKE could not reopen the makefile *filename* after closing a #included file.

### **command *expression* returned with error code *number***

In a command block, the command *expression* failed; it returned error code (ERRORLEVEL) *number*.

### **directory not found**

In a command block, the specified (DOS) directory was not found.

### **don't know how to make *filename***

SMAKE did not have enough information to make target *filename* based on the makefile.

### **error in macro substitution syntax: *identifier***

The macro *identifier* uses internal substitution (for example, \$(XYZ:a=b) incorrectly.

### **exceeded max nesting level for conditionals**

Conditional directives (for example !IF) within the makefile were nested greater than 32 levels deep.

### **extension *ext* too long in rule**

In a rule, SMAKE encountered a file extension *ext* that was too long.

### **file not found**

In a command block, the external (DOS) command was not found.

### **illegal operator and/or operand in expression**

A conditional expression was not written properly.

**illegal string expression**

On the specified line, a string expression within a conditional statement used invalid syntax.

**invalid option** *number* in parameter *number*

The *number* th option in the *number* th parameter on the SMAKE command line was invalid.

**invalid parameter** *number*

The *number* th parameter on the SMAKE command line was invalid.

**not enough memory for command**

In a command block, there was not enough memory to execute the specified (DOS) command.

**out of memory**

SMAKE could not allocate enough memory to continue.

**parameter** *number* requires a filename

The *number* th parameter on the SMAKE command line requires a file name argument.

**special macro** *expression* is undefined in this context

The predefined macro *expression* was used incorrectly.

**syntax error:** *expression*

On the specified line, *expression* contained a syntax error.

**text found after !ELSE**

SMAKE encountered text on a line following a !ELSE directive.

**too many rules or blocks for target** *filename*

Target *filename* was associated with too many rules or blocks.

**unexpected directive**

SMAKE encountered an invalid conditional directive (one beginning with a ! character).

**unknown error**

An unexpected error occurred.

**unknown error** *number* executing command

In a command block, a command returned an error code *number* that SMAKE could not interpret.

**unmatched quotes in command** identifier filename

SMAKE was passed a response file (specified using @) that contained unmatched double quotation marks (").

## SMAKE Overview

SMAKE lets you automate program maintenance, instead of typing in commands to compile and link your program. You create a makefile, which describes how all the files in your project depend on each other and how to update a file (such as an object file) when the files it depends on (such as source files) change.

SMAKE is particularly useful for programs that have many source files, since it recompiles only those source files that have changed. It does this by looking at the date and time stamp of each relevant file before performing any tasks.

**Note:** When you use SMAKE, always be sure your PC displays the correct date and time. Be especially careful when logging on to networks that change the time on the local machine, in case the network time is incorrect.

SMAKE differs from a batch file in that it performs only the commands required to rebuild your program, while a batch file executes all tasks, whether or not they are necessary.

## Writing Makefiles

A makefile is a text file that tells SMAKE how to build your program. By default, SMAKE expects the makefile to be named `MAKEFILE`, with no extension. However, makefiles can have any valid extension (for example `.mak`).

The makefile contains one or more dependence lines, each followed by one or more command lines. Each dependence line specifies a target (upon which the associated command lines act), followed by a colon and a (possibly empty) list of dependent files (files needed to create the target). For example, this dependence line:

```
app.exe : app.obj app.res app.def
```

tells SMAKE to perform the commands that follow if any of

`app.obj`, `app.res`, or `app.def` have a more recent date and time than `app.exe`.

Command lines tell SMAKE how to create the target. Command lines must begin with a tab or space character. For example:

```
app.exe : app.obj app.res app.def
    link app, ,NULL, lib, app
    rcc app.res app.exe
```

tells SMAKE to run the linker and the resource compiler.

A makefile can also contain comments, macros, preprocessing directives, operating system commands, and inference rules.

## Example of a Simple Makefile

This makefile creates a program that has only one source file.

```
hello.exe: hello.obj          # Dependence line 1
        link hello.obj        # Command line 1
hello.obj: hello.cpp          # Dependence line 2
        sc -c hello.cpp       # Command line 2
```

**Note:** You can put comments in your makefile. A comment begins with the # character and continues until the end of the line.

The first dependence line indicates that `hello.exe` depends on `hello.obj`. The first command line specifies how to make `hello.exe` by executing `link hello.obj`.

The second dependence line indicates that `hello.obj` depends on `hello.cpp`. The second command line specifies how to make `hello.obj` by executing `sc -c hello.cpp`.

Suppose you just finished editing `hello.cpp` and typed `smake` to create a new version of your program. SMAKE reads the dependence lines in your makefile. It notices that `hello.cpp` is newer than `hello.obj`, so it re-creates `hello.obj` by executing `sc -c hello.cpp`. Then it notices that `hello.obj` is now newer than `hello.exe`. It re-creates `hello.exe` by executing `link hello.obj`.

## Makefile Syntax

**See Also**

The syntax for a basic makefile is:

```
target-file : dependent-file1 dependent-file2 ...  
    command 1  
    command 2  
    . . .
```

SMAKE assumes that if a target file is older than any of the files needed to create it, then it must re-create the target file by executing the command lines that follow the dependence line.

Target files are typically executable files or libraries, but they can also be labels for sets of commands (like `all :`); these are often called pseudo-targets. Similarly, a dependent file is usually a C or C++ source file, but can be a pseudo-target. For more information see "Specifying targets."

**Note:** SMAKE reads all the dependence lines in a makefile before executing any of the associated commands. It then compares the date and time of the dependents to the target and executes the commands for any target that is out of date. It also checks to see if that target has other dependents that might be more recent.

**See Also**

[Specifying What Targets to Build](#)

## Dependence Line Syntax

The name of the target file must be the first thing on the dependence line, with no white space preceding it. The target and subsequent dependents are separated by a single colon, which can optionally be preceded by tabs or spaces. A dependence line can span multiple lines; all lines to be continued must end with a backslash (\) character.

### Specifying targets

The dependence line can contain one or more target names separated by one or more spaces. Each target can be a fully qualified path and file name, or just a label. However, if the last target has a single character name (with no extension), at least one space is required before the colon to distinguish it from a drive letter.

To build a complicated target that requires more than one set of dependence lines and commands, use double colons (::) to separate it from the dependents in each case. For example:

```
app.exe :: app.obj app.def
    link app, , NULL, lib, app
app.exe :: app.res
rc app.res app.exe
```

If the same target, followed by a single colon, appears on two consecutive lines, and commands follow only one of those instances, SMAKE interprets this as a single, cumulatively combined dependence line. For example:

```
target.exe : file_1.obj
target.exe : file_2.obj
# is interpreted as
# target.exe : file_1.obj file_2.obj
```

If the same target, followed by a single colon, appears on two different lines in various places in the makefile, and commands follow only one of those instances, SMAKE interprets this as if the dependence lines were adjacent. This means that SMAKE does not apply an inference rule (see "Inference Rules") to the instance without commands. Use double colons (: :) after the target to avoid this side effect.

If a target is not specified on the SMAKE command line, SMAKE uses the first target it encounters in the makefile. SMAKE always interprets a label used as a target name as being out of date; this is useful to avoid having to pass several targets on the command line. For example:

```
all: app.exe app2.exe app3.exe
```

### Specifying dependent files

The names of zero or more dependents can follow the target on the dependence line. As with targets, a dependent can be a fully qualified path name or a label, although it is usually a file that is used to build the target. Multiple dependents must be separated by one or more spaces or tabs.

A search path can precede a dependent file; enclose the path in curly braces ({} ) and separate the directories with semicolons (;). SMAKE will search the specified directories in order for the dependent. (The search path applies only to the dependent it proceeds.) For example:

```
app.exe : app.obj app.res \
    {c:\project;c:\default} app.def
```

## Special Characters in Makefiles

SMAKE accepts the special characters listed below in makefiles. These characters have special meaning depending on their context.

For any of these characters to be interpreted literally in a makefile, you must precede it with a caret (^) character. If ^ appears in front of a character that does not have special meaning to SMAKE, it is interpreted literally.

**Note:** You can use ^ at the end of a line to include a carriage return/line feed sequence in a macro or string.

### Special characters in makefiles

Character	Meaning
:	Separates target from rest of dependence line.
#	All characters following a # on a line are interpreted as a comment and ignored.
( )	Encloses the value string of a macro to be expanded.
\$	Precedes an instance of a macro to be expanded.
^	The next character is interpreted as a literal character.
\	Continues a line on the next line.
{ }	On a dependence line, encloses a path name associated with a dependant.
!	As the first character on a line, recognized as the start of a preprocessing directive. On a dependence line, acts as a command modifier.
@	On a dependence line, acts as a command modifier.
-	On a dependence line, acts as a command modifier.

### Using % and \$ in commands

SMAKE interprets a percent sign (%) as the beginning of a file name specifier. To place a percent sign in a command, use two percent signs (%%). Similarly, to use a dollar (\$) sign in a command, specify two consecutive dollar signs (\$\$). You can also specify multiple literal % and \$ characters this way. For example, SMAKE interprets \$\$\$\$ in a command as two consecutive dollar signs.

Because SMAKE interprets the \$ and @ symbols as special characters, it is recommended that you do not use these symbols in file or directory names that will be processed by SMAKE.

### Comments in makefiles

SMAKE interprets all characters following a # on a line as comment characters, and ignores them. Comments need not start at the beginning of a line; they can appear almost anywhere, including at the ends of lines defining macros, dependence lines, and inference rules. Comment are also permitted in command blocks, but cannot be on the same line as the command itself, even if the command spans multiple lines.

## Running SMAKE

When you run SMAKE (from an operating system command prompt), it prints your commands as it executes them. If you run SMAKE and it doesn't need to update your files, it lets you know like this:

```
Target is up to date
```

You can redirect the output of SMAKE with the greater-than sign, like this:

```
SMAKE > log
```

**Note:** When SMAKE calls another program, it redirects that program's output to the log file only if that program writes to `stdout`. The tools that come with Symantec C++, such as SC and OPTLINK, write to `stdout`.

## SMAKE Command Line Syntax and Command Options

### SMAKE Command Line Syntax

The syntax for the SMAKE command is:

```
SMAKE {options} {macros} {targets}
```

The command line arguments for SMAKE are:

#### SMAKE command line arguments

Argument	Description
options	Zero or more command line options used to control the current SMAKE session. If no options are specified, SMAKE uses the options specified in the <code>tools.ini</code> file. If a required option is not specified, SMAKE uses the default value for that option specified in <code>tools.ini</code> .
macros	One or more macros of the form <code>macro=text</code> . A macro defined on the command line overrides any corresponding macro definition in the makefile.
targets	The targets to build.

#### Command line options

SMAKE options are preceded by either a dash (-) or slash (/). Case is irrelevant when specifying options. If a required option is not specified, SMAKE uses the default value for that option specified in `tools.ini`. The following options are valid:

#### SMAKE command line options

Option	Description
/A	Rebuilds all targets, including targets that are not out of date with respect to their dependents. Unrelated targets will not be rebuilt. See also /B, below.
/B	Rebuilds a target if its date and time are identical to the date and time of the dependent. Because most operating systems assign file date and time stamps to a resolution of two seconds, commands that execute very quickly might not result in a different date and time between a target and its dependent. Therefore, SMAKE might conclude that a target is current when it is not. Though this option can result in unnecessary build steps, it is recommended for use on very fast systems. See also /A, above.
/C	Suppresses the default SMAKE output, which includes any nonfatal error messages, warning messages, date and time information, as well as the SMAKE copyright message. If both the /C and the /K options are specified, /C suppresses the warnings issued by the /K option. See also /K.
/D	Displays extra information during the SMAKE session. SMAKE displays the date and time of each target and its dependent when they are evaluated during the build, and outputs a message when a target does not yet exist. The names of the dependents for a target precede the target itself, and are indented. The /D, /N, and /P options are

useful for debugging makefiles. See also /N, /P.

- /E Causes the settings for environment variables to override the macro definitions within the makefile.
- /F *filename* Specifies *filename* as the name of the makefile to be used. White space consisting of zero or more spaces or tabs can precede the *filename*. Using a dash (–) instead of a filename causes SMAKE to get the makefile input from the standard input device. Keyboard input is then ended by typing either F6 or Ctrl-Z. SMAKE can accept more than one makefile; use a separate /F specifier for each makefile. If /F is omitted, SMAKE obtains its input from the file MAKEFILE.
- /HELP /? Displays a brief summary of the SMAKE command line syntax.
- /I Ignores exit codes from all commands processed by SMAKE. To ignore errors for unrelated parts of the build, use the /K option; /I overrides /K if both are specified. See also .IGNORE, /K.
- /K Continues building unrelated parts of the dependencies, even if a command terminates with an error. SMAKE normally stops processing if any command returns a non-zero exit code. /K allows SMAKE to continue processing any unrelated targets (those which do not depend on the results of the current command). If an error is encountered, SMAKE returns an error code of 1, unlike the /I option which ignores all exit codes. /I overrides /K if both options are specified. Also, /C suppresses the warnings issued by /K. See also /C, /I.
- /N Displays but does not execute the commands that SMAKE would have executed. However, any preprocessing commands (*.directive*, *!directive*) are executed. Use /N to determine which targets are out of date. The /N, /D, and /P options are useful in debugging makefiles. Note that /N does not automatically recurse to subsequent calls to SMAKE. See also /D, /P.
- /NOLOGO Suppresses the SMAKE copyright message.
- /P Displays SMAKE information to the standard output device before processing begins. This includes all macro definitions, inference rules, and target descriptions, as well as the .SUFFIXES list. Specifying /P without a makefile or command line target causes SMAKE to display its information without issuing an error. The /P, /D and /N options are useful for debugging makefiles. See also /D, /N.
- /Q Checks the dates and times of targets that SMAKE would have updated, but does not update any files; only the preprocessing commands (*.directive*, *!directive*) are executed. A non-zero exit code is returned if any target is not up to date. Note that /Q does not automatically

recurse to subsequent calls to SMAKE.

- /R** Clears the .SUFFIXES list and ignores the default tools.ini settings.
- /S** Suppresses the display of all commands executed by SMAKE. See also .SILENT, @.
- /T** Changes the dates and times of the command line targets to the current date and time and executes the preprocessing commands, but does not process the normal target commands. If a command line target is not passed, SMAKE changes the date and time of the first target in the makefile. The contents of the target files are not changed, only the date and time stamp is updated.
- /X *filename*** Sends all SMAKE error output to *filename*. This can be either a file or a device. White space consisting of zero or more spaces or tabs can precede *filename*. Using a dash (-) instead of a filename causes SMAKE to send the error messages to the standard output device. SMAKE normally sends any error messages to the standard error device. This option only affects the SMAKE output, and not the output of any command executed by SMAKE.
- @*command*** A *command* file can be used to pass command line options to SMAKE. SMAKE first attempts to interpret command as an environment variable. If it does not exist, SMAKE attempts to open a linker response file with that name. The *command* file name must be preceded by the @ character, with no white space separating the @ character from the file name. Use of a *command* file also permits a command sequence longer than the 128 character limit imposed by DOS. The *command* file options can be placed on a single line separated by white space, or on multiple lines. If multiple lines are used, carriage return/line feed sequences are replaced by spaces when the lines are concatenated into a full command. Macro definitions can be continued on multiple lines by putting a backspace (\) character at the end of the lines that are to be continued.

**Note:** You can change any command line option except /F, /HELP, /?, or /NOLOGO from within a makefile with the !CMDSWITCHES directive.

## Specifying What Targets to Build

Usually SMAKE creates the first target that it finds in your makefile. However, if you specify a target on the command line, SMAKE creates that target instead.

For example, this makefile creates several utility programs:

```
all: count.exe write.exe read.exe
```

```
count.exe: count.c
           sc count.c
```

```
write.exe: write.c
           sc count.c
```

```
read.exe: read.c
          sc read.c
```

If you want to create all the targets, don't specify a target on the command line. In the example above, SMAKE encounters the target `all` first. Since the dependency list for `all` contains the executables for all the utilities, SMAKE creates all the utilities.

If you want to create just some targets, specify them on the command line. For example, this command creates the programs `write.exe` and `read.exe`:

```
smake write.exe read.exe makefile
```

If you want SMAKE to build a target only when you specify the target on the command line, make sure that target has no dependencies and is not the first target in the makefile. Such a target can delete files you no longer need or install software you just created. For example, suppose the makefile above ends with this target:

```
clean:
       del *.obj
```

If you type only `SMAKE`, SMAKE does not perform this target's command. But if you type `SMAKE clean`, SMAKE performs only this target's command and exits.

## Defining Macros

**See Also**

SMAKE lets you define macros--identifiers that are replaced with text you specify. By combining the macros in a makefile with macros in `tools.ini` (see "Customizing SMAKE Sessions with `tools.ini`"), you can use a single makefile for multiple projects. Macros are useful for specifying compiler options, paths for targets, dependents, and inference rules.

You define a macro by putting a line with the following syntax in your makefile, in `tools.ini`, or on your SMAKE command line:

```
macro = value_string
```

macro is a case sensitive combination of up to 1024 letters, digits, and underscore (`_`) characters. If a macro is to be used as a command, it cannot be null or undefined.

The `value_string` can be any sequence of characters, including no characters, or white space (interpreted as a null string). It includes all the text between the equals sign and the end of the line. Note that a macro defined as a null string is not equivalent to an undefined macro; it is still defined in relation to directives like `!IFDEF` and `!IFNDEF`.

To specify an instance of a macro that is to be expanded, enclose the macro name in parentheses and put a dollar sign in front of it, like this:

```
$(macro)
```

If a macro name is only a single character, the parentheses are not required.

**See Also**

[Customizing SMAKE Sessions with tools.ini](#)

## Examples of When to Use Macros

Macros are especially useful for defining frequently repeated text. For example, if you need to make sure that all your source files are compiled with the same compiler options, define a macro that contains those options, like this:

```
FLAGS = -c -g -DDEBUG=1 -ml
```

```
file1.obj: file1.cpp file2.h  
    sc $(FLAGS) file1.cpp
```

```
file2.obj: file2.cpp  
    sc $(FLAGS) file2.cpp
```

If you need to refer frequently to the same directory path, put the path in a macro, like this:

```
MYLIB = c:\dev\lib
```

```
prog.exe: prog.obj $(MYLIB)\mylib.lib  
    sc prog.obj $(MYLIB)\mylib.lib
```

## **Comments in Macros**

A hash character (#) on a line that defines a macro starts a comment. To use a # as part of a macro name, precede it with a caret (^). To continue a macro definition on another line, end the first line with a backslash (\). To specify a literal backslash at the end of a line, as for a directory name, precede the \ with a caret. You can use a carriage return as part of a macro name by preceding the carriage return with a ^; this also continues the macro on the next line.

## Specifying Text to Be Substituted

You can specify text to be substituted when an instance of a macro is expanded by including a colon (:) followed by the string to be substituted for, an equal sign (=), and the string to substitute. For example:

```
MODEL=L           # defines macro
$(MODEL)          # instance to be expanded
$(MODEL:L=S)      # expands 'L' to 'S'
```

The substitution applies only to the current instance, not to the original macro definition. Do not put white space characters before the colon; white space after the colon is interpreted as part of the string to be substituted. The text to be substituted is case sensitive.

## Overriding Macros on the Command Line

You can override the macros defined in a makefile or `tools.ini` by specifying a new definition on the SMAKE command line. For example, to override the definition of `MYLIB` in the example above, enter a command like this:

```
smake MYLIB=c:\dev\test\lib
```

You can also refer to macros that you define in the environment with the operating system command `SET`. For example, if you define `temp` with this command:

```
set temp = c:\dos
```

you can refer to it like this:

```
test.obj: test.cpp
    sc -cod$(temp)\test.cod test.cpp
```

If you define a macro in more than one place, SMAKE chooses definitions in this order of priority:

1. Definitions on the command line
2. Definitions in the makefile
3. Definitions in the environment (specified using the `SET` command)

**Note:** If a macro is to be passed on the command line, double quotation marks (") must surround any part of its definition that contain spaces. This is true not only for spaces that are part of the macro's value string, or even for spaces that appear on either side of the equal sign (=) that separates the macro's name from its value string.

## Recursively Defined Macros

Ordinarily, macros are defined only for the current SMAKE session or iteration. The only macros that are defined recursively in calls to SMAKE from within a makefile are:

- Macros specified on the command line
- Macros predefined by environment variables
- One of the macros MAKE, MAKEDIR, or MAKEFLAG

Macro	Description
MAKE	The name of the executable specified on the SMAKE command line. The default is SMAKE.
MAKEDIR	The current directory when SMAKE was called.
MAKEFLAG	The current SMAKE options (you can change these with the !CMDSWITCHES directive).

## Predefined Macros

SMAKE supports the following predefined macros, which you can use in your makefile:

Macro	Description
\$?	Lists dependencies that are newer than the target.
**	Full list of dependents of the current target.
\$*	Current target's name and path, without the extension.
\$<	The dependent file that is newer than the target. This macro is only valid within commands in inference rules.
\$@	The current target's fully qualified path name.
\$\$@	The current target's fully qualified path name. This macro is valid only for specifying a dependent on a dependence line.
\$\$	Expands to \$.

### Modifiers for predefined macros

You can use the following filename modifiers in combination with the predefined macros listed above:

#### Filename modifiers for use with predefined macros

Modifier	Description
B	Base file name
D	Drive and directory
F	Base name and extension
R	Drive, directory, and base name

For example, for the file `c:\project\app.exe`:

```
$(@D) refers to c:\project
$(@F) refers to app.exe
$(@B) refers to app
$(@R) refers to c:\project\app
```

### Predefined command and option macros

`tools.ini` predefines macros that correspond to commands and command options. The option macros are undefined by default. You can define these macros to expand to the commands and options you want to pass to the compiler and tools. (See `tools.ini` for a list of definitions.)

#### Predefined command and option macros

Macro	Description
AS	Command to run the Microsoft Macro Assembler

CC	Command to run the Symantec C Compiler
CPP	Command to run the Symantec C++ Compiler
CXX	Command to run the Symantec C++ Compiler
RC	Command to run the Symantec Resource Compiler
AFLAGS	Options for the Microsoft Macro Assembler
CFLAGS	Options for the Symantec C Compiler
CPPFLAGS	Options for the Symantec C++ Compiler
CXXFLAGS	Options for the Symantec C++ Compiler
RFLAGS	Options for the Symantec Resource Compiler

#### **Macros for predefined environment variables**

In addition to the predefined macros listed above, every environment variable that is defined when SMAKE starts up is equivalent to a predefined macro. This can cause unexpected results if the value of an environment variable used as a macro contains a \$ character, because SMAKE interprets \$ as the beginning of a macro invocation.

## Macro Precedence

The order of precedence for macros in SMAKE sessions is:

1. Macros defined on the command line
2. Macros defined in a makefile or include file
3. Macros that correspond to predefined environment variables
4. Macros defined in `tools.ini`
5. Macros that correspond to commands or command options, like CPP or CPPFLAGS

## Using Operating System Commands

Command lines that follow dependence lines, as well as inference rules, can contain any command that is valid on the command line. SMAKE runs these commands when a target is out of date. You can specify multiple commands by putting each command on its own line. If there are no commands following a dependence line, the dependency is checked against the inference rules.

A command line must begin with one or more spaces or tabs, and must immediately follow the dependence line. No blank lines can separate them; however, a line containing only white space can be used to specify a null command. Blank lines can appear within a list of commands. You can continue commands by ending the line to be continued with a backslash (\). If any characters, including spaces or tabs, follow the backslash, SMAKE interprets them literally. You can place single commands at the end of a dependence line by separating the dependency from the command with a semicolon (;).

To let SMAKE know how to execute a command line, precede a command with any of the following modifiers. More than one modifier can be used with a command:

### Modifiers for SMAKE commands

Modifier	Description
-	Tells SMAKE to ignore the exit status from a command; processing continues no matter what value the command returns. Otherwise, SMAKE stops executing if a command returns an error.
-N	Tells SMAKE to halt execution only if the exit status a command returns is greater than the number N (even if .IGNORE or /I are not being used). Otherwise SMAKE halts when a command returns a non-zero exit code.
@	Tells SMAKE not to display a command when it is executed, even if /S or .SILENT are not specified. Otherwise, SMAKE displays commands as they are executed.
!	Executes a command for each of the dependents in the dependency list. The command must use one of the file name macros <code>\$\$\$</code> or <code>\$?</code> , or the <code>!</code> prefix is ignored. <code>\$\$\$</code> causes the command to be executed for all of the dependent files. <code>\$?</code> causes the command to be executed only for those dependent files which are more recent than the target.
*	Allows SMAKE to accept long command lines. Causes all arguments to be assigned to the temporary variable <code>\$MAKE\$</code> , and all arguments to be replaced with <code>@\$MAKE\$</code> .
<<file	Specified after a command. If file is specified it must appear immediately after the angle brackets (<<), with no white space. If file is not specified, SMAKE uses a unique filename. Tells SMAKE to use inline files when processing a command. The file is created from the literal text beginning on the line immediately following the angle brackets. You end the inline file by beginning a line with <<. Optionally, you can write one of the modifiers KEEP or NOKEEP immediately after the closing <<. NOKEEP (the default) tells SMAKE to delete file at the end of the session. KEEP tells SMAKE not to delete file at the end of the session (though file will be overwritten each time it is used).

For example, in the makefile fragment below, SMAKE continues even if the RM command returns an error, and SMAKE executes the PROG command with `command.com` so you can use I/O redirection (with the greater than symbol `>` in this case):

```
install:
    -rm \bin \prog
```

SMAKE always executes the commands in the table below by calling the operating system. You don't need to use the `+` modifier when you specify these commands.

break	cd	chdir	cls	copy
ctty	date	del	dir	echo
erase	exit	if	md	mkdir
pause	rd	rem	rmdir	ren
rename	time	type	verify	vol

**Note:** SMAKE handles `del` commands itself, unless they are followed by a file specification containing wildcard characters. This allows you to specify a command like

```
del $(OBJS)
```

where `OBJS` is a macro that expands to a list of file names.

## Preprocessing Directives

Preprocessing directives are commands for SMAKE. SMAKE processes these directives before it processes dependency lines and commands. You can place preprocessing directives in makefiles or in `tools.ini`.

There are two types of preprocessing directives:

- Directives that begin with an exclamation point ( `!` ) work like C preprocessing directives. The `!` must be the first character the line; it can be followed by white space. `!` directives are not case sensitive.
- Directives that begin with a period ( `.` ) and end with a colon ( `:` ) cannot appear among dependency lines and commands. They must begin the line on which they appear, and are case sensitive.

## ! (Exclaim) Preprocessing Directives

**See Also**

SMAKE supports these ! preprocessing directives:

### ! preprocessing directives

Directive	Description
! CMDSWITCHE S {+/-} <i>opt</i>	Turns on or off one or more command line options with the exception of /F, /HELP, /NOLOGO, /X, or /?. In the makefile, only the /D, /I, /N, and /S options are valid, though <code>tools.ini</code> can contain the others.
!ELSE	If the preceding <b>!IF</b> , <b>!IFDEF</b> or <b>!IFNDEF</b> evaluated to zero, the statements between the <b>!ELSE</b> and the next <b>!ENDIF</b> are processed. An additional <b>IF</b> , <b>IFDEF</b> or <b>IFNDEF</b> can be combined on the same line.
!ELSEIF	Equivalent to the <b>!ELSE IF</b> directives.
!ELSEIFDEF	Equivalent to the <b>!ELSE IFDEF</b> directives.
!ELSEIFNDEF	Equivalent to the <b>!ELSE IFNDEF</b> directives.
!ENDIF	Marks the end of a block beginning with an <b>!IF</b> , <b>!IFDEF</b> , or <b>!IFNDEF</b> directive.
!ERROR <i>text</i>	Stops the SMAKE session with a fatal error, followed by the <i>text</i> . This directive will stop the session even if other options, directives or command modifiers, such as /K, /I, .IGNORE, or -, are being used.
!IF <i>expression</i>	If the <i>expression</i> evaluates to other than zero, the statements between the <b>!IF</b> and the next <b>!ELSE</b> or <b>!ENDIF</b> are processed. See "Expressions in preprocessing directives" for information.
!IFDEF <i>macro</i>	If the <i>macro</i> is defined, even with a null value, the statements between the <b>!IFDEF</b> and the next <b>!ELSE</b> or <b>!ENDIF</b> are processed.
!IFNDEF <i>macro</i>	If the <i>macro</i> is not defined, the statements between the <b>!IFNDEF</b> and the next <b>!ELSE</b> or <b>!ENDIF</b> are processed.
!INCLUDE < <i>filename</i> >	Reads and evaluates the <i>filename</i> and continues the SMAKE session. If angle brackets (<>) are used, the directories in the <b>INCLUDE</b> macro is searched to locate the <i>filename</i> . If angle brackets are not used, the current directory or the specified path is searched to locate the <i>filename</i> . For compatibility with NMAKE and Symantec MAKE, you can specify this directive without the ! prefix.
!MESSAGE <i>text</i>	Writes the <i>text</i> to the standard output and continues the SMAKE session.
!UNDEF <i>macro</i>	Removes the <i>macro</i> from the SMAKE symbol

table.

**See Also**

[Expressions in Preprocessing Directives](#)

## . (Period) Preprocessing Directives

SMAKE supports these . preprocessing directives:

### . preprocessing directives

Directives	Description
.IGNORE	Directs SMAKE to ignore exit codes from all the commands it processes from this point to the end of the file. You can also use <b>!CMDSWITCHES +/I</b> . See also /I, -, <i>-number</i> .
.LONGCOMMAN DLINE: <i>tools</i>	Permits longer command lines. <i>tools</i> is a list of one or more executable program names, with no path or extension. Multiple occurrences of .LONGCOMMANDLINE are cumulative. If <i>tools</i> is not specified, the current list is cleared. Any tool on the <i>tools</i> list whose name appears in the makefile has its command line arguments passed in a temporary environment variable. For example: .LONGCOMMANDLINE:sc link lib
.PRECIOUS : <i>targets</i>	Tells SMAKE not to delete the <i>target</i> if the command to make it is interrupted. If a command is interrupted with a CTRL+C or CTRL+BREAK, SMAKE deletes the <i>target</i> by default. Making a <i>target</i> .PRECIOUS: will make the target immune to deletion across the entire makefile, not just a portion of it.
.SILENT	Disables the display of command lines before they are executed. You can also use <b>!CMDSWITCHES /S</b> . See also /S, @.
.SUFFIXES : <i>lis</i>	Provides a list of file name suffixes for SMAKE to use when applying inference rules. The predefined list (as defined in <i>tools.ini</i> ) contains: .exe .obj .asm .c .cpp .cxx .res .rc.  Additional suffixes can be added by using <b>.SUFFIX</b> ; separate each suffix in the list with white space, one or more spaces, or tabs. To clear the suffix list, specify <b>.SUFFIX</b> without any suffixes. See also /P.

## Expressions in Preprocessing Directives

The !IF and !ELSE IF directives use the result of an expression, which is evaluated when these directives are encountered. This expression can consist of any combination of string constants, integer constants, or the names of external programs to be run.

You can group subexpressions by enclosing them in parentheses. Any constant string in an expression must be enclosed in double quotation marks (""), even if it is a macro. Quoted strings can be compared using the equality (==) or inequality (!=) operators.

Numeric values are treated as signed long integers. Numbers are assumed to be decimal values. Octal values must start with 0; hexadecimal values must start with 0x. Constant expressions can use any binary operators (see below); the integer constants can also use the unary operators.

An expression can also consist of the name of an external program, enclosed in square brackets ( []). The program will be executed during the preprocessing phase of makefile processing, and that portion of the expression will be

replaced by an integer value equal to the error level returned by the executed program.

## Operators in Makefiles

You can use the following operators in expressions:

### Operators used in SMAKE expressions

Operator	Description
DEFINED ( <i>macro</i> )	Unary operator that evaluates to TRUE if the <i>macro</i> is defined. The expression <b>!IF DEFINED</b> ( <i>macro</i> ) is equivalent to the expression <b>!IFDEF</b> <i>macro</i> .
EXIST( <i>path</i> )	Unary operator that evaluates to TRUE if the <i>path</i> exists. Some operating systems allow spaces within the path name. If a space is used, the <i>path</i> must be surrounded by double quotation marks (").
+	Addition
-	Subtraction
*	Multiplication
/	Division
==	Equality
!=	Inequality
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
&&	Logical AND
	Logical OR
<<	Left shift
>>	Right shift
%	Modulus

## Running Programs with Preprocessing Directives

When you run a program with a SMAKE preprocessing directive, you typically obtain and test its exit code and execute other commands based on the result. For example:

```
!if [c:\mydir\myprog] == 0
# continue processing
!else
# do something else
!endif
```

If you use macros as part of the command line, SMAKE expands them before running the program.

## Inference Rules Overview

You can define rules that tell SMAKE how to automatically create certain types of files. For example, you might define rules that tell SMAKE how to create an object file (`.obj`) from a C++ source file (`.cpp`) and how to create an executable file (`.exe`) from an object file (`.obj`). By defining inference rules, you reduce the number of actions you must explicitly specify.

SMAKE interprets inference rules as templates for creating a target from dependent files, based on the extensions of the files involved. You can use predefined inference rules or write your own; they can be specified in the makefile or in `tools.ini`.

To determine priorities for applying inference rules, SMAKE uses the list associated with the **.SUFFIXES** directive.

## **When SMAKE Applies Inference Rules**

SMAKE applies inference rules when:

- A dependence line does not contain any dependents or is not followed by any commands
- A target is specified that does not exist in the makefile (or the makefile does not exist).
- A dependent file does not exist and is not itself a target.

## Inference Rule Syntax

The dependence line for an inference rule contains the source file extension, the destination file extension, and a colon. Inference rules have the following syntax:

```
{dependent_path}.dependent_ext{target_path}target_ext:
    commands
```

where `dependent_path` is the path for the dependent file and `dependent_ext` its extension, and `target_path` is the path for the target and `target_ext` its extension. If no paths are specified, SMAKE looks for the files in the current directory. `commands` are the steps SMAKE will take if the dependent file is out of date.

You can use macros for paths and extensions. Use the macros in Table 13-4 to specify the arguments. Do not use white space in an inference rule.

For example, this rule builds an object file from a C++ file:

```
.cpp.obj :
    sc -c $*          # $* contains the name of
                      # the target file without
                      # an extension
```

A simple makefile can use inference rules to create a program from two source files, such as:

```
.cpp.obj
    sc -c $*

.obj.exe
    sc $**

test.exe: test.obj util.obj

test.obj: test.cpp util.h

util.obj: util.cpp
```

### Levels of inference rules

SMAKE can interpret no more than one level of inference rules. For example, this makefile does not produce an executable:

```
.c.obj: ;sc -c $
.obj.exe: ;sc $
hello.exe: hello.c          # ERROR
```

## How Inference Rules Work

Inference rules work on targets and dependents with the same file name and different extensions; they do not match multiple files. For example, you can specify an inference rule to build `myprog.obj` from `myprog.cpp`, but not `yourprog.obj` from `myprog.cpp`.

For example, given this inference rule:

```
.cpp.obj:
    $(CPP) $(CPPFLAGS) $<
```

SMAKE will apply this rule to any pair of target/dependency files in the current directory (or on the specified path) that have the same name, where one has the extension `.cpp` and the other `.obj`. SMAKE will expand the predefined macros and run the resulting commands on those dependents that are newer than their corresponding targets (as specified by `$<`).

If the target specified has a `.exe` extension, SMAKE searches for a file with the same base name and an extension that is in the **.SUFFIXES** list to find which inference rule to use.

If the dependence line is followed by commands but does not list dependents, SMAKE will use the inference rules and the **.SUFFIXES** list to determine the dependent file and then apply the commands specified in the makefile for the target.

### Order of precedence for inference rules

SMAKE determines the precedence of inference rules as follows:

- | Order | Inference Rule  |
|-------|---|
| 1     | Defined in a makefile; the latest defined inference rule applies. |
| 2     | Defined in the <code>tools.ini</code> file.                       |
| 3     | Predefined inference rule.  |

## Predefined Inference Rules

The following inference rules are predefined in `tools.ini`:

### Predefined inference rules

Inference rule	Command
.asm.exe	\$(AS) \$(AFLAGS) \$*.asm
.asm.obj	\$(AS) \$(AFLAGS) /c \$*.asm
.c.exe	\$(CC) \$(CFLAGS) \$*.c
.c.obj	\$(CC) \$(CFLAGS) /c \$*.c
.cpp.exe	\$(CPP) \$(CPPFLAGS) \$*.cpp
.cpp.obj	\$(CPP) \$(CPPFLAGS) /c \$*.cpp
.cxx.exe	\$(CXX) \$(CXXFLAGS) \$*.cxx
.cxx.obj	\$(CXX) \$(CXXFLAGS) /c \$*.cxx
.rc.res	\$(RC) \$(RCFLAGS) /r \$*

## Customizing SMAKE Sessions with tools.ini

You can customize SMAKE sessions by placing default values for command line options in the initialization file `tools.ini`. Any settings you specify in `tools.ini` will be used for every SMAKE session, unless `/R` is specified on the command line. The following is a sample of SMAKE information in `tools.ini`:

```
[SMAKE]
#           macro redefinitions
            CCFLAGS = -A -wx -mn
            DEBUGFLAGS = -gh
#           inference rule
```

```
.c.boj:
    $(CC) $(CCFLAGS) $<
```

SMAKE information must follow one of the tags `[NMAKE]` or `[SMAKE]`. Case is not significant. The format for comments is the same as in makefiles. `tools.ini` is read before the makefile; you can override macro definitions and inference rules specified in `tools.ini` in the makefile.

When it searches for `tools.ini`, SMAKE searches the directory for executable files first, then the current directory, and finally in the directory specified by the `INIT` environment variable.

## Creating Response Files

SMAKE can automatically create a response file on disk, which contains whatever text you specify in the makefile. You can specify the response file by name as input to another program, link OPTLINK or LIB.

To write a line to a response file, use the command ECHO and redirect its output with one and two greater-than signs (>>). Write the first line with one greater-than sign so that ECHO creates a new file. Write the rest of the lines with two greater-than signs so that ECHO does not overwrite the file you just created.

For example, this makefile creates the program `main.exe` from the source files `main.cpp` and `funcs.cpp`. It not only creates a linker response file, but also writes the linker's output to `cmp.err`:

```
main.exe: main.obj funcs.obj
    echo main.obj+funcs.obj > linker.rsp
    echo main.exe,,, >> linker.rsp
    +link @linker.rsp > lnk.err

main.obj: main.cpp
    +sc -c main.cpp > cmp.err
funcs.obj: funcs.cpp
    +sc -c funcs.cpp >> cmp.err
```

## Changing Dates with TOUCH

Sometimes you want SMAKE to think that some files are newer than their modification date suggests. For example, you might want to recompile all your C++ source files, regardless of their dates. Or, you might want to use old object files without recompiling your source files.

The TOUCH utility changes the modification date of files to the current date. TOUCH commands have the form:

```
TOUCH file1 file2 . . .
```

Use TOUCH with SMAKE to maintain the dates of files and to modify the operation of SMAKE. For example, if you want all the C++ source files that comprise a program to be recompiled (not just the ones last modified), use the following commands:

```
TOUCH *.cpp  
SMAKE
```

Also, if you edit one or more files, but you change only comments or other items that do not affect the resultant object file, then TOUCH does not remake the object files.

```
TOUCH *.obj  
SMAKE
```

REMOVED

## Choosing Dependency Files

Since you use dependency files to create the target file, the names of the dependency files usually appear in the command lines that follow the dependence line. But if an object file depends on a source file and that source file includes a header file, the object file depends on that header file. If that header file changes, you must recompile the source file to create a new object file. It can be useful to include these kinds of files in dependency lists.

Suppose that `hello.cpp` includes the file `hello.h`. The dependence line for `hello.obj` looks like this:

```
hello.obj : hello.cpp hello.h
```

If a header file isn't likely to change, omit it from the dependency list. For example, system header files are unlikely to change. System header files include header files for standard libraries, such as `stdio.h`, and header files for operating system functions, such as `windows.h`.

## Producing Dependency Lists with MAKEDEP

The MAKEDEP utility produces dependency lists that you can use in your makefiles. MAKEDEP commands have the following form:

```
MAKEDEP [-d] {-ipath} [-m] [-s] [-t]  
{-xexclude} filenames . . .
```

where:

Argument	Description
-d	Prints debugging output
-ipath	Specifies a search path for header files
-m	Include files that are referenced in #include statements, but do not actually exist. By default, MAKEDEP only includes files that exist.
-s	Doesn't list system include files as dependencies (that is, header files enclosed in less-than and greater-than signs, such as #include <stdio.h>)
-t	Do not include path names in generated targets
-xexclude	Specifies a directory to exclude from the search path. Any path of which this directory is a part will be excluded.
filenames . . .	A list of filenames, separated by spaces, for which dependency lists are to be created

You can redirect the output of MAKEDEP to a file that you can use as the starting point for a makefile, like this:

```
makedep -s menu.cpp wbase.cpp > makefile
```

The makefile looks like this:

```
menu.obj : wbase.h menu.h menu.cpp  
wbase.obj : wbase.h wbase.cpp
```

## **Including Files**

You can include files in makefiles by adding an include statement to a makefile command line, like this:

```
!include filespec
```

The `!include` statement lets one makefile support several configurations of a program. You can use `SET` commands to set environment variables for executed programs in the included file.

- Definition file directives for Win32 and Win32s applications, see the section "Win32 Programming Guidelines."
- Definition file directives for Win16 applications, see the section "Win16 Programming Guidelines."
- Definition file directives for DOS applications, see the section "DOS Programming Guidelines."

Win32 Programming Guidelines

Win32 Programming Guidelines

DOS Programming Guidelines

## Definition File

OPTLINK uses a module definition file (`.def` file) to provide additional input for its operation when generating a segmented `.exe` or `.dll` output file. The `.def` file usually is omitted to generate an MS-DOS `.exe` file. The `.def` file describes the essential characteristics of an application or library for Win16 or Win32. It also assigns segment sequences, function orders for encapsulated functions, and so on, even in conventional MS-DOS `.exe` files.

## Directive Overview

[Click a directive for more information.](#)

OPTLINK recognizes the directives summarized below. These directives and their effects are described in the paragraphs which follow.

### OPTLINK directives

Directive	Action
<u>APPLoader</u>	Used with self-loading FORTRAN applications (16-bit only)
<u>CODE</u>	Sets default attributes for code segments
<u>CODECLASSES</u>	Specifies additional class names to be treated as code segments
<u>DATA</u>	Sets default attributes for data segments
<u>DESCRIPTION</u>	Specifies one-line description of module
<u>EXETYPE</u>	Identifies target operating system
<u>EXPORTS</u>	Establishes exported functions
<u>FIXDS</u>	Eliminates the need for <u>EXPORTS</u> (Win16 only)
<u>FUNCTIONS</u>	Orders COMDAT records
<u>HEAPSIZE</u>	Establishes local-heap size in bytes
<u>IMPORTS</u>	Establishes imported functions
<u>INCLUDE</u>	Specifies the point at which another file is inserted into <code>.def</code> file
<u>LIBRARY</u>	Names dynamic link library; does not create an application
<u>LONGNAMES</u>	Indicates that module is HPFS-aware (16-bit only)
<u>NAME</u>	Assigns name to an application; does not create library
<u>NEWFILES</u>	Indicates that module is HPFS-aware (16-bit only)
<u>OBJECTS</u>	Same as <u>SEGMENTS</u>
<u>OLD</u>	Preserves export information from previous library version (16-bit only)
<u>PROTMODE</u>	Prevents module from running in real mode (16-bit only)
<u>RC</u>	During linking, copies <code>.res</code> file into <code>.exe</code> during linking
<u>REALMODE</u>	Relaxes <u>PROTMODE</u> restrictions (16-bit only)
<u>SECTIONS</u>	Same as <u>SEGMENTS</u>
<u>SEGMENTS</u>	Sets attributes for specific segments
<u>STACKSIZE</u>	Establishes local stack size in bytes
<u>STUB</u>	Adds DOS stub to generated file, usually to terminate program

## Using Definition File Directives

Only one of the directives NAME or LIBRARY can appear in a single definition file. If neither directive appears, NAME is the default, and the module is treated as an application.

You may enter comments in the definition file by prefixing them with a " ; ." The comment extends to the end of the line on which it appears.

You may use either the single- or the double-quote character, especially if the symbol matches any reserved word, regardless of case.

Except for the restriction cited above, format of the file is essentially free. Except for their actions in terminating a comment, new line characters are treated simply as white space, having no syntactic significance.

## **APPLOADER (16-bit only)**

This optional directive tells OPTLINK the name of an external symbol that is defined in a loader module and, by prefixing it with "LOADER\_", is treated as the identifying portion of the loader code segment name.

SYNTAX:     APPLOADER 'symbol'

EXAMPLE:    APPLOADER 'START'

COMMENTS: The example specifies that `LOADER_START` is the name of the code segment in the loader module at which execution is to begin. This keyword supports self-loading FORTRAN applications.

## CODE

This directive establishes the default attributes for all code segments within the application or library being linked. You must use an attribute from the following list, and only one attribute for each field may appear. Order of appearance is not significant. Fields marked with "\*" have no effect on protected-mode code segments; they apply only to real-mode modules.

### Code attributes

Field	Keywords
load	LOADONCALL (default), PRELOAD
executeonly	EXECUTEREAD (default), EXECUTEONLY
iopl	NOIOPL (default), IOPL
conforming	NONCONFORMING (default), CONFORMING
shared*	NONSHARED (default), SHARED
movable*	FIXED (default), MOVABLE
discard*	NONDISCARDABLE (default), DISCARDABLE

Meanings of the keywords for each of these fields follow. The `load` field determines when a code segment is loaded:

### Field keywords

Keyword	Meaning
PRELOAD	Automatically loads when the program begins.
LOADONCALL	Is not be loaded until it is accessed. If no attribute is specified, this is the default condition.

The `executeonly` field determines whether a code segment can be read as well as executed:

### Execution field keywords

Keyword	Meaning
EXECUTEONLY	Executes only; valid only in protected mode. Equivalent to <code>EXECUTE</code> .
EXECUTEREAD	Reads and executes. If no attribute is specified, this is the default condition. Equivalent to <code>READ</code> .

The `iopl` field determines whether a code segment has I/O privilege:

### iopl field keywords

Keyword	Meaning
IOPL	Has I/O privilege and can access hardware directly; enforced only in protected mode.
NOIOPL	Does not have I/O privilege; hardware access is made only through the operating system. If no attribute is specified for the field, this is the default condition.

The `conforming` field specifies the privilege level for the process; it is meaningful only for device drivers and system-level procedures. A conforming segment is called from either Ring 2 or Ring 3, and executes at the caller's privilege level. Keywords for this field are:

### Conforming field keywords

Keyword	Meaning
CONFORMING	Is conforming. This attribute is used only in protected mode.
NONCONFORMING	Is not conforming. If no attribute is specified for the field, this is the default condition.

The `shared` field, ignored by protected mode but used by real-mode modules, determines whether a code segment can be shared by all instances of the program. Keywords for this field are:

#### Shared field keywords

Keyword	Meaning
SHARED	Can be shared.
NONSHARED	Cannot be shared. If no attribute is specified, this is the default condition.

The `movable` field, ignored by protected mode but used by real-mode modules, determines whether a code segment can be moved around in memory. Keywords for this field are:

#### Movable field keywords

Keyword	Meaning
MOVABLE	May be moved during operation.
FIXED	May not be moved during operation. If no attribute is specified, this is the default condition.

The `discard` field, ignored by protected mode but used by real-mode modules, determines whether a code segment may be discarded from memory when not currently in use. Keywords for this field are:

#### Discard field keywords

Keyword	Meaning
DISCARDABLE	Can be discarded.
NONDISCARDABLE	Cannot be discarded. If no attribute is specified, this is the default condition.

SYNTAX:     CODE [attribute . . .]

EXAMPLE:    CODE PRELOAD EXECUTEONLY

COMMENTS: The example sets default attributes of `PRELOAD` and `EXECUTEONLY` for all code segments. These defaults may be overridden by means of the `SEGMENTS` directives. Note that some attributes imply other attributes: A `DISCARDABLE` segment is also `MOVABLE`, whether so specified or not. Similarly, a `FIXED` segment is `NONDISCARDABLE`, whether or not so specified. In case of conflicts, the last specified attribute (together with all of its implications) will be used. Thus a directive such as "`CODE FIXED DISCARDABLE`" results in attributes of "`MOVABLE DISCARDABLE`". If this is not your intention, one or the other of the attributes must be removed.

The `CODE` directive should precede any `SEGMENTS` directives in the definition file that refer to code segments; it will have no effect on any code segment declarations which precede it in the file.

## CODECLASSES

By default OPTLINK assumes that segments with a class name ending in `CODE` are code segments. This directive may be used to specify additional class names to be treated as "CODE".

SYNTAX:     CODECLASSES [classname ...]

EXAMPLE:    CODECLASSES TEXT

COMMENTS: The example specifies that class name "TEXT" identifies a code segment, in addition to the defaults.

## DATA

This directive establishes the default attributes for all data segments within the application or library being linked. Each attribute must be taken from the following list, and only one attribute for each field may appear. Order of appearance is not significant. Fields marked with "\*" have no effect on protected mode code segments; they apply only to real-mode modules.

### Data keywords

Field	Keywords
load	LOADONCALL (default), PRELOAD
readonly	READWRITE (default; equivalent to READ WRITE), READONLY
instance	MULTIPLE (default for application), SINGLE (default for DLL), NONE
iopl	NOIOPL (default), IOPL
shared	NONSHARED (default), SHARED
movable *	FIXED (default), MOVABLE
discard *	NONDISCARDABLE (default), DISCARDABLE

Meanings of the keywords for each of these fields follow. The load field determines when a data segment will be loaded:

### Data keyword explanations

Keyword	Meaning
PRELOAD	The segment is automatically loaded when the program begins. In real-mode, data should be PRELOADED.
LOADONCALL	The segment will not be loaded until it is accessed. If no attribute is specified for the field, this will be the default condition.

The `readonly` field determines access rights to a data segment. Keywords for this field are:

### readonly field keywords

Keyword	Meaning
READONLY	The segment can only be read from. This attribute is valid only in protected mode.
READWRITE	The segment can be both read from, and written into. If no attribute is specified for the field, this will be the default condition.

The `instance` field determines creation of automatic data segments, and the number that will be created. Keywords for this field are:

### Instance field keywords

Keyword	Meaning
NONE	No automatic data segment will be created.
SINGLE	A single automatic data segment will be created. If no attribute is specified for the field, this will be the default condition for dynamic link library modules. The shared attribute is forced to <code>SHARED</code> when this

keyword is used.

**MULTIPLE** Separate automatic data segments will be created for each instance of the module. If no attribute is specified for the field, this will be the default condition for application modules. The shared attribute is forced to `NONSHARED` when this keyword is used.

The `iopl` field determines whether a data segment can be accessed by code that does not have I/O privilege. Keywords for this field are:

#### **iopl field keywords**

<b>Keyword</b>	<b>Meaning</b>
<code>IOPL</code>	Access to the data segment is permitted only from code segments which have <code>IOPL</code> set.
<code>NOIOPL</code>	Access to the data segment is permitted from any code segment.

The `shared` field, ignored by protected mode if the segment has the `READONLY` attribute but used by real-mode modules, determines whether a data segment can be shared by all instances of the program. Keywords for this field are:

#### **Shared field keywords**

<b>Keyword</b>	<b>Meaning</b>
<code>SHARED</code>	One copy of the data segment will be loaded, and shared by all instances of the program. The instance attribute is forced to <code>SINGLE</code> when this keyword is used.
<code>NONSHARED</code>	The segment cannot be shared, and a separate copy will be loaded for each instance of the program. If no attribute is specified for the field, this will be the default condition. The instance attribute is forced to <code>MULTIPLE</code> when this keyword is used.

The `movable` field, ignored by protected mode but used by real-mode modules, determines whether a data segment can be moved around in memory. Keywords for this field are:

#### **Movable field keywords**

<b>Keyword</b>	<b>Meaning</b>
<code>MOVABLE</code>	The segment may be moved during operation.
<code>FIXED</code>	The segment may not be moved during operation. If no attribute is specified for the field, this will be the default condition.

The `discard` field, ignored by protected mode but used by real-mode modules, determines whether a data segment may be discarded from memory when not currently in use. Keywords for this field are:

#### **Discard field keywords**

<b>Keyword</b>	<b>Meaning</b>
<code>DISCARDABLE</code>	The segment can be discarded as needed.
<code>NONDISCARDABLE</code>	The segment cannot be discarded. If no attribute is specified for the field, this will

SYNTAX:           be the default condition.  
DATA [*attribute* ...]

EXAMPLE: DATA PRELOAD

COMMENTS: The example sets the default attribute for all data segments to PRELOAD. As with the CODE directive attributes, it is possible to declare contradictions and in such cases the last declared attribute takes effect, canceling earlier ones. The fields which interact with each other are the instance and shared fields.

The DATA directive should precede any SEGMENTS directives in the definition file that refer to data segments; it will have no effect on any data segment declarations which precede it in the file.

## DESCRIPTION

This directive inserts its associated text into the application or library being linked.

SYNTAX:     DESCRIPTION '*text*'

EXAMPLE:   DESCRIPTION 'Copyright 1993 by Symantec Corp.'

COMMENTS: The example causes the copyright notice to be embedded into the application or library being linked. Maximum length of the text is 254 characters. This directive differs from a comment line in that comments are not added to the output module.

## EXETYPE

This optional directive tells OPTLINK the operating system under which the executable file is to run. The `EXETYPE` keyword must be followed by one of five operating system keywords.

**SYNTAX:**     `EXETYPE [OS2 | WINDOWS [n.nn] | DOS4 | UNKNOWN | DOS | NT]`

**EXAMPLE:**    `EXETYPE WINDOWS 3.00`

**COMMENTS:** The example specifies that the current executable file is to run under Windows version 3.00. If no `EXETYPE` directive appears in the definition file or if Windows appears without a specified minimum version, Windows version 3.10 is assumed.

This directive sets bits in the `.exe` header which are checked by the operating system loader to verify that the program being loaded was intended to be run under that system.

The DOS keyword selects "normal" MS-DOS EXE file format, and is required if you use a definition file to create such an executable. One reason for doing so would be to take advantage of the COMDAT-record capabilities to assign functions to specific segments via the `FUNCTIONS` directive. Another is to specify segment order with `SEGMENTS`. The only definition-file directives that make sense for use with "`EXETYPE DOS`" are `INCLUDE`, `SEGMENTS`, `FUNCTIONS`, `STACKSIZE`, and `EXETYPE` itself.

The `DOS4` and `UNKNOWN` keywords are recognized but are not supported.

## EXPORTS

By default, functions within one executable file (including Dynamic Link Libraries, DLL's, which are bound at run time) are hidden at run time from all other executable files. This directive makes specified functions available to other executable files at run time. It is also used to define names and attributes for functions that run with I/O privilege. Note that the `FIXDS` directive may be used to reduce the need for `EXPORTS`.

Each export definition begins with the *extname*, which is the name of the function being defined, as it is known outside its own executable file. This may be followed by the "=" symbol and the *intname*, the name as it appears inside the executable file. If no *intname* is given, it is assumed to be the same as the *extname*.

If the function may be referenced by name or by ordinal position within the executable file's entry table, the "@" symbol followed by *numb*, the ordinal position number, should appear next. If the name is to remain resident at all times even though *numb* is given, the keyword `RESIDENTNAME` should follow. If the symbol can only be referenced by ordinal number, the keyword `NONAME` should follow. Use of ordinals and `NONAME` provide the smallest executable files and fastest access.

The remaining two fields apply to functions with I/O privilege. For protected mode, *parms* specifies the total number of words occupied by the function's parameters. For real-mode Windows, the keyword `NODATA` may appear.

**SYNTAX:**        `EXPORTS`  
                  `extname[=intname] [@numb[ RESIDENTNAME | NONAME]] [ parms] [ NODATA]`  
                  ...

**EXAMPLE:**       `EXPORTS`  
                  `CharInput 6`  
                  `GetString = StringIn`  
                  To export a symbol starting with an "@", enclose the  
                  symbol in quotes.

**COMMENTS:** The example defines two functions within the current executable file as being available to other executable files: `CharInput`, defined elsewhere as having I/O privilege, uses 6 words of parameter information; `GetString` is defined within the executable file as `StringIn`. See also the discussion of the `FIXDS` directive for an alternative method of exporting functions under Windows, but not for OS/2.

## **FIXDS (Win16 only)**

This directive, which is useful only for Windows applications, eliminates the need for the `EXPORTS` directive. It cannot be used with dynamic link libraries, or for OS/2 executable files.

SYNTAX:     `FIXDS`

EXAMPLE:    `FIXDS`

COMMENTS: When this directive is used, the entire `EXPORTS` section becomes superfluous. So does every call to `MakeProcInstance()` or `FreeProcInstance()` in the executable file. Where the return value from `MakeProcInstance()` was used (such as a `DialogBox()` call), just use the function name instead.

`FIXDS` also gives one capability that isn't ordinarily available: your Windows application can export functions to be called from a dynamic link library. You can do this by listing, in the `EXPORTS` section, the functions to be made available to the `.dll`. (This is the only case requiring an `EXPORTS` section, and only the functions to be made available to `.dll`'s should be listed.) Then run `OPTIMP` on this `.def` file to produce a `.lib` that can be linked with the `.dll`'s.

## FUNCTIONS

This directive is used to order COMDAT records and optionally assign them to specific segments.

SYNTAX:     FUNCTIONS [ :segmentname]  
              functionnames

EXAMPLE:    FUNCTIONS  
              Function1  
              Function5  
              Function2

COMMENTS: The example resequences three functions to appear in the order in which they are named.  
All functions named in this directive must be contained in COMDAT records, which may not be available from all language processors.

## HEAPSIZE

This directive sets the size of the executable file's local heap, in bytes, and thus affects the size of the automatic data segment, if any.

**SYNTAX:**     HEAPSIZE *n* [, *commit*] or HEAPSIZE MAXVAL

**EXAMPLE:**    HEAPSIZE 2000

**COMMENTS:** The example establishes size of the local heap at 2000 bytes. *n* is the total heap space to reserve; it must be an integer. *commit* is the amount of memory to allocate each time the program runs.

For compatibility with Microsoft LINK, the default *commit* size is 4KB. The minimum value for HEAPSIZE is 64KB.

By default, *n* and *commit* are to assumed be decimal, but the C language conventions for hexadecimal and octal values can be used if desired.

For 16-bit output only, if the keyword MAXVAL is used instead of a number, the size of the heap will be set at 65,520 bytes (64KB minus 16) minus STACKSIZE, minus the area within DGROUP that is statically allocated by the program. This provides the largest possible heap with no need for calculations. Windows requires that DGROUP's size be smaller than 65,520 bytes.

## IMPORTS



This directive specifies all functions referenced in the current executable file (including Dynamic Link Libraries, DLL's, which are bound at run time) that actually appear in other executable files, by name and executable file. The `IMPORTS` directive appears on a line by itself, followed by any number of import definitions. For 16-bit output, the only limit on the number of definitions is that total storage required for the names cannot exceed 64KB.

Each import definition consists of an optional internal name, and a required executable file name and function identifier pair. The function identifier, *func* in the syntax description which follows, may be either the name listed in the `EXPORTS` definition of the exporting executable file, or an ordinal value. If it is an ordinal value, then the internal name is required. If the internal name is omitted and *func* is a name, it is used as the internal name.

SYNTAX:     `IMPORTS`  
              `[internal=]externalfile.func`  
              `...`

EXAMPLE:    `IMPORTS`  
              `read.CharInput`  
              `NewString = read.GetString`

COMMENTS: The example defines two functions to be imported from an executable file or library named "read" into the current executable file. The first of these was listed by the `EXPORTS` directive of "read" as `CharInput`, and the other as `GetString`. Within this executable file, `CharInput` is known by that name, but `GetString` is referred to as `NewString`. See also the example for the `EXPORTS` directive.

A much simpler way to accomplish the same goal is to use the `IMPLIB` utility to create import libraries for each `.dll` and `.exe` that you will be importing functions from, and then link those import libraries into the using file.

**See Also**

EXPORTS

## INCLUDE

This optional directive tells OPTLINK the name of a file to be included in the `.def` file at this point.

SYNTAX:     `INCLUDE filename`

EXAMPLE:    `INCLUDE generic.inc`

COMMENTS: The example specifies that file `generic.inc` is to replace the `INCLUDE` line. Processing continues with the first line of `generic.inc`. Include files may be nested up to 10 levels deep. At the end of each file, processing continues with the next line of the file at the next higher level.

## LIBRARY

This directive indicates that the executable file produced is a dynamic link library, and optionally assigns it a name and specifies the initialization required. If no name is given, the name of the executable file (less extension) is assumed. Initialization is specified by one of the listed keywords; if none is given,

INITGLOBAL is assumed. "

### Library keywords

Keyword	Meaning
BASE	Equivalent to the /BASE switch; see "OPTLINK Switches" for information.
INITGLOBAL	The library's initialization routine is called only when the module is initially loaded into memory.
INITINSTANCE	The library's initialization routine is called each time a new process gains access to the library.
PRIVATELIB	The library is a private DLL; that is, it can be referenced by only one application at a time.
PROCESSINIT	Calls the library's entry point when a process attaches.
PROCESSTERM	Calls the library's entry point when a process terminates.
THREADINIT	Calls the library's entry point when a thread is created.
THREADTERM	Calls the library's entry point when a thread terminates.
WINDOWAPI	This is a Windows (or PM) application and can execute only in the Windows or PM environment. For Windows, this eliminates the need to use RC if there are no resources.

SYNTAX:     LIBRARY [*libname*] [*init*] [*privatelib*]

EXAMPLE:    LIBRARY mylib INITGLOBAL

COMMENTS: The example defines a dynamic link library named `mylib` and specified that it is to be initialized only when first loaded into memory.

## **LONGNAMES (16-bit only)**

This directive may be used to specify a program that is HPFS-aware. It sets a bit in the file header which makes it possible for the program to access the high performance file system.

SYNTAX:      `LONGNAMES`

## NAME

This directive indicates that the executable file produced is an application, and optionally assigns it a name and defines its type. If no name is specified, the name of the executable file (less its extension) becomes the name of the application. Three keywords are defined to signify type information, as follows:

### Name keywords

Keyword	Meaning
BASE	Equivalent to the /BASE switch; see "OPTLINK Switches" for information.
WINDOWAPI	This is a Windows (or PM) application and can execute only in the Windows or PM environment. For Windows, this eliminates the need to use RC if there are no resources. This is the default if no keyword is supplied.
WINDOWCOMPAT	This application can run inside PM, or it can run in a separate screen group. Any application which uses the proper subset of OS/2 mouse, keyboard, and video functions that are supported in PM applications can be of this type.
NOTWINDOWCOMPAT	This application must operate in a separate screen group from PM as it is not PM-compatible.
SYNTAX:	NAME [appname] [apptype]
EXAMPLE:	NAME myapp
COMMENTS:	The example defines an application named <code>myapp</code> and specifies (via the <code>WINDOWAPI</code> default) that it can be run only under Windows or PM. The <code>apptype</code> information is kept in the executable-file header.

## **NEWFILES (16-bit only)**

This directive may be used to specify a program that is HPFS-aware. It sets a bit in the file header which makes it possible for the program to access the high performance file system, in which file names need not follow the "normal" MS-DOS filename.ext (8.3) convention.

SYNTAX:      `NEWFILES`

## **OBJECTS**

This directive is identical to the SEGMENTS directive. Refer to the SEGMENTS directive discussion for details.

## OLD (16-bit only)

This directive tells OPTLINK to search another executable file (normally a DLL) for `EXPORT` ordinals. OPTLINK searches the directories listed in the path environment variable to locate the executable file, if it is not in the current working directory. If the file is not found, a warning is issued. Exported names in the current executable file which match those in the old executable file are assigned ordinal values from the old executable file.

SYNTAX:     `OLD 'filename'`

EXAMPLE:    `OLD 'oldlib.exe'`

COMMENTS: The primary use of this directive is to preserve the original ordinal numbers, through successive revisions of a library executable file. Refer to the `EXPORTS` and `IMPORTS` directives for more information about ordinal numbers for function identification.

## **PROTMODE (16-bit only)**

This directive specifies that the executable file be run only in protected mode, and not in real or dual mode. If neither this directive nor the `REALMODE` directive is present when `EXETYPE` is `WINDOWS`, `OPTLINK` assumes that the application can run only in protected mode.

SYNTAX:     `PROTMODE`

COMMENTS: `PROTMODE` should always be used when generating protected-mode-only `.exe` files; less relocation information is required, thus yielding faster linking, a smaller output `.exe` file, and faster loading at run time. `OPTLINK` cannot assume `PROTMODE` when you specify "`EXETYPE OS2`", since you may "`BIND`" this as a family (real-mode) application, thus requiring the extra relocation information.

## RC

This optional directive tells OPTLINK to copy binary resources from a `.res` file into the binary output file generated, and to set the appropriate flags in the header to indicate that the file is valid for Windows. By default, it also reorders segments and resources in such a manner that any preloaded, or fixed, or non-discardable segments and resources appear first, in a "fast-load" section, followed by the items that are to be "demand-loaded".

SYNTAX:     RC [ -e -k -l -m -p -t -30 -31 ] filename.res

EXAMPLE:    RC myfile.res

COMMENTS: The example specifies that the binary resources in `myfile.res` are to be copied to the output `.exe` or `.dll` file, eliminating the need to run the resource compiler after linking. Segments and resources are reordered as necessary to assure that preloaded items appear first.

The eight option switches accepted by this directive have the following meanings:

- -e Create a driver which uses EMS memory.
- -k       Keep segments in `.def` file order; do not sort or create a fast-load section.
- -l       LIM 3.2 EMS application is to be created.
- -m       Set Multiple Instance flag.
- -p Create a private library (`PRIVATELIB`).
- -t       Create a protected-mode-only application (`PROTMODE`).
- -30      Tag the executable as requiring Windows 3.0 or later to run.
- -31      Tag the executable as requiring Windows 3.1 or later to run.

If no filename is supplied, `xxx.res` (where `xxx` is the output filename) is assumed. The current default directory is searched first for the file. If none is found there, the path(s) listed in the `OBJ` environment variable are used. If none is found there, the path(s) listed in the `LIB` environment variable are used. If the file is not found, a valid no-resource output file is created.

## **REALMODE (16-bit only)**

This directive specifies that the executable file can be run in real or protected mode. If neither this directive nor the `PROTMODE` directive is present, OPTLINK assumes that the application can run only in protected mode for "EXETYPE WINDOWS". The default is `REALMODE` for "EXETYPE OS2".

SYNTAX:     `REALMODE`

## **SECTIONS**

This directive is identical to the SEGMENTS directive. Refer to the SEGMENTS directive discussion for details.

## SEGMENTS

This directive establishes the attributes of one or more segments on an individual segment by segment basis, and overrides the defaults specified by the `CODE` and `DATA` directives. The `SEGMENTS` directive appears on a line by itself, followed by one or more definition lines. These definition lines, by their sequence, establish the order in which segments appear in the output file.

Each definition line begins with a segment name, which may be enclosed in single quotes (the quotes are required, if the name conflicts with any `OPTLINK` keyword). The keyword `CLASS` may then appear, followed by a class name in required quotes. If no `CLASS` is specified, `CODE` is assumed. The rest of the line contains attributes for the fields applicable to the segment's class, from the lists presented in describing the `CODE` and `DATA` directives.

SYNTAX:     `SEGMENTS`

```
[']segname'[ ] [CLASS'classname'] [attribute [attribute ...]]
[']segname'[ ] [CLASS'classname'] [attribute [attribute ...]]
...
```

EXAMPLE:   `SEGMENTS`

```
seg1 EXECUTEREAD CONFORMING
seg2 CLASS 'special' IOPL
seg3 CLASS 'data' READONLY
```

COMMENTS: The example defines three segments, named `seg1`, `seg2`, and `seg3`. The class of `seg1` is `CODE` by default, that of `seg2` is "special", and that of `seg3` is `data`.

Note that the `SEGMENTS` directive can implicitly change the type of a segment: calling a code segment with an attribute of `READWRITE` causes that segment to be treated as a data segment. The opposite is also true: calling a data segment with an attribute of `EXECUTEREAD` causes that segment to be treated as code.

It is also essential to preserve case significance in all references to segment or class names; wrong case or other typographical errors in segment or class names are not flagged as errors. `OPTLINK` ignores erroneous segments and no relevant information is generated within the `.map` file.

Private segments should be avoided since the `SEGMENTS` directive affects only the first occurrence of a private segment with a given name and class.

## STACKSIZE

This directive performs the same functions as the command-line option switch `/STACKSIZE`, but overrides the effects of the option switch.

SYNTAX:     `STACKSIZE n[, commit]`

EXAMPLE:    `STACKSIZE 1024`

COMMENTS: The example sets stack size for the generated module to 1024 bytes, no matter what is specified on the command line when OPTLINK is run. *n* is the total stack space to reserve; it must be an integer, and should be an even number since stack operations are normally performed in units of 2-byte words. *commit* is the amount of memory to allocate each time the program runs. By default, *n* and *commit* are assumed to be decimal, but the C language conventions for hexadecimal and octal values can be used if desired.

## STUB

By default, OPTLINK adds a standard "stub" routine to the output executable file to terminate its operation with a warning message if a user attempts to execute the file under MS-DOS. This directive permits a substitute stub file to be defined for inclusion in the output. This can often be used to "stub" in a DOS version of the program, permitting the same .exe file to run under DOS and Windows. OPTLINK searches the directories listed in the PATH environment variable to locate the stub file, if it is not in the current working directory.

SYNTAX:     STUB 'filename' or STUB:NONE

EXAMPLE:    STUB 'newstub.exe'

COMMENTS: The example causes file `newstub.exe` to be inserted into the output executable file for execution under MS-DOS. If `newstub.exe` is not found in the current directory while linking, the list of directories specified in the `PATH` environment variable will be searched. If the file cannot be found, a warning is issued. If the file is found, but is not a valid MS-DOS .exe file, an error occurs.

If the keyword `NONE` is used, no stub at all will be included in the output file. This generates a slightly smaller .exe file, but one which will crash with no warning if an attempt is made to run it under MS-DOS.

## **VERSION**

Equivalent to the `/VERSION` switch; see "OPTLINK Switches" for information.

## **Linker Operation**

The purpose of a linker is to convert object files into an executable form by resolving address references that span object module boundaries. This makes it possible to combine a number of distinct object modules into a single executable program. In the absence of such a capability, it would not be possible to create today's complex systems.

The file format accepted by the linker is rigidly defined; each file contains all information required to resolve all address references contained within it, although this may be simply a reference to some "external" symbol. As the linker processes successive object modules, such external references may be resolved. If any remain unresolved after the last module has been processed, the link operation has failed.

## **Modularly Designed Programs**

Most programs are modularly organized into relatively small sections that perform a single function or procedure. Each of these smaller sections may, in turn, call upon others, so that the final program becomes hierarchical.

Organizing programs in this manner maximizes the opportunity to re-use code. For example, once you (or someone else) have created a function to input a single keystroke from the keyboard, you never need to re-create it. All you have to do is to refer to the function whenever you need to get keyboard input.

For this strategy to work, it's necessary that organization and formats follow certain rules. These rules were set forth, for "object modules", by Intel in their specification for Object Module Format (OMF),<sup>o</sup> and have been extended by other firms (notably Microsoft and Borland).

In addition to making it easy to re-use common functions and procedures, modular design has another significant advantage: when you change a program, only those functions that were actually changed need to be re-compiled. The entire collection of object modules that make up your program are re-linked.

## Source File Design

To organize a program into a series of object modules, the place to start is with the program's source file(s). Most high-level language compilers create a single object file from a single source file, no matter how many different functions or procedures that file contains.

Originally, if you placed all your functions and procedures into the same source file, the compiler would put them all into a single object module. Then, when you need only one of those object modules later, you'll find that all the other object modules are loaded along with the one you wanted. To overcome this, it was necessary to break the source file into a number of smaller source files, each containing only one (or a few, related) functions. Organizing source files in this manner tended to complicate maintenance of the program, but simplified re-use of the code.

The preceding paragraph was true until the introduction of "smart linking". It is now possible to generate encapsulated functions within a single object module, yet have only those functions which actually are used by a specific program linked into the executable file. OPTLINK recognizes the special records that make this possible and treats them correctly. However, not all language processors yet generate the special COMDAT records that make smart linking work.

A compromise that addresses both problems is to combine the smaller source files into a single larger file for storage and while editing, but then split it into smaller files before compiling or assembling it. Details of doing this are outside the scope of this manual; the thing to keep in mind is the trade-off between ease of maintenance and ease of re-use, so that you can plan your projects for maximum effectiveness in both areas.

When you run source files through a compiler or assembler, the normal output is one or more object modules per file. These individual modules must still be linked; that is, they must all be combined into a single executable program, with memory locations assigned to each symbol in each module, and all references within each module to symbols which are defined in other modules must be resolved. This task is a function of OPTLINK.

## **Object Module Library Design**

When a complex program is properly divided into its component sections, the number of object modules involved can easily become huge. Just keeping track of all these modules can become a serious problem.

To solve this problem, the concept of an "object module library" or "library" was invented. A library is a collection of object modules combined into a single file, together with an index that makes it possible to quickly locate any module contained in the library.

Most high-level languages include one or more "run time library" files as a part of their package, and make extensive use of the modules it contains. In addition, general software products such as screen display utilities may be sold as add-on libraries. And you can combine your own object modules into libraries as well. Thus using libraries is the way to simplify tracking the numerous modules involved in a complex program.

## Library Searching

While collecting object modules into library files makes it simpler to keep track of the many modules involved in a typical program, the use of library files imposes some constraints on the linker program's operation. Only those modules that are required by the program being linked need actually be included in the final executable file. A library contains some form of indexing so that the linker can locate requested modules readily, without having to search through every object module stored in the library.

Typically, an object module's original source contains references to symbols which may be in other object module's source compiled at approximately the same time, and also contain references to symbols defined in modules within the libraries being used. These are known as "external" symbol references, and each time an actual memory location is assigned to one of these symbols, it is said to "resolve" the symbol reference.

If any external symbol references remain unresolved after all object modules have been processed, the linker searches for their definitions within any libraries that have been specified. Each object module specified to the linker for inclusion (whether specified explicitly, or by being located during a library search) may resolve previous external symbol references, but it may also introduce new ones that will require resolution.

OPTLINK will search any number of libraries in order to resolve external references that remain after all supplied object modules have been processed. The library files to be searched may be specified either by means of commands embedded in the object modules, or by explicit commands to OPTLINK. If any references remain unresolved after all libraries have been searched, OPTLINK reports an error, but can still create the executable file (see `/ONERR` and `/ERRORFLAG` options).

A significant difference between OPTLINK and other linkers is that OPTLINK always resolves external references from the first library (in the supplied or default list of library files) that contains a definition, even if the reference itself occurs in a subsequent library. Microsoft LINK and many other linkers resolve such a reference by using the first definition found after the reference.

## Reading Object Modules

The first action taken by OPTLINK (after setting all applicable switches for the current run) is to read all .obj files, in the sequence in which they are specified in the FILE command. You control the sequence in which files are read by the sequence in which you provide them to OPTLINK. In some cases, this is significant. As OPTLINK reads the files, it collects information about sizes, segments, and symbols, for use in the later stages of linking.

OPTLINK searches for each file first in the current working (default) directory, then in the directories named by the `OBJ` environment variable, and finally in the directories named by the `LIB` environment variable.

If a requested object module cannot be found, operation terminates with a fatal error.

## Search Library Link

After all object modules named in the input have been read, OPTLINK then searches through all applicable library (`.lib`) modules while any `EXTERN` symbols remain undefined.

The modules first searched are those named in the command line or supplied interactively, followed by those named in the input data (searched in the order in which they were named). The first `PUBLIC` symbol encountered that matches an undefined `EXTERN` is used and any subsequent occurrences of that symbol as a `PUBLIC` are ignored, so the sequence in which libraries are searched may have significant impact on a program's operation.

After all named libraries have been searched, or if no libraries are named in the input, the libraries called for by internal records of the object modules (i.e. requested by the translator which generated the object modules) are searched unless this capability has been turned off by the use of the `/NODEFAULTLIBRARYSEARCH` command.

In library searches, when no path is specified with the library name, the current default directory is searched first when looking for any specific `.lib` file. If none is found there, the path(s) listed in the `LIB` environment variable are used. If a requested library module cannot be found, a warning message is issued but operation continues.

## Assigning Segment Addresses

Physically, every segment must begin on a paragraph boundary (an address of which the low four bits are all zero). Every segment referenced in a program is identified by two things: a segment name and a class name.

The segment name is assigned by you, if you write in assembly language, when you use the `SEGMENT/ENDS` declarations. If the object module was generated by a high level language, the segment name is assigned by the translator. The class name is also supplied by you, by means of a modifier you may add to the `SEGMENT` declaration. If given, the class name is enclosed in single quotes, as in:

```
CODESEG          SEGMENT          PUBLIC BYTE 'CODE'
```

In this example, the segment name is `CODESEG` and the class name is `CODE`.

Unless a `/PACKCODE` or `/PACKDATA` option switches are used, `OPTLINK` combines segments having matching segment and class names based on their `combine type`, which may be `PUBLIC`, `COMMON`, `PRIVATE`, or `STACK`. `PUBLIC` segments combine into a single bigger segment; `COMMON` segments are assigned the same address (that is, all use the same memory, at the same time), and `PRIVATE` segments are not combined at all. The final attribute of a segment is its alignment, which may be `BYTE`, `WORD`, `DWORD`, `PARA`, or `PAGE` (corresponding to boundaries at multiples of 1, 2, 4, 16, or 256, respectively).

Within each program, all segments of the same class are loaded in memory adjacent to each other. If no alignment is specified, `PARA` is used.

## **Segment Re-Ordering**

Segments may be, and often are, collected into groups by language translators. The difference between segments in a group and those which are not is that, in a group the applicable segment register is not changed when moving from one segment to another within the group. If groups are not used, code to change the segment register is required whenever the segment changes. Like segments themselves, the size of any group may not exceed 65,536 bytes.

## **Collecting Relocation Information**

When all the segments, groups, combine types, and classes have been sorted out and processed appropriately, addresses are assigned to all the segments. With all addresses known, relocation tables can be generated (internal to OPTLINK) which are then used to reconcile address references.

## Assigning Public Addresses

All `public` symbols within any object module are identified as such, and only such `public` symbols can be addressed from outside the module. OPTLINK keeps track of the segment and offset values for each `public` symbol encountered, both while reading the object modules and when a library search locates a module that contains a needed `extrn` symbol declared to be `public`.

## Reconciling Address References

Every standard `.obj` module normally contains several records devoted to relocation information. These records, identified in the OMF documentation as type `FIXUPP`, are universally called "fix-ups."

Fix-ups are processed after all segment addresses and symbol references are known, to perform final reconciliation of cross-module references.

## **Writing Output Files**

When all address references have been reconciled, OPTLINK writes the output executable file and any requested report files.

## **OPTLINK Overview**

OPTLINK, the Symantec linker, links object files and libraries to produce an executable program or DLL for DOS, Windows 3, or Win32. Most of the time you probably won't need to run OPTLINK on its own, because the compiler automatically calls OPTLINK unless you specify otherwise.

However, if you need more control over how your files are linked than you can get by setting options in the Linker subpage on the Build page of the IDDE's Project Settings dialog box, run OPTLINK separately and link the files yourself.

### **What is OPTLINK?**

OPTLINK is a super-fast, high-capacity program linker. It is designed for use by professional programmers to efficiently create both standard `.exe` and `.com` executable files for use with MS-DOS and segmented `.exe` files (segmented executable) and dynamic link libraries (`.dll`) for use with Windows and Win32 in significantly less time than is required by other linkers.

### **What does OPTLINK do?**

OPTLINK combines program and library files into a single file that is executable by the operating system. OPTLINK may also, at your option, create memory maps and symbol tables that you can use to assist in debugging your application.

OPTLINK works with modules in Intel OMF format (`.obj` file), including those generated by standard assemblers and compilers and standard `.obj` module libraries. The output files produced have all address assignments resolved and are in one of three standard formats: the `.exe` relocatable executable format, the `.com/.sys/.bin` memory image format (both apply only to MS-DOS output files), the 16-bit segmented `.exe` format used by Windows, or the 32-bit `.exe` format used by DOSX or Win32.

### **How OPTLINK works**

All available memory is used by OPTLINK when creating your executable program. In 16-bit programs, the number of program description objects that can be accommodated is limited to 64KB for each kind of object (PUBLIC, LIBRARY, SEGMENT, and so on). If memory space is inadequate, the program descriptions are automatically paged out to disk files which are then deleted once the link process completes.

These features combine to create an environment in which you can create large programs, despite apparent memory constraints.

## OPTLINK Features

Although compatibility with existing linkers was a prime objective in the design of OPTLINK, the product sets itself apart from other linkers with a number of exclusive features. OPTLINK features include:

- **Speed**  
Executes faster than any other method for creating executable files. Removal of duplicate records (CVPACK) is built in.
- **Capacity**  
Successfully builds huge applications, including debug information.
- **Smart linking**  
Unlike others, recursively eliminates unreferenced code. Also eliminates unreferenced uninitialized global variables, see `/PACKFUNCTIONS`.
- **Efficient program generation options**  
Builds smaller, more efficient, faster loading and even faster running 16-bit programs by packing code and data, plus converting intra-segment far calls to near calls, see `/PACKCODE`, `/PACKDATA`, `/EXEPACK`, `/FARCALLTRANSLATION`.
- **Cross referencing**  
Optionally included within `map` output. Every global symbol and the module name(s) of where defined and referenced is listed, see `/XREF`.
- **Environment use**  
Dedicated environment variable usage simplifies searches.
- **Virtual memory system**  
When DOS hosted, automatically uses EMS memory, XMS memory, or disk storage, whichever is available.

## Setup of the OPTLINK Linker

Reading the following will help you better understand OPTLINK.

### Option switch precedence

Reading the following will help you better understand OPTLINK. Any subsequent switch overrides any previous switch setting.

- The `optlinks.cfg` configuration file.
- The `LINK` environment variable.
- The `OPTLINKS` environment variable.
- Command line and indirect response file in the order supplied. Recognizes the most recent switch syntax selected.

### OPTLINK configuration

To specify default operating parameters not available through the IDDE, modify the file named `optlinks.cfg`. You can supply option switches that you may regularly use or desire as default. When called, `optlinks.cfg`, an ASCII file, is read by OPTLINK from its home directory, unless `optlinks.cfg` already exists in the current directory. OPTLINK recognizes the "#" as a comment character and thus ignores the remainder of the line.

### DOS environment variables

The environment variable names recognized are `LIB`, `TMP`, `LINK`, `OPTLINKS` and `OBJ`. OPTLINK also searches the `PATH` environment variable to locate certain executable files. The environment variables are typically defined in `autoexec.bat` or another batch file.

#### LIB

Supplies paths to search for libraries, object files, and resource files. See also `/SCANLIB`.

#### TMP

Supplies a path used by OPTLINK for its virtual memory file. Otherwise, the current directory is used. OPTLINK's performance can be severely slowed when its virtual memory file must be stored across a network.

#### LINK

Supplies linker switch settings. Recognized for Microsoft LINK compatibility. See also `/SCANLINK`.

#### OPTLINKS

Supplies linker switch settings. OPTLINKS is scanned after the `LINK` variable and can override any previous settings specified.

#### OBJ

Supplies paths to be searched by OPTLINK for `.obj` files not found in the current working directory. Paths specified by `OBJ` are always searched before any paths specified by the `LIB` environment variable.

#### PATH

Supplies paths to be searched by OPTLINK for any stub `.exe` file specified in the `.def` file.

For example, to select the linker option `/IGNORECASE` as the default, the following command could be used:

```
SET OPTLINKS=/IG
```

The environment variables can be overridden by command line input. Every switch can be changed, since it has an opposite.

## **DLL Version of OPTLINK**

The DLL version of OPTLINK is `linknd.dll`.

## Optimizing Linker Performance

For fast execution of the linker, follow these guidelines.

- Disable relocation checks by using the /NORELOCATIONCHECK option.
- If the /MAP option is being used to create a .map file, disable sorting of the map by using the /MAP:address option.
- If the /MAP option is being used to create a .map file, disable placement of line number information in the map by using the /NOLINENUMBERS option (the default).
- If the executable file is a Windows program, use the /RC option to add resources to the executable file.
- If you are running OPTLINK from a Windows DOS box, use the DPML version (`link.exe`) and set the DOS box .pif file parameters in `dosprmt.pif` as follows:  
XMS required: 0           XMS memory not locked (-1 desired)  
EMS required: 0           EMS memory not locked (-1 desired)
- Configure a small RAM disk so more EMS, XMS, or DPML memory is available for OPTLINK.
- Avoid linking to a network server.

## Number Representation

Several option switches require you to supply numeric values. These are indicated by the suffix ":*n*" or "*=n*" within the syntax description. You must separate the numeric value from the option by the appropriate colon or equals sign.

The numeric base is, by default, decimal. If the value begins with an 0, its base is taken to be octal, and if it begins with 0x or 0X, hexadecimal is assumed. That is, the standard C language base convention is assumed by OPTLINK.

## OPTLINK Option Switch Overview

Click an option switch name for more information.

The following is an overview of OPTLINK switches. Either a "-" (hyphen) or a "/" (slash) must precede an option switch. You can find further descriptions of the option switches and their possible opposites in "Using OPTLINK Option Switches" below.

### OPTLINK option switches

Option Switch	Purpose
<u>ALIGNMENT</u>	Segment alignment size
<u>BASE</u>	Set the base address of the executable image
<u>BINARY</u>	Generates a <code>.sys</code> file
<u>BYORDINAL</u>	Export by ordinal
<u>CHECKSUM</u>	Parsed and ignored
<u>CODEVIEW</u>	Outputs CodeView debugger information
<u>COMDEFSEARCH</u>	Specifies whether an undefined COMDEF causes a library search
<u>CPARMAXALLOC</u>	Sets <code>.exe</code> maximum bytes to occupy in DOS RAM
<u>CVVERSION</u>	Preserves OPTLINK's CodeView compatibility.
<u>DEBUG</u>	Controls all debug information for files that follow
<u>DEBUGFILES</u>	Controls debug information for specific files only
<u>DEBUGLINES</u>	Controls debug line number information for files that follow
<u>DEBUGLOCALS</u>	Controls debug local symbols information for files that follow
<u>DEBUGMODULES</u>	Controls debug information for specific modules only
<u>DEBUGPUBLICS</u>	Controls debug public symbols information for files that follow
<u>DEBUGTYPES</u>	Controls debug type information for files that follow
<u>DEFAULTLIBRARYSEARCH</u> <u>H</u>	Searches default libraries named in <code>.obj</code> files
<u>DELEXECUTABLE</u>	Deletes <code>.exe</code> upon encountering any linking errors
<u>DETAILEDMAP</u>	Produces detailed map reports.
<u>DOSSEG</u>	Controls segment sequence
<u>ECHOINDIRECT</u>	Controls echoing of indirect response file input
<u>EMSMAXSIZE</u>	Sets maximum EMS size
<u>EMSPAGEFRAMEIO</u>	Gives permission to use EMS page frame for I/O
<u>EMSUSE40</u>	Allow LIM 4.0 adherence
<u>ERRORFLAG</u>	Controls error flag in segmented

<u>EXEPACK</u>	executable header Performs run-length encoding (packs executable)
<u>EXETYPE</u>	Specifies the target operating system
<u>FARCALLTRANSLATION</u>	Converts intra-segment far calls to near
<u>FIXDS</u>	Identical to the <code>.def</code> directive <code>FIXDS</code>
<u>FIXED</u>	Fixes the executable image in memory
<u>GROUPASSOCIATION</u>	Controls <code>GROUP</code> information found in <code>.obj</code>
<u>GROUPSTACK</u>	Controls stack definition in <code>.exe</code> file header
<u>HEAP</u>	Sets the size of the local heap
<u>HELP</u>	Displays switch options
<u>IGNORECASE</u>	Preserves and ignores case of all symbols
<u>IMPDEF</u>	Generate <code>.din</code> file from <code>EXPORTS</code> section of <code>.def</code> file
<u>IMPLIB</u>	Create <code>.lib</code> import library for <code>.dll</code>
<u>INFORMATION</u>	Display status information throughout the link process.
<u>LINENUMBERS</u>	Outputs line number information in <code>.map</code> file
<u>LOWERCASE</u>	Converts all symbols to lowercase
<u>MACHINE</u>	Specifies the type of the target machine
<u>MAP</u>	Controls information content in <code>.map</code> file
<u>NOLOGO</u>	Suppresses OPTLINK's sign-on copyright message
<u>NONAMES</u>	Eliminates name text for ordinal exports
<u>NULLDOSSEG</u>	Outputs null bytes in the <code>_TEXT</code> segment
<u>ONERROR</u>	Same as <code>/DELEXECUTABLE</code>
<u>PACKCODE</u>	Combines segments of class <code>CODE</code>
<u>PACKDATA</u>	Combines data segments where possible
<u>PACKFUNCTIONS</u>	Performs "smart-linking" on code and data
<u>PACKIFNOSEGMENTS</u>	Forces <code>/PACKCODE</code> on when no <code>SEGMENTS</code> directive
<u>PACKSIZE</u>	Packs size for <code>/PACKCODE</code> and <code>/PACKDATA</code>
<u>PAGESIZE</u>	Set <code>/IMPLIB</code> page size

<u>PAUSE</u>	Provides time to swap disks while creating output
<u>PMTYPE</u>	Specifies type of segmented output
<u>PROMPT</u>	Specifies whether OPTLINK will prompt for more options
<u>RC</u>	Like <code>.def</code> directive <code>RC</code> , automatic resource binding
<u>RELOCATIONCHECK</u>	Ensures no relocation overlaps have occurred
<u>REORDERSEGMENTS</u>	Performs segment reordering
<u>SCANLIB</u>	Scans the <code>LIB</code> environment variable
<u>SCANLINK</u>	Scans the <code>LINK</code> environment variable
<u>SILENT</u>	Does not display linking status information
<u>STACK</u>	Defines stack segment and/or its size.
<u>STUB</u>	Adds a stub file to the executable
<u>TINY</u>	Generates a <code>.com</code> file
<u>UPPERCASE</u>	Converts all symbols to upper case
<u>VERSION</u>	Adds a version number to the executable
<u>WARNDUPS</u>	Warns of duplicate public symbols in <code>.lib</code>
<u>WINPACK</u>	Build compressed output utilizing decompressing loader
<u>XMSMAXSIZE</u>	Sets maximum XMS size
<u>XNOIGNORECASE</u>	Treats <code>EXPORT</code> and <code>IMPORT</code> symbols as case significant
<u>XREF</u>	Controls information content in <code>.map</code> file
<u>XUPPERCASE</u>	Forces <code>EXPORT</code> and <code>IMPORT</code> symbols to upper case
<u>?</u>	Like <code>/HELP</code> , displays available switches

## Unsupported Microsoft LINK Switches

OPTLINK does not recognize the following Microsoft LINK switches:

```

/DSALLOC
/DYNAMIC
/HIGH
/OLDOVERLAY
/OVERLAYINTERRUPT
/QUICKLIBRARY
/r

```

OPTLINK parses and ignores the following Microsoft LINK switches:

```

/BATCH
/CHECKABORT
/CHECKEXE
/NOEXTDICTIONARY

```

/PADCODE  
/PADDATA  
/SEGMENTS  
/WARNFIXUP

## **Using OPTLINK Option Switches**

While this section supplies the full name of each option switch, you need to supply only enough characters to uniquely identify it. These option switches are not case significant. Either a "-" or a "/" must precede an option switch. "Default" shows OPTLINK's assumptions before encountering any option switch or directive.

## ALIGNMENT

Syntax: /ALIGNMENT:n

Default: /ALIGNMENT:16

Win32 default: /ALIGNMENT:512

In the segmented `.exe` format, the `ALIGNMENT` value determines maximum file size because segments and resources are referenced by a 16-bit page number. Page size is set by `ALIGNMENT`, so if `n` is left at its default value of 16, the maximum file size will be 16\*65535 bytes, or 1 megabyte. To increase the file size limit, increase `n`. The value specified must be a power of 2.

When generating a conventional MS-DOS `.exe` files, this switch causes the `.exe` file header to be rounded up to the specified page size. `n` is the file header alignment size in bytes. Defaulting to 16 allows for the smallest `.exe` file size. Other linkers use a page size of 512, causing unnecessarily large `.exe` files. Some `.exe` post-processors (notably EXE2BIN) may require you to retain the 512-byte header size. The value specified must be a power of 2.

## **BASE**

Syntax: `/BASE:nn`

Sets the base address of the executable image to the hexadecimal value nn.

## **BINARY**

Syntax: `/BINARY[:n]`

Causes OPTLINK to generate a `.sys` file instead of an `.exe` file.

The value `n` is the lowest address generated in the `.sys` file. If you want your `.sys` file to start with code at `8000H`, use `"/BI:0x8000"` to define the lowest address. Defaults to 0 when not supplied.

## BYORDINAL

Syntax: `/BYORDINAL`

For 32-bit output, `/BYORDINAL` causes OPTLINK to export symbols by ordinal.

For 16-bit output, causes OPTLINK to move the name text for every exported symbol, designated by `__export` in source code, from the RESIDENT name table (which requires Windows memory) to the NONRESIDENT name table. Effectively, this option reduces memory requirements and load times for 16-bit programs. It also allows you to export more symbols from 16-bit `.exe` and `.dll` files, since the file format allows a maximum `.exe` header size of 64Kb, and RESIDENT names are part of that header. The NONRESIDENT table can be 65,535 bytes in size.

Use `/BYORDINAL` with `/IMPDEF`, `/IMPLIB`, and `/NONAMES` to optimize the size and speed of your 16-bit code. However, any additional modifications to entry points can cause ordinal changes that effectively necessitate rebuilding the entire application. To minimize the impact of this problem:

- Rebuild the entire application using `/BYORDINAL`, `/IMPDEF`, `/IMPLIB`, and `/NONAMES`. `/IMPDEF` creates a `.din` file (a `.def` include file) for the module being built. `/IMPLIB` generates the DLL's `.lib` file.
- For all subsequent builds, include the `.din` file (via the directive `INCLUDE 'dinfilename'`) for the respective module, plus continue using all four of the above OPTLINK switches. This will eliminate ordinal problems by preserving the `.def` settings in the `INCLUDE'd` file.

## CHECKSUM

Syntax: / [NO] CHECKSUM

Default: /NOCHECKSUM

Parsed and ignored.

## CODEVIEW

Syntax: `/[NO]CODEVIEW[:n]`

Default: `/NOCODEVIEW`

Controls whether OPTLINK should include CodeView information within the output `.exe` file. CodeView expects such information to be present within applicable `.obj` files being linked. OPTLINK includes any available CodeView information for all object modules unless the options `/DEBUG`, `/DEBUGCOVERAGES`, `/DEBUGFILES`, `/DEBUGLINES`, `/DEBUGLOCALS`, `/DEBUGMODULES`, `/DEBUGPUBLICS`, or `/DEBUGTYPES` are supplied.

`n` specifies the CodeView major version format desired (applicable only with `/CODEVIEW`). For CodeView 2.x, enter 2, for CodeView 3.x, enter 3, and for CodeView 4.x, enter 4. When you enter `/CODEVIEW` without a number, OPTLINK uses the default. OPTLINK automatically performs the CVPACK step when you use 4.

## **COMDEFSEARCH**

Syntax:    / [NO] COMDEFSEARCH

Default:   / NOCOMDEFSEARCH

Specifies whether OPTLINK performs a library search if it encounters an undefined COMDEF record.

## **CPARMAXALLOC**

Syntax: `/CPARMAXALLOC:n`

Default: `/CPARMAXALLOC:0xFFFF`

Only valid when generating MS-DOS `.exe` files. This switch sets the maximum number, `n`, of 16-byte paragraphs needed by a DOS program to occupy when loaded in memory. This functionality is useful for reserving memory needed for other programs.

If the program requires more space, OPTLINK sets the allowable RAM usage to the actual amount required by the program. If you do not use this switch, OPTLINK sets the RAM requirement to its maximum possible value, causing the program to use all available RAM at run-time. Acceptable values are 1 to 65,535.

## CVVERSION

Syntax: `/CVVERSION:n`

Default: `/CVVERSION:4`

Enhance the Microsoft LINK drop-in replacement capability of OPTLINK. Your existing make files or other Microsoft LINK command sets may assume a different version of CodeView. By placing this switch in `optlinks.cfg`, you avoid making any changes to established make files.

`n` specifies the CodeView major version format desired (applicable only when using `/CODEVIEW`). For CodeView 2.x, enter 2, for CodeView 3.x, enter 3, and for CodeView 4.x, enter 4. When you enter `/CODEVIEW` without a number, OPTLINK uses the default. OPTLINK automatically performs the CVPACK step when you use 4.

## DEBUG Options



Syntax: `/[NO]DEBUG[modifiers]`

Enable you to selectively include and exclude debug information within the target program. OPTLINK lets you select debug information for the entire program, specific object modules (see also `/DEBUGMODULES`), specific object files, and specific library files (see also `/DEBUGFILES`). Additionally, you can selectively include or exclude the debug information itself (i.e. line numbers, local symbols, public symbols, and types). These switches can save recompiling to turn debugging information on and off in certain modules. By default, when directed to output debug information, OPTLINK includes all debug information content. Use `/CODEVIEW` to inform OPTLINK of the desired debugger format.

If you supply no modifiers, the switch controls the inclusion of all debug information for any subsequently supplied files. You can also supply any of the following modifiers:

Supplying `/NODEBUG` turns off debug information generation. The switch must appear before selectively including debugging information for specific object files. For example:

```
LINK/CO:4/NODEBUG obj1 /DEBUGLINES obj2  
/DEBUG obj3;
```

This forces OPTLINK to output full CodeView v4.x debugging information for file `obj3` and only CodeView line number information for file `obj2`. It generates no debugging information for file `obj1`.

### **LINES**

Controls the inclusion of line number information in debug and map output for any subsequently supplied files.

### **LOCALS**

Controls the inclusion of local symbol information in debug output for any subsequently supplied files.

### **PUBLICS**

Controls the inclusion of public symbol information in debug and map output for any subsequently supplied files.

### **TYPES**

Controls the inclusion of type information in debug output for any subsequently supplied files.

Supplying `/NODEBUG` excludes debug information for specific object files. The switch must appear before selectively including debug information for specific files. For example:

```
LINK /CO:4 /NODEBUG obj1 /DEBUGLINES obj2 /DEBUG \ obj3;
```

This example generates full CodeView 4.x debugging information for file `obj3` and only CodeView line number information for `obj2`. No debugging information is generated for `obj1`.

**See Also**

DEBUGMODULES

DEBUGFILES

## DEBUGFILES

Syntax: `/DEBUGFILES:filename[[ :filename]...]`

Causes OPTLINK to include all debugging information from the selected files. This saves recompiling to turn debugging information on and off in certain modules or when your debugger can't handle all modules at once. Use `/CODEVIEW` to inform OPTLINK of the desired debugger format.

`filename`

File name to include debug information. OPTLINK includes debugging information in all object modules in the named object or library file. Any file name specified without an extension defaults to the extension `.obj`. You can enter multiple file names, but you must include complete paths and an additional blank before the colon, `:"myfile"`.

## DEBUGMODULES

Syntax: `/DEBUGMODULES:modname [ [:modname] . . . ]`

Causes OPTLINK to include all debug information from the selected object modules. This saves recompiling to turn debugging information on and off in certain modules or when your debugger can't handle all modules at once. OPTLINK includes only the named object modules from the file. Use `/CODEVIEW` to inform OPTLINK of the desired debugger format.

`modname`

Object module name(s) to include debug information.

## DEFAULTLIBRARYSEARCH

Syntax:    / [NO] DEFAULTLIBRARYSEARCH [ : libname]

Default:   /DEFAULTLIBRARYSEARCH

Controls how OPTLINK handles comment records in .obj file requesting a library search. The default setting causes OPTLINK to search all requested libraries. Otherwise, OPTLINK does no searching.

libname

Applicable only with /NODEFAULTLIBRARYSEARCH. Specifies a library not to be searched. For example,

"/NODEFAULTLIBRARYSEARCH:CLIBC" causes OPTLINK to ignore requests for CLIBC.LIB. You must provide a separate copy of the switch for each library name.

## **DELEXECUTABLE**

Syntax:    / [NO]DELEXECUTABLE

Default:   /NODELEXECUTABLE

Controls whether OPTLINK should delete the .exe or .dll output file upon encountering any link error.

## DETAILEDMAP

Syntax:    / [NO] DETAILEDMAP

Default:   /NODETAILEDMAP

Controls the amount of detail provided in the segment list produced at the beginning of the .map file.

/DETAILEDMAP provides much more detail about segment type, which modules added, how many bytes to the segment, and where each module begins.

## DOSSEG

Syntax:    / [NO]DOSSEG

Default:   /NODOSSEG

Controls `.exe` file segment sequence. When used, the segment sequence is: segments with a class name ending in `CODE`, then all segments not included in `DGROUP`, and, lastly, all segments of `DGROUP` (which will in turn be sequenced as `BEGDATA`, except `BEGDATA`, `BSS` and `STACK`, `BSS`, and finally `STACK`). Otherwise, segments appear in the sequence in which they are encountered while linking.

## **ECHOINDIRECT**

Syntax:    / [NO]ECHOINDIRECT

Default:   /NOECHOINDIRECT

Controls the echoing of indirect response file input lines to the console.

## **EMSMAXSIZE**

Syntax:    /EMSMAXSIZE:n

Default:   /EMSMAXSIZE: uses maximum available

For 16-bit real mode hosted versions only. Controls the maximum number of kilobytes of EMS that OPTLINK uses. n set to 0 forces no EMS use.

## **EMSPAGEFRAMEIO**

Syntax:    / [NO]EMSPAGEFRAMEIO

Default:   /EMSPAGEFRAMEIO

For 16-bit real mode hosted versions only. Most EMS memory systems are reliable when doing I/O to their page frame. Not quite so rare are disk cache programs that assume they are the only ones using the page frame and thus become confused when another program does. If you use one of these programs, supply /NOEMSPAGEFRAMEIO to stop the linker from doing I/O in the EMS page frame area. Normally, /EMSPAGEFRAMEIO is the fastest operation.

## **EMSUSE40**

Syntax:    / [NO]EMSUSE40

Default:   /EMSUSE40

Controls whether OPTLINK adheres to LIM 3.2. Some EMS memory managers claim to be LIM 4.0 compatible but are not. If you encounter problems using EMS memory, try forcing the linker to comply to LIM 3.2 by supplying /NOEMSUSE40. If the problems go away, report the problem to your EMS driver vendor. This option is normally /EMSUSE40 (LIM 4.0) for fastest operation.

## **ERRORFLAG**

Syntax:    / [NO]ERRORFLAG

Default:   /ERRORFLAG

Controls the setting of the error flag in a segmented-EXE or Win32 file header. When used, OPTLINK sets the flag when it encounters errors in the link process. Otherwise, OPTLINK does not set the flag even when it detects errors.

## EXEPACK

Syntax:    / [NO] EXEPACK [:n]

Default:   /NOEXEPACK

Causes OPTLINK to run-length encode the generated `.exe` output file to reduce its size. It also compresses relocation entries to produce much smaller `.exe` files, especially in medium and large memory models. The EXEPACK technique is fully compatible with the `/RC` option of OPTLINK.

Microsoft's Resource Compiler is not compatible.

`n` is applicable only with `/EXEPACK` on DOS executables. It specifies how many bytes at the beginning of the memory image are not to be compressed. This permits easier maintenance of patch, configuration, serialization, and similar areas in the `.exe` file. The default value is 0.

## **EXETYPE**

Syntax: /EXETYPE:[OS2 | WINDOWS [n.nn] | DOS4 |  
UNKNOWN | DOS | NT]

Specifies the operating system under which the current executable file is to run. This option works like the EXETYPE definition file directive; see its description in the section "Definition File Directives" for information.

## **FARCALLTRANSLATION**

Syntax:    / [NO] FARCALLTRANSLATION

Default:   /NOFARCALLTRANSLATION

Causes far calls to targets in the same segment to be converted to the equivalent combination of `PUSH CS`, `NOP`, and `CALL NEAR`. This can improve code performance and reduce `.exe` file size, since it eliminates the need for relocation entries. It also converts far jumps where possible. Supplying `/FARCALLTRANSLATION` forces `/RELOCATIONCHECK`.

## **FIXDS**

Syntax:    / [NO]FIXDS

Default:   /NOFIXDS

This switch, applicable only to 16-bit Windows executables, does the same action as FIXDS directive in the `.def` file.

## **FIXED**

Syntax: `/FIXED`

Causes the executable image to be fixed in place in memory.

## **GROUPASSOCIATION**

Syntax:    / [NO] GROUPASSOCIATION

Default:   / GROUPASSOCIATION

Controls the use OPTLINK makes of GROUP information found in the .obj files. When used, OPTLINK uses all such information found in the files.

## GROUPSTACK

Syntax:    / [NO] GROUPSTACK

Default:   /NOGROUPSTACK

Controls stack definition in the standard DOS `.exe` file header. When used, OPTLINK writes values into the header in a group-based format that does not require your programs to modify the value of SS after loading. This action is not compatible with the output of most high-level languages; their startup code assumes that the stack is wrong, and adjusts it back into DGROUP. Thus, it is recommended to use this switch only with assembly language programs. Otherwise, OPTLINK writes these values in physical segment : *offset* form just as other linkers do, which effectively takes SS out of DGROUP.

## HEAP

Syntax: `/HEAP [ :n, commit]`

Sets the size of the executable file's local heap to n bytes. This option works like the `HEAPSIZE` definition file directive, except that the `MAXVAL` keyword is not supported. See the description of `HEAPSIZE` in the section "Definition File Directives" for information.

## HELP

Syntax: /HELP

Displays a list of available switches. In the list, those characters required to identify each switch appear in uppercase, while characters appearing in lowercase are optional and may be omitted.

## **IGNORECASE**

Syntax:    / [NO] IGNORECASE

Default:   / IGNORECASE

Forces OPTLINK to preserve but ignore the cases of identifiers and symbols, including the optional symbol table that may be appended to the output file. Only one of /IGNORECASE, /NOIGNORECASE, /LOWERCASE, and /UPPERCASE can be in effect for any specific run of OPTLINK.

## **INFORMATION**

Syntax: /**INFORMATION**

Default: /**SILENT**

This switch causes OPTLINK to display status information throughout the link process. It is the opposite of /**SILENT**.

## IMPDEF



Syntax: `/IMPDEF[:dinname]`

Causes OPTLINK to generate a `.din` file that combines information from the EXPORTS section of the `.def` file with `__export` directives, `/BYORDINAL` or `/NONAMES` switches. `dinname` is the `.din` file to be generated.

Use `/IMPDEF` with `/BYORDINAL`, `/IMPLIB`, and `/NONAMES` to optimize the size and speed of your 16-bit code. See the description of `/BYORDINAL` for information.

**See Also**

BYORDINAL

## IMPLIB



Syntax: `/IMPLIB[:libname]`

Builds an import library (`.lib`) file describing the exported symbols available to be imported from a dynamic link library (`.dll`), thus eliminating the need to run a separate utility to build the library. `libname` is the import library to build.

Use `/IMPLIB` with `/BYORDINAL`, `/IMPDEF`, and `/NONAMES` to optimize the size and speed of your 16-bit code. See the description of `/BYORDINAL` for information.

**See Also**

BYORDINAL

## **INFORMATION**

Syntax: /INFORMATION

Default: /SILENT

This switch causes OPTLINK to display status information throughout the link process. It is the opposite of /SILENT.

## **LINENUMBERS**

Syntax:    / [NO] LINENUMBERS

Default:   /NOLINENUMBERS

Controls whether line-number information shall be contained within the `.map` file.

## **LOWERCASE**

Syntax:    /LOWERCASE

Default:   /IGNORECASE

Forces OPTLINK to convert symbols to lowercase. Only one of /IGNORECASE, /NOIGNORECASE, /LOWERCASE, and /UPPERCASE can be in effect for any specific run of OPTLINK.

## **MACHINE**

Syntax: `/MACHINE:type`

Specifies the type of the target machine. type can be i386.

## MAP

Syntax: `/[NO]MAP[:modifiers]`

Default: `/NOMAP`

Causes the addition of public symbols to the `.map` file. The `/MAP` option normally outputs the symbol twice: sorted alphabetically and by increasing addresses. It accepts any of the following modifiers:

## **ADDRESS**

Outputs only the increasing address sort listing.

## **FULL**

Produces detailed maps as does the `/DETAILEDMAP` switch.

**GROUP**

Displays the GROUP column on the report.

## **NOGROUP**

Does not display the GROUP column. /MAP defaults to :NOGROUP.

## **NOCVPACK**

Syntax:    /NOCVPACK

This switch suppresses automatic execution of the CVPACK step after a successful link for CodeView Version 4. This allows you to run Microsoft's CVPACK utility. By default, OPTLINK performs the CVPACK step when linking for CodeView 4.

## **NOLOGO**

Syntax: `/NOLOGO`

This switch suppresses OPTLINK's normally displayed sign-on copyright message, unless errors are detected.

## NONAMES



Syntax: `/NONAMES`

Eliminates storage of name text for symbols exported by ordinal. This option can shrink `.exe` or `.dll` files by up to 64K bytes for 16-bit files. It also eliminates the 64K limit on exported names text for 16-bit files.

Use `/NONAMES` with `/BYORDINAL`, `/IMPLIB`, and `/IMPDEF` to optimize the size and speed of your 16-bit code. See the description of `/BYORDINAL` for information.

**See Also**

**BYORDINAL**

## **NULLDOSSEG**

Syntax:    / [NO]NULLDOSSEG

Default:   /NULLSDOSSEG

Controls the generation of 16 null bytes at the beginning of the \_TEXT segment in DOSSEG mode. The Microsoft run-time usually requires this.

## **ONERROR**

Syntax: `/ONERROR:NOEXE`

Default: `/NODELEXECUTABLE`

Causes OPTLINK to delete the `.exe` or `.dll` output file if any link error occurs. Normally the `.exe` file is not deleted upon an error.

## **OPT**

Syntax: `/OPT`

Uses OPTLINK syntax. Parsed and ignored.

## PACKCODE

Syntax: / [NO] PACKCODE [ :n]

Default: /NOPACKCODE

Causes OPTLINK to combine any segments of class name ending in CODE, to a maximum size of n bytes. The main purposes of this are:

- To improve operation of the /FARCALLTRANSLATION option, since more targets will be in the same segments, and
- To reduce the segment count for segmented .exe files. Some debuggers also function better when the segment count is kept low. /PACKCODE causes segments to be logically combined (packed) into larger physical segments.

**Note:** Applicable only with /PACKCODE is n: the maximum segment size of groups. /PACKSIZE controls the default value. Values of 8192 or 16384 may provide better performance for large applications because the segments being swapped, discarded, and reloaded are smaller.

## PACKDATA

Syntax:    / [NO] PACKDATA[:n]

Default:   /NOPACKDATA

Causes OPTLINK to combine data segments if possible, to produce larger, but fewer, data segments.

/PACKDATA causes segments to be combined (packed) into larger physical segments.

**Note:** Applicable only with /PACKDATA is n: the maximum segment size of groups. /PACKSIZE controls the default value. Values of 8192 or 16384 may provide better performance for large applications because the segments being swapped, discarded, and reloaded are smaller.

## **PACKFUNCTIONS**

Syntax:    / [NO] PACKFUNCTIONS

Default:   / PACKFUNCTIONS

Enables true smart linking of object files containing COMDAT records. OPTLINK retains only referenced COMDAT records. OPTLINK is fully recursive in this support; it removes unnecessary COMDATs and library modules from the executable.

## **PACKIFNOSEGMENTS**

Syntax: `/[NO]PACKIFNOSEGMENTS`

Default: `/PACKIFNOSEGMENTS`

Forces `/PACKCODE` on Windows output if the `.def` file does not use a `SEGMENTS` directive. MS-LINK has an undocumented

feature/bug in which `/PACKCODE:65500` is enabled for Windows `.exe/.dlls` if the `.def` file does not use a `SEGMENTS` directive. This option is enabled by default for compatibility. You may want to disable it.

## **PACKSIZE**

Syntax: `/PACKSIZE:n`

Default: `/PACKSIZE:65500`

Sets the default size (in bytes) for the `/PACKDATA` and `/PACKCODE` switches.

## PAGESIZE

Syntax: `/PAGESIZE:n`

Default: `/PACKSIZE:16`

Sets the page size for the `/IMPLIB` link option. The default value is 16, which allows a `.lib` file of up to 1Mb in size. `n` must be a power of 2 greater than or equal to 16.

## **PAUSE**

Syntax: / [NO] PAUSE

Default: /NOPAUSE

Allows swapping of diskettes just before OPTLINK writes binary output data.

## **PMTYPE**

Syntax: `/PMTYPE: type`

Default: `/PMTYPE: PM`

This switch, used only when producing segmented executable or `.dll` output, requires one of three type specifiers.

## **PM**

The application is Windows or PM and can execute only in the Windows or PM environment. For Windows, this eliminates the need to use RC if there are no resources. This format is the default output from OPTLINK.

## PROMPT

Syntax: / [NO] PROMPT

Default: / PROMPT

Determines whether OPTLINK prompts for input if additional link switches were expected (that is, if no semicolon (;) appeared at the end of the list of switches).

## RC Options

Syntax: `/RC[options] [:filename]`

Provides an alternative to the RC statement in the `.def` files. The `.res` file is assumed to be the same primary name as the `.exe` or `.dll`. Effective only when the switch is supplied. You can specify multiple instances of `/RC`.

This switch accepts the same eight options (each preceded by "-", a hyphen) accepted by the RC directive in the `.def` file; They have the following meanings:

- e Creates a driver that uses EMS memory
- k Keep segments in `.def` file order; does not sort or create a fast-load section
- l Creates an LIM 3.2 EMS application
- m Sets multiple instance flag
- p Create a private library (`PRIVATELIB`).
- t Creates a protected-mode-only application (`PROTMODE`).
- 30 Tags the executable as requiring Windows 3.0 or later to run
- 31 Tags the executable as requiring Windows 3.1 or later to run

`filename`

The optional file name. You can use `/RC` without a *filename* to mark `.exe` or `.dll` files as valid Windows files (ignore the file-not-found warning that results). It also causes preload segment reordering to occur.

## **RELOCATIONCHECK**

Syntax:    / [NO]RELOCATIONCHECK

Default:   /RELOCATIONCHECK

This switch causes OPTLINK to do extra processing to ensure that no relocation overlaps occur. Although rare, such overlaps can occur under the following circumstances: when initializing common blocks in more than one module or ORGing back over a relocation item in assembly language or when a translator error occurs.

## REORDERSEGMENTS

Syntax:    / [NO] REORDERSEGMENTS

Default:   / NOREORDERSEGMENTS

Allows OPTLINK to perform segment reordering. First, it orders all grouped segments so that segments belonging to a particular group are contiguous. It then orders all non-grouped segments so that code segments (classes ending in CODE) are together, and, therefore, all data segments are together. This allows the maximum benefit from

/PACKCODE and /PACKDATA.

## SCANLIB

Syntax: / [NO] SCANLIB

Default: /SCANLIB

Forces OPTLINK to scan the `LIB` environment variable for additional library paths. You can use the `LIB` environment variable to aid in the drop-in-replacement use of OPTLINK. This option works only in the `optlinks.cfg` file, since OPTLINK scans environment variables before scanning the command line.

## SCANLINK

Syntax: / [NO] SCANLINK

Default: /SCANLINK

Forces OPTLINK to scan the LINK environment variable for additional option switches. You can use the LINK environment variable to aid in the drop-in-replacement use of OPTLINK. When supplied, OPTLINK scans LINK before scanning the OPTLINK environment variable. This option works only in the `optlinks.cfg` file, since OPTLINK scans environment variables before scanning the command line.

## **SILENT**

Syntax: /SILENT

Default: /SILENT

This switch causes OPTLINK to suppress status information throughout the link process. It is the opposite of /INFORMATION.

## STACK

Syntax: `/STACK[:n, commit]`

Controls the stack size. If supplied, it also defines a stack segment (in DGROUP if you use `/DOSSEG`) if one isn't specified in the `.obj` file.

`n` specifies the size of the stack in bytes. You must specify this number; in 16-bit code it must not exceed decimal 65535.

## **STUB**

Syntax: `/STUB:filename` or `STUB:NONE`

Specifies a "stub" routine to be added to the output executable file to terminate its operation with a warning message. This option works like the `STUB` definition file directive; see its description in the section "Definition File Directives" for information.

## TINY

Syntax: /TINY

This switch causes OPTLINK to generate a .com file instead of an .exe file. The .com file automatically starts at offset 0100h in the code segment. To generate memory image files that start at other locations, use the /BINARY option switch.

## **UPPERCASE**

Syntax: /UPPERCASE

Default: /IGNORECASE

Forces OPTLINK to convert symbols to upper case. Only one of /IGNORECASE, /NOIGNORECASE, /LOWERCASE, and /UPPERCASE can be in effect for any specific run of OPTLINK.

## **VERSION**

Syntax: `/VERSION:nn[.mm]`

Puts the specified version number in the executable file. nn is the major version number; mm is the minor version number.

## **WARNDUPS**

Syntax:    / [NO]WARNDUPS

Default:   /NOWARNDUPS

Forces OPTLINK to output warning messages upon encountering duplicate library symbols. Very useful in debugging complex applications.

## WINPACK



Syntax: `/[NO]WINPACK`

Compresses 16-bit Windows `.exe` or `.dll` output files and adds a decompressing loader to the file. `/WINPACK` typically decreases the size of `.exe` and `.dll` output files by 50%. This option requires the use of OPTLINK's `/RC` link option (rather than using other resource binders).

To obtain maximum compression:

- For best results, pack all the executable files and DLLs that make up an application.
- Turn off debug output.
- Use `/PACKDATA:0` for packing to full 64K data segments.
- Use the largest `/PACKSIZE` possible. 16K is good if you need small segments. 64K is best for compression.
- Use a smaller `/ALIGNMENT` size.
- Export and import by ordinal number, and eliminate name text. See the `/BYORDINAL` and `/NONAMES` link options.

If your application contains `.res` files, you must use OPTLINK to bind them to the packed executable. `/WINPACK`'s decompressing loader must be segment number 1 in an `.exe` or `.dll` file, leaving 253 segments available for the application (one less than without `/WINPACK`). All segments are compressed, so if some value or string is modified after linking (for example, a product serial number), it will have to be located somewhere other than a code or data segment, perhaps in a resource.

For DLLs that cross-reference one another (for example, A.DLL references items in B.DLL, and B.DLL references items in A.DLL), neither `DGROUP` nor the code segment containing the start address can contain references to a DLL it cross-references.

Since `/WINPACK` compresses segments and relocations but not resources, the output file may be made even smaller by running it through another compression program after linking.

**See Also**

BYORDINAL  
NONAMES

## **XMSMAXSIZE**

Syntax: `/XMSMAXSIZE:n`

Default: `/XMSMAXSIZE` uses maximum available

Controls the maximum number of kilobytes of XMS memory that OPTLINK uses. n set to 0 forces no XMS use.

## **XNOIGNORECASE**

Syntax: /XNOIGNORECASE

Default: /XNOIGNORECASE

Forces OPTLINK to treat EXPORT and IMPORT symbols as case significant. Only one of /XNOIGNORECASE and /XUPPERCASE can be in effect for any specific run of OPTLINK.

**Note:** By default, the case of IMPORT and EXPORT symbols is significant. (In previous versions of Symantec C++, the default was /XUPPERCASE.) If a working application now generates run-time errors pertaining to undefined symbols, the case of symbols in some modules or libraries might be mismatched. Try rebuilding the application using the /UPPERCASE option, or declare the EXPORT and IMPORT symbols in definition files the same way they appear in object files.

## **XREF**

Syntax:    / [NO]XREF

Default:   /NOXREF

Controls whether OPTLINK adds cross reference information to the `.map` file. That is, for every global symbol, OPTLINK lists the module name where defined, followed by every other module name that referenced it.

## **XUPPERCASE**

Syntax: /XUPPERCASE

Default: /XUPPERCASE

Forces OPTLINK to convert `EXPORT` and `IMPORT` symbols to upper case. Only one of `/XNOIGNORECASE` and `/XUPPERCASE` can be in effect for any specific run of OPTLINK.

## **? (question mark)**

Syntax: `/?`

Identical to `/HELP`. Displays a list of available switches. In the list, those characters required to identify each switch appear in uppercase, while characters appearing in lowercase are optional and may be omitted.

## Operational Modes

OPTLINK's available operational modes are: command line, interactive, and indirect file.

All input on the command line should be in a series of parameters. For the interactive mode at a console, use `link.exe`. Using indirect file operation, you can enter prompt-like responses into a file. The specific file name is passed to OPTLINK on the command line and thus read by OPTLINK for its linker input.

In all modes of operation, OPTLINK requires that you provide the same kinds of control information, in the form of OPTLINK commands. This section describes the methods by which your commands may be transmitted to OPTLINK.

## Command Line and Interactive Operation



Command line operation of OPTLINK uses the following syntax:

```
LINK obj[, out[, map[, lib[, def[, res]]]]]
```

For interactive operation simply input `LINK` followed by any option switches, which are described later, and press Enter. OPTLINK then prompts you for responses (option switches may be entered after any response).

OPTLINK parameters include the following:

### **obj**

One or more input (`.obj`) file names. If multiple names are supplied, they may be joined by "+" characters or blank spaces, but the comma separates the four parts of the command line. If more names are required than are able to fit onto the command line, a "+" immediately before the end of the line makes the list continue to the next line.

You may use ambiguous names wild-card characters "?" and "\*" If specific names are preceded or followed by wild-card references, those modules specifically named will not be included twice. At the end of the input list, you may use specific data or a "+" character to invoke the default choices. If neither is supplied, operation switches to the interactive mode, and you are prompted for the remaining inputs. The "#" character tells OPTLINK to ignore the rest of the comment characters following on that line.

### **out**

Specifies the name for the output executable or dynamic link library. If no extension is supplied, an `.exe` file is generated; if the extension is `.com`, a `.com` file is created automatically, and if extension is `.sys`, a binary device driver format file is created. To skip this entry, cause OPTLINK to default to the executable type provided in a definition file or option switch, supply a comma or press the Enter key in the interactive mode. If no directive or option switch is supplied, the default file name (that of the first `.obj` file) with `.exe` extension will be created.

### **map**

The file name for `map` output. Using an Enter key while in the interactive mode suppresses all map output, unless previously configured with a `/MAP` suggested file name. See the related switch descriptions (`/MAP`, `/XREF`, `/LINENUMBERS`, or `/DETAILEDMAP`) for more information.

### **lib**

Searches in order one or more library names to resolve any symbols not defined in the input `.obj` files. You may use wild card references as you can with the `obj` entry. The libraries are searched in the order specified here. You may use the same separators as you used for the `obj` portion apply. You may use the "+" continuation character that comes before the end-of-line or before the comment (#) string to extend past the end of the line.

The `lib` entry may be either a single file name or a pathname (with trailing "\") to a directory containing the libraries.

OPTLINK searches library files in the order that they are presented to it: that is, those named on the command line, and then those referenced by embedded commands in object modules.

To locate each library file for which no path is specified, OPTLINK first searches the current directory followed by any paths specified in the `LIB` variable, and finally any paths named in the command line.

### **def**

The definition file name. Optionally supplied, this file contains program linking information. This file type and its contents are described later in this manual.

### **res**

One or more resource (`.RES`) file names. When supplied, OPTLINK performs the bind step of the Resource Compiler while linking. This eliminates the need to use a Resource Compiler to combine resources to the output file.

Also like the `.OBJ` entry, wild-card references may be used. The libraries will be searched in the order specified. The same separators used for the `obj` portion apply, and the `res` entry may also be extended

past the end of a line by means of the continuance character (+) immediately before the end of line or comment (#) string.

**See Also**

[OPTLINK Option Switch Overview](#)

## Examples of Command Line Operation

The simplest method of using OPTLINK is to provide only the name of the .obj file to be used and accept the default suggestions for all other parameters:

```
C> LINK DUMP;  
OPTLINK Copyright (C) Symantec Inc. 1989-94  
All rights reserved.
```

OPTLINK's sign-on copyright message is displayed on the console as shown above.

If the .obj file does not exist in the current directory or in any of the paths supplied by the environment variables, OPTLINK sends an error message:

```
C>LINK DUMPER;  
OPTLINK Copyright (C) Symantec Inc. 1989-94  
All rights reserved.  
DUMPER.OBJ  
Error 2: File Not Found DUMPER.OBJ
```

If you do not use a semicolon on a short command line, OPTLINK reverts to the interactive mode after all command line input is used, as the following examples indicate:

```
C>LINK DUMP,,MYMAP;  
OPTLINK Copyright (C) Symantec Inc. 1989-94  
All rights reserved.  
C>LINK DUMP,,MYMAP  
OPTLINK Copyright (C) Symantec Inc. 1989-94  
All rights reserved.  
Libraries and Paths: (.lib):  
Definition File: (.def):  
Resource Files: (.res)
```

## Examples of Interactive Operation

Use the Enter key at the Map File: prompt.

```
C> LINK
OPTLINK Copyright (C) Symantec Inc. 1989-94
All rights reserved.
OBJ Files: (.obj):curset
Output File: (curset.exe):
Map File: (.map):
Libraries and Paths: (.lib):
Definition File: (.def):
Resource Files: (.res)
```

Using the "+" and the "#" comment characters causes multiple responses at the OBJ Files: prompt:

```
C> LINK
OPTLINK Copyright (C) Symantec Inc. 1989-94
All rights reserved.
OBJ Files: (.obj):file1+ # this is the first object file
OBJ Files: (.obj):file2 +
OBJ Files: (.obj):file3 # this is the last
Output File: (file1.exe):myfile
Map File: (.map):
Libraries and Paths: (.lib):mylib
Definition File: (.def):
Resource Files: (.res)
```

If you prefer, use multiple file names on the same line:

```
C> LINK
OPTLINK Copyright (C) Symantec Inc. 1989-94
All rights reserved.
OBJ Files: (.obj):file1 file2 file3
Output File: (file1.exe):myfile
Map File: (.map):
Libraries and Paths: (.lib):mylib
Definition File: (.def):
Resource Files: (.res)
```

## Indirect File Operation

If your application requires more file names or option switches than the command line can hold, you can use indirect file operation to effectively eliminate any limit on the amount of information you can supply to OPTLINK. For most complicated applications, this is the best choice.

OPTLINK uses the following syntax for indirect file operation:

```
LINK @indirfile
```

After entering the "@" character, LINK parameters are:

### **indirfile**

The name of an indirect response file. During interactive operation at each prompt, the indirect response file has a number of text lines.

## Example of Indirect File Operation

If the file `lnklst` contains the following lines:

```
DUMP  
NEWEXE  
MYMAP;
```

then the following command lines are equivalent except that line length in an indirect file is unlimited, while DOS limits the command line to a maximum of 125 characters by DOS itself.

```
LINK @LNKLST  
LINK DUMP,NEWEXE,MYMAP;
```

It is legal to use combinations of indirect file operation and either command line or interactive operation. OPTLINK accepts more than one indirect file specification. There are only two constraints: the indirect files that are combined must result in a valid sequence of responses and cannot be nested more than 10 deep.

**Note:** It is possible for a valid file name to actually begin with the "@" character. To keep such a file from being erroneously recognized by OPTLINK as an indirect response file, enclose the entire name of the file in quotation marks.

## Command Formats and File Names

All three methods of providing command input use exactly the same format for input information.

### File name usage

OPTLINK uses the same file-naming conventions as those supplied by the host operating system.

If a file extension is supplied, it must be separated from the name by a period. If none is supplied, OPTLINK assumes or creates a default value as described below.

The file name may be preceded by a pathname, which may be either absolute or relative and may include a drive specifier. If no path is specified, OPTLINK searches only the current working directory for the file or creates it in the current working directory. If a drive is specified, but no path, only the current directory for the specified drive is used.

### Default file extensions

If the file extension is omitted from a file name, OPTLINK assumes or creates a default extension for the file that is determined by the command with which the file name is associated. For instance, the default extensions for the file names supplied with the `OBJ Files:` prompt is `.obj` and for the `Map File:` prompt is `.map`. The defaults follow and are described later for each command that assumes or creates one. If you must refer to a file that has no extension and do not want a default to be used, you should add the `."` separator after the file name to tell OPTLINK not to supply an extension.

### File name extensions

Extension	Used for
.def	Definition file describing desired output
.exe	Executable files created by OPTLINK
.lib	Library files
.map	Listing of addresses generated by OPTLINK
.obj	Object files (in OMF format)
.cfg	Configuration file for default OPTLINK settings
.res	Resource files
none	Indirect command file used by OPTLINK

### Special file names

Since OPTLINK uses standard DOS functions for keyboard input and screen output, you may use device names in place of file names and you may redirect input and output. The device names are:

### Device names

Name	Device
AUX	refers to an auxiliary device (usually the same as <code>COM1</code> )
CON	refers to the console (keyboard input or display output)
PRN	refers to the printer (usually the same as <code>LPT1</code> )
NUL	a null (nonexistent) file; if given as an input name, returns end-of-file; if given as an output name, no output is generated

To redirect OPTLINK input and output from and to files, use the standard redirection keys. For example, the following command redirects OPTLINK output (including all prompts) to file `link.log` instead of to the screen:

```
LINK UTILITY; > LINK.LOG
```



## Build an Import Library: IMPLIB

You use IMPLIB to create an import library. An import library lets you import a function implicitly when you link an application with a DLL. An import library consists of a symbol table that the linker uses to define the locations of the functions in the DLL that are to be implicitly imported.

### IMPLIB Command Syntax

IMPLIB commands have the following format:

```
implib [/system] libfile.lib deffile.def
```

`libfile.lib` is the name of the import library file you want to create. `deffile.def` is the name of the DLL's module definition file.

You use the `/system` switch to create import libraries from Windows NT system DLLs (for example, `kernel32.dll`). Note that this switch is not available via the IDE.

IMPLIB creates an import library for the DLL whose name matches `deffile`. When you link the application, list the import library along with the other libraries it uses.

## Analyze Library and Object Files: LIBUNRES

Use LIBUNRES to check the names of symbols in library and object files. For example, you can run LIBUNRES with the `-p` option to obtain a list of the mangled names of functions in a DLL.

### LIBUNRES command syntax

LIBUNRES commands have the following format:

```
libunres [switches] file
```

file is the name of the library or object file you want to analyze. You can specify more than one file. The switches specify what you want to find out about the file(s):

### LIBUNRES switches

Switch	Description
--------	-------------

- |    |   |
|----|---|
| -c | Handle COMDAT records.  |
| -e | Print the names of all <code>extern</code> objects to stdout. |
| -i | Become case-insensitive.                                      |
| -l | Print symbol names to stdout.                                 |
| -p | Print the names of all public and common symbols.             |

By default, LIBUNRES lists the names of all `extern` symbols that are not public or common.

## Disassembling Object Files: OBJ2ASM

OBJ2ASM disassembles object files compiled with Symantec C++ or MASM. If you compile the file with the `-gl` option, OBJ2ASM mixes the source code and assembler instructions so that you can tell which statements produce which instructions. If you compile the file with the `-g` option, OBJ2ASM prints a listing of the file's debug records.

**Note:** You must edit the output of OBJ2ASM before you can use it with an assembler.

### OBJ2ASM command syntax

OBJ2ASM commands have the following format:

```
obj2asm [-l -o] [-soutfile] objectfile [.obj] [sourcefile]
```

The `objectfile` is the object file to disassemble. If you don't specify an extension, OBJ2ASM uses the extension `.obj`. The `sourcefile` is the file compiled to produce the object file. If you don't specify the `sourcefile`, OBJ2ASM looks at the module header information in the object file to determine the source file name.

The `-l` option tells OBJ2ASM to omit code labels in the assembly language file it generates. Use this option when you are comparing two object files; it helps highlight the differences between them. The `-o` option directs it to emit object code for each assembly language instruction. The `-x` option lets you show the code segment offset for each instruction. The `-s` option lets you direct OBJ2ASM's output to a file; the default is the standard output.

### Example

To see the code generated by compiling `test.cpp`, use the following commands:

```
sc -c -gl test
obj2asm test > test.asm
```

## **Unmangling Identifier Names: UNMANGLE**

The UNMANGLE utility unmangles identifier names in map and executable files. This utility is available in source code form in the distribution directory `\sc\src\ctools`.

### **UNMANGLE command syntax**

UNMANGLE commands have the following format:

```
UNMANGLE file
```

The file can be a text, linker map, or binary file. UNMANGLE writes its output to STDOUT by default.

## Using a version control system (VCS) with Visual Cafe



{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

With Visual Cafe Professional Edition, 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, allowing end users to install and work with multiple version control systems on a per-project basis.
- The *Microsoft Source Code Control (SCC) interface* is supported by Visual Cafe. Most version control software that uses the SCC interface will work with Visual Cafe.

### Installing version control systems for integration with Visual Cafe

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.

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.

Some products 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 with Visual Cafe:

- Intersolv PVCS Version Manager version 5.3 and 6.0
- Microsoft Visual SourceSafe (VSS) versions 4.0 and 5.0
- MKS Source Integrity version 7.3
- Rational ClearCase version 3.2
- StarBase Versions version 2.0

### Enabling version control for a project

You enable version control software on a per-project basis. If you have properly installed version control software on your computer, you will be able to select it in the Visual Cafe project options. Version control systems using the SCC interface are prefaced with the words “SCC Provider” in the project options.

The Visual Cafe 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.


 See Also

**Note** After creating a project, you should save it to the location you want it before choosing a version control system.

### Managing projects under a version control system 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. 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; for example, when you use a command in the Tools  Version Control menu. In addition, you can synchronize your Visual Cafe and VCS projects at any time by choosing Tools

 See Also Version Control



Refresh Status. 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 the following: when a VCS project is opened or closed as a result of opening or closing a Visual Cafe project, and file status inquiries.



**Note** If you want to move a Visual Cafe project and files in a project to another location, make sure no files are checked out before moving the files.

#### **Choosing version control menu items within Visual Cafe**

You can choose commands in the Tools



Version Control menu. The menu items that are visible in the menu depends on what your version control system supports.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can use version control system (VCS) software that integrates into the Visual Cafe environment. Visual Cafe supports version control in two ways: the Microsoft Source Code Control (SCC) interface or the Visual Cafe Version Control interface. If you have properly installed version control software that works with Visual Cafe on your computer, you will be able to select it in the Visual Cafe project options. You enable version control on a per-project basis.

See the documentation provided with your version control system for more information.

 See Also

**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). If you want to use the Visual Cafe Version Control interface (supported by vendors such as StarBase Versions and MKS Source Integrity), choose that version control install option, which might be listed as a Visual Cafe install option.

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

 See Also

**Note** After creating a new project, you should save the project to the location you want it before choosing a version control system.

2. Choose Project  Options

 See Also

Version Control Tab.

3. 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 have the line "SCC Provider: StarBase Versions;" whereas, if StarBase does not use the SCC interface, the line would show "StarBase Versions," but not "SCC Provider."

4. Click OK.

The change takes effect immediately.

If you are using the SCC interface, the Version Control Options dialog box appears, the provider may prompt you to create a new project or open an existing project, and then the Add to Version Control dialog box appears.

 See Also

[Details on setting version control options](#)

 See Also

[Details on adding files to version control](#)

 See Also

**Tip** If you are using the SCC interface, choose Tools  Version Control to view the version control menu items available to you. For example, you can set version control options and check files in and out.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are using the SCC interface, after enabling version control for a project, you can set version control options for the project.

#### To access the Version Control Options view

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

2. Choose Tools  Version Control

 See Also Options.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

#### To specify general version control options

1. In the Version Control Options dialog box, select the options you want:


##### Select...

Prompt to add files to VCS when new files are inserted

##### To specify this...

If selected, when you add one or more new files to a project, you are asked if you want to add them to the version control system. If deselected, new files are not added to version control. If you do not select this option, you can later add the files to the version control system by choosing Tools


 See Also Version Control

 Add. (Default: selected)

Prompt to remove files from VCS when files are removed

If selected, when you remove one or more files from a project, you are asked if you want to remove them from the version control system. If deselected, the files are not removed. If you do not select this option, you can later remove the files from the version control system by choosing Tools

 See Also Version Control

 Remove. (Default: selected)

Prompt to rename VCS files when files are renamed





If selected, when you rename a file in a project, you are asked if you want to rename it in the version control system. Alternatively, if your version control system does not support file renaming, you are 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. Realize that this can create complications by creating a discrepancy between your local files and the files in the version control system.

(Default: selected)

Prompt to check out files

If selected, when you try to rename or edit a read-only file in a project, you are asked if

when read-only files are edited	<p>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 do not select this option, you can later add the files to the version control system by choosing Tools  Version Control  Checkout. (Default: selected)</p>
Prompt to get files when opening the project	<p>If selected, when you open a project, you are 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 do not select this option, you can later add the files to the version control system by choosing Tools  Version Control  Get. (Default: selected)</p>
Prompt to check in files when closing the project	<p>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. (Default: selected)</p>
Enable background updates	<p>If selected, your version control system performs background status updates. If deselected, there are not background updates. How background status updates are implemented depends on your version control system. (Default: selected)</p>

2. Click OK.

The change takes effect immediately.

### To specify connection options

Currently, these options are not modifiable.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

If you are 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 are 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](#) for more information.

#### To add one or more files to version control

1. To access the Add to Version Control dialog box, activate the project you want to work with, then choose Tools

 See Also Version Control

 See Also Add 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.

 See Also


**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

2. 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 dialog boxes.
  - 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.
3. Click OK.

#### To remove one or more files from version control

1. To access the Remove from Version Control dialog box, activate the project you want to work with, then choose Tools

 See Also Version Control

 See Also Remove from Version Control.

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.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

2. 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 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.
3. Click OK.



{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are using the SCC interface, within Visual Cafe you can check files in and out of a version control system, and 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](#) for more information.

#### To check out one or more files in a project

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 Check Out.

The Check Out File(s) dialog box appears. It lists the files in your project that are not checked out by you or another user; if the project file (**vep**) is not checked out, it is also listed.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what 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 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.

#### To check in one or more files

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 Check In.

The Check Out File(s) dialog box appears. It lists the files in your project that are checked out by you; if the project file (**vep**) is checked out, it is also listed.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what 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 dialog boxes.
  - 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.
  - 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.

#### To undo one or more file checkouts

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 Undo Checkout.

The Undo Checkout dialog box appears. It lists the files in your project that are checked out by you; if you checked out the project file (**vep**), it is also listed.



**Note** What menu items you see in the Tools  Version Control menu depends on what 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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are 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](#) for more information.

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 See Also Get Latest Version.

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 also been added, it is also listed.

 See Also


**Note** What menu items you see in the Tools  Version Control menu depends on what 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.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe synchronizes with the version control system after any significant version control operation; for example, when you use a command in the Tools  Version Control menu. In addition, you can synchronize your Visual Cafe and VCS projects at any time by choosing Tools

 See Also Version Control

 See Also Refresh Status. 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, and file status inquiries.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are using the SCC interface, within Visual Cafe you can obtain the version control history of files .

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 Show History.

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 also been added, it is also listed.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what 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, CONTROL-click the files.
4. Click Show.

A dialog box that is implemented by your version control system appears.

5. Show the history of other files, if needed, then click Close when you are finished.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are 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.

1. Activate the project you want to work with.

2. Choose Tools  Version Control

 See Also Show Differences.

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 also been added, it is also listed.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

3. Select the file you want to view the differences of.
4. Click Show.

A dialog box that is implemented by your version control system appears.

5. Show the differences between other files, if needed, then click Close when you are finished.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

If you are using the SCC interface, within Visual Cafe you can show the properties of files in a version control system.

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 See Also VCS Properties, 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 also been added, it is also listed.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

3. Select the file you want to view the properties of.
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 are finished.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


If you are using the SCC interface, within Visual Cafe you can launch the main interface (front end) of the version control system.

1. Activate the project you want to work with.
2. Choose Tools  Version Control

 VCS, where VCS is the name of the version control system.

The front end of the version control application launches.

 See Also

**Note** What menu items you see in the Tools  Version Control menu depends on what your version control system supports.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

In Visual Cafe, you can set a default user name for the SCC interface. You might want to set it for a number of reasons:

- Not all version control systems prompt for it.
- The default Windows user name might be undefined (for example, under Windows 95 if you do not log in) or different from the version control user name you use.

You can specify the user name in the Visual Cafe environment as follows:

1. Choose Tools  Environment Options


 General tab.

2. Type the Version Control User Name.
3. Click Apply or OK to save the change.

The change takes effect immediately.

Obsolete

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

You can use version control system (VCS) software that integrates into the Visual Cafe environment. If you are using the SCC interface, VCS-specific menu items appear in the Tools  Version Control menu; for example, VCS Properties and VCS, where VCS is the name of the version control system, such as SourceSafe or PVCS. See your version control system documentation for more information on the dialog boxes that appear when you choose these menu items.

## NextGen features (including JFC Swing)

<http://java.sun.com/products/jfc/swingdoc-current/text.html>




{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also

New topics for this version are contained in the What's New book and Components Reference book of the Help Topics Contents.

(Choose Help  Help Topics

 Contents tab to locate the books.) In addition, the new Java macro utility is included; to get help on this feature, look in the Help menu of Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Form Designer is now scrollable. In Visual Cafe version 2.5 and earlier, the Form Designer window size was the size of your [form](#). Now the form is simply outlined in the Form Designer, and the Form Designer window size is not dependent on the size of your form. If the Form Designer window is smaller than your form, scrollbars appear in the Form Designer. Remember that the size of your form is displayed in the Property List.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


Visual Cafe provides [layouts](#) (also called layout managers) to help you organize components on a [form](#) or in a [panel](#). You can group components in panels on your form and use different layout managers to suit the components contained by the panel.

#### To arrange components in GridBagLayout by using the GridBag Constraints Editor

Use GridBagLayout to arrange components by size and space. Like GridLayout, GridBagLayout treats the form or panel as a grid of cells. Unlike GridLayout, however, a component can occupy more than one cell.

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 activated, choose Layout  Edit Constraints.

The GridBag Constraints Editor for that component appears. Only one GridBag Constraints Editor appears at a time.

4. Change Grid Bag Constraints 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.

 See Also

[Details on Grid Bag Constraints](#)

5. Use the GridBag Constraints Editor or Property List to set the Grid Bag Constraints properties to adjust each component's place in the layout. You can rearrange and add components as needed.

 See Also

**Tip** In the Property List, you can press F1 on a Grid Bag Constraints 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 resolutions. For example, you can resize the form when you run it from Visual Cafe.
7. To close the GridBag Constraints Editor, click OK.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

After you complete a Java program or bean in Visual Cafe, you are ready to deploy it. Visual Cafe has an extensive deployment capability that lets you deploy using a variety of archive options. Here are the general deployment steps:

1. Set up your environment for deployment by setting your environment options. This includes specifying paths to tools you want to use for deployment.
2. For each project you want to deploy, set up your project options to create the type of archive you want, such as a JAR, CAB, ZIP, or directory of files. You also choose what files are in the archive and where the archive is placed on the local computer or a remote computer.

 See Also

**IMPORTANT** Select Create Standalone Archive when you are ready to deploy your final Java program. This ensures that classes your Java program needs to run are included in the archive.

3. When you are ready to create an archive, choose Project  Deploy. Alternatively, you could use Project

 See Also

AutoJAR to create a JAR. Both commands compile the project and create the archive based on what you specified in project options.

### About setting Environment Options

In the Environment Options, Deployment tab, you can specify paths to tools you are using:

- If you are using JARs, you need to specify the path to Jar.exe; this program is provided with Visual Cafe.
- If you are using CABs, you need to specify the path to Cabarc.exe.
- If you are using security options for applets, you need to specify the path to Sun Microsystems tools; they should be in the same location as Jar.exe.
- If you are using Swing components, you need to specify the path to Swingall.jar; this JAR is provided with Visual Cafe.

Here are some URLs you might want to use to obtain the tools you need. (Remember that URLs and versions 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 \Java\Bin. If you want to use signed Jar files with Netscape or Internet Explorer browsers, you need a Java plug-in from JavaSoft.

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.

### To specify paths in Environment Options

1. Choose Tools  Environment Options

 See Also

Deployment tab.



2. Select the Include Tools For options you want, then specify the paths you need:

Field	Tool	Description
Sun Tools	Jar.exe and Javakey.exe  (Both are in the \Java\Bin directory.)	The first creates JAR files and the second lets you sign JAR files. Signing allows downloaded applets in JAR files to run with the same full rights as local applications. You can run signed Jar files in the Applet Viewer, HotJava, and a browser with a Java plug-in from JavaSoft.

Microsoft Tools	Cabarc.exe	Creates CABs.
Swing	Swingall.jar	Contains Swing components.
FTP		Type the Host, Port, Username, and Password.

3. Click Apply or OK to save the change.  
The change takes effect immediately.

### To set deployment Project Options

1. Select the project, then choose Project  Options  
 Deployment tab.
2. For each Deployment Category view, set the options you want.  
It is recommended that you start with General and work down the list of categories. Depending on your other selections, some options might be grayed. For example, when you choose to deploy to a local directory, you do not need to specify a host, user name, and password.  
The Project Options are divided into five Deployment Category views:
  - [Using the General deployment category](#)
  - [Using the Files deployment category](#)
  - [Using the Archiver deployment category](#)
  - [Using the Signer deployment category](#)
  - [Using the Advanced deployment category](#)
3. In the Project tab, set the release type you want, either Debug or Final.  
If you want to include debug information, select Debug. If you want an optimized program to deploy, select Final.
4. Click OK to save changes.  
The changes take effect immediately.

### To create the archive

- Choose Project  Deploy.



**IMPORTANT** Select Create Standalone Archive when you are ready to deploy your final Java program. This ensures that classes your Java program needs to run are included in the archive.

You could also use the AutoJAR command to create a JAR. See [Creating a JAR file by using AutoJAR](#) for more information.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

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 archive type.
-out <i>filename</i>	Specifies the location of the resulting deployment target.
-tempdir <i>directoryname</i>	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{[ <i>directory-name</i>   <i>jarname</i>   <i>zipname</i>  ;...}	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 <i>filename</i> , ...}	Specifies a list of files to be archived, and each file is separated by a space. For example, c:\projects\file.gif.
-entries <i>entryname</i> ,...}	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.
-debuglog <i>filename</i>	Specifies the file to log messages to.
-mstools <i>directoryname</i>	Specifies the directory where the CAB tool (CABARC.exe) is located.
-suntools <i>directoryname</i>	Specifies the directory where the JAR.exe andJAVAKEY.exe are located.
-JA “ <i>list of tool specific arguments</i> ”	The command line specific to the archiver tool being used. Type arguments (in quotes) that work with the specified archiver tool.
-JS “ <i>list of tool specificarguments</i> ”	The command line specific to the signing tool being used. Type arguments (in quotes) that work withthe specified signing tool.
@[ <i>responsefilename</i> ]	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.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

In the Deployment tab of your project options, if you chose a JAR or CAB archive type, you can specify archive options in the Archiver deployment category.

For a CAB, type the ID of the CAB file. See Microsoft documentation for more information.

For a JAR, if you want to include a manifest file, select Include Manifest, then specify the following options:

Option	Description
Supply Initial Manifest	Select this option if you want to supply a manifest file for Visual Cafe to use as a start. For example, you can specify Depends-On attributes. Then type the name and path of the manifest file in the Initial Manifest File field, or click the ... browse button to locate the file.
Save Generated Manifest	Select this option if you want to save the manifest file that Visual Cafe generates. For example, to run a Jar command-line tool, you may want to supply a manifest file. Then type the name and path of a manifest file in the Initial Manifest File field, or click the ... browse button to specify a file. You can specify an existing or nonexistent file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

In the Deployment tab of your project options, you can choose in the General deployment category:

Option	Description
Archive Type	Choose JAR, ZIP, Directory, or CAB. The CAB option is available only if you have included it in your deployment Environment Options.
Signer	If you select JAR, you can choose a signer. You must specify a signer in the deployment Environment Options. Currently only Sun tools are available.
Protocol	Choose FTP or File. You must specify FTP in your deployment Environment Options for it to be available; it is available by default. If you choose File Transfer Protocol (FTP), you can transfer files to a remote computer; you need to specify information in all of the Location fields. If you choose File, Visual Cafe deploys to the directory specified in the Directory field; this directory can be a local directory or a remote directory you have mapped to through the operating system.
Compress	Select this option to compress the archive. It is available for JAR and CAB files only. When you compress, download time is reduced, but run time speed might be compromised.
Overwrite existing output file	Select this option to overwrite the previous version of the archive or directory, if it is in the same location as your current project option settings. Note that if a file exists in this location

	and the file is not generated when you deploy again, the file is not deleted.
Add to Component Library	If you chose JAR, select this option to add a <a href="#">bean</a> to the Component Library. If there are no valid beans to add, you receive a warning message. Note that when you add a JAR file to the Component Library, all JavaBeans components in the JAR file are added (as specified in the <a href="#">manifest file</a> ). You cannot remove one component independently of the others in the JAR file.
Create standalone archive	<p>Select this option to deploy all files in your project, as well as additional class files needed for the program to run. These additional files must be on your classpath. See <a href="#">Setting internal VM environment options</a> and <a href="#">Specifying file search paths and the output directory</a> for more information. You can modify what files are included by changing information in the Files deployment category; for example, if you know users will have certain files on their computers.</p> <p>Deselect this option to include only the class files your project generates, plus any other files in your project, such as images, properties files, HTML files, and so on.</p> <p>For JARs, if you specify an initial manifest file with Depends-On attributes, these files are also added, whether you select this option or not.</p>
Copy resources to output directory	<p>Select this option to copy resources to the same location as the archive. If you choose an archive type other than directory, the files are placed outside of the archive, because the RelativeURL class cannot access files inside a JAR, ZIP, or CAB file.</p> <p>Resources are specified in your program by using the getResourceAsStream method or by the RelativeURL class. This way, you can move your program to another location and the resource files can still be found. Resource files are specified relative to the Java program.</p>
Directory	Type a directory or click the ... browse button to locate a directory. If you specified File in the Protocol field, specify a directory according to the local operating system rules. If you specified FTP protocol, use syntax that is valid on the remote server.
File	Type the name of the JAR, ZIP, or CAB file, either an existing file or a file you want to create. This option is not valid for a Directory archive type. No file extension is required.
Host	Type a host name or IP address, or specify localhost if it is running locally. The initial setting is specified in your deployment Environment Options.
Port	Type the port number. The standard default for FTP is 21. The initial setting is specified in your deployment Environment Options.
User	Type your user name for the remote FTP server. The initial setting is specified in your deployment Environment Options.

Password                      Type your password for this user name on the remote FTP server. Note that this password is not secure. The initial setting is specified in your deployment Environment Options.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In the Files deployment category of project options Deployment tab, if you chose the Create Standalone Archive in the General view, you can specify what classpath entries are included and not included in your archive by selecting or deselecting Include. Note that these zip files are shown but are not included in the archive:

- symclass.zip is not included because it contains classes for running in the Visual Cafe environment
- classes.zip is already included in the virtual machine
- swingall.jar is too large to include and currently the pieces needed by your Java program cannot be accurately determined

In the Files deployment category, click Manage Archive Entries to specify the files you want in the archive. Here are the options you can specify:

Option	Description
Included	Select this field to include the file in the archive. For example, you can choose to include source files in the archive; they are not included by default.
Java_Bean	Select this option to specify the file as a bean in the manifest file. It is valid for a JAR only. Note that you can specify an initial manifest file in the Archiver view, which affects these settings; if you are using an initial manifest file, you should specify it then change the Java_Bean settings here as needed.
Design_Time_Only	Select this option to mark the file as one you use during the design phase, so it is not included in a JAR. For example, you might want to select this option for a Java source file. Note that you can specify an initial manifest file in the Archiver view, which affects these settings; if you are using an initial manifest file, you should specify it then change the Design_Time_Only settings here as needed. This option is valid for a JAR only.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In the Deployment tab of your project options, if you chose Sun in the Signer field of the General deployment category, a Signing deployment category is available. Here you can specify a directive file to be passed to Javakey.exe to sign your JAR.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In the Advanced deployment category of your project options Deployment tab, you can specify debugging options to help technical support debug problems:

Option	Description
Generate Debug Log	Select this option if you want to generate a debug log. Then type the name and path of a manifest file in the Initial Manifest File field, or click the ... browse button to specify a file. You can specify an existing or nonexistent file.
Generate Response File	Select this option if you want to generate a response file, for example, for a command-line archiver. Then type the name and path of a manifest file in the Initial Manifest File field, or click the ... browse button to specify a file. You can specify an existing or nonexistent file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

From the Internal VM tab of the Environments Options, you can enable, disable, and perform a stack trace on the Just-In-Time compiler (JIT) of the internal virtual machine (VM) of Visual Cafe. In addition, you can set the classpath and Java heap and stack size parameters here, instead of having to edit the sc.ini file. You need to restart Visual Cafe for the changes to take effect.

The JIT in the VM converts Java bytecodes to native machine code; the machine code provides performance comparable to that of C or C++. Turning the JIT on 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.

Disabling the JIT makes Visual Cafe run slower. However, 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). It is also useful if the JIT has bugs so you need to turn the JIT off.

Turning on the stack trace means that the JIT is still enabled, but Visual Cafe runs a little slower. When an exception occurs, you get the same information as when the JIT is disabled, but without the file name and line number.

#### To open the Environment Options Internal VM view

- Choose Tools  Environment Options

 Internal VM tab.

#### To enable, disable, or perform a stack trace on the JIT

1. In the JIT field of 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 heap and stack size parameters

- In the Internal VM view, modify variables as needed. (Instead, you could modify the sc.ini file.)

Field	Description	sc.ini Variable
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= <i>n</i>
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= <i>n</i>
Maximum Java Stack	The maximum Java stack size for any thread. (Default: 400 KB,	VCAFE_IVM_oss= <i>n</i>

	Minimum: 1000 bytes)	
Maximum Native Stack	The maximum native stack size for any thread (the stack size is used by C code). (Default: 128 KB, Minimum: 1000 bytes)	VCAFE_IVM_ss= <i>n</i>

**Note** The value (*n*) is in bytes. Append a **k** to specify kilobytes or an **M** for megabytes.

#### To set the classpath

1. In the Internal VM view, modify the classpath list as needed:
  - To change the order that 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%.

2. Click Apply or OK to save the changes.

You need to restart Visual Cafe for the change to take effect.

#### Sample code for trying the different JIT settings

You can try this code with all three modes (you need to restart Visual Cafe when you switch modes). You will want to run it from a DOS prompt, so make sure that *VisualCafe\bin* and *VisualCafe\java\bin* are in your PATH.

```
// Test.java file
public class Test {
    public static void main(String[] argv) {
        new MyClass().foo();
    }
}

class MyClass {
    void foo() {
        bar();
    }

    void bar() {
        throw new RuntimeException("Hi there");
    }
}
```

## Localization

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}



Visual Cafe Professional Edition assists you in using [resource bundles](#) to localize your Java programs. It can create resource bundles, replace strings in your files, and help you easily add values to each resource bundle.

While you are developing a Java program, Visual Cafe can search for strings in your file, then replace them with a reference to a resource bundle. Visual Cafe also creates one or more resource bundles, which translators can use to specify words in different languages for different countries.

For example, this line:

```
textField1.setText("Hello");
```

Might be replaced with this line:

```
textField1.setText(resourceBundle.getString("textField1_text"));
```

And a resource bundle is added to the Visual Cafe project.

Visual Cafe can automatically localize all code in your program, or automatically localize auto-generated code and let you use a wizard to localize code you wrote yourself. Note that the default variable (key) names are different depending on which option you chose.

### Resource bundle types

Visual Cafe can automatically generate two types of resource bundles, a Java list type or a properties type. Here is an example of a Java list type of resource bundle called Applet1Bundle.java:

```
public class Applet1Bundle extends java.util.ListResourceBundle
{
    public Object[][] getContents()
    {
        return contents;
    }

    static final Object[][] contents =
    {
        {"textField1_text", "Hello"}
    };
}
```

Here is an example of a properties type of resource bundle called Applet1Bundle.properties:

```
textField1_text=Hello
```

A Java list type of resource bundle contains a class definition that gets compiled into the object that is being localized. As a result, it is faster to use once the Java program is compiled and used outside the Visual Cafe environment.

The properties type of resource bundle can be easier to replace during development, if you want to modify the file outside of the Visual Cafe environment.

You must use one type for all resource bundles in a project. If you switch types during development, Visual Cafe generates new resource bundles for that type (if they are not already present). You need to manually delete resource bundles of the other type in your project directory.

### Locales

Visual Cafe makes it easy to create resource bundles for different locales, which can include a language, country, and variant (which might be a dialect of a language). 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

The default is generally in English.

The translation for a string is searched in this 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

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

Visual Cafe Professional Edition assists you in using [resource bundles](#) to localize your Java programs. It can create resource bundles, replace strings in your files, and help you easily add values to each resource bundle. Here is a general process for localizing your code with Visual Cafe.

You can begin localizing code at any point in the development process. In addition, you can repeat the process as often as needed during development — Visual Cafe updates your resource bundles without destroying data you have entered previously. If you turn off localization for auto-generated code, Visual Cafe saves your resource bundles so they reappear when you turn on localization again.

1. Create a default resource bundle and replace strings.

You can use the project options or Localization Tool to localize auto-generated code and to create or update the default resource bundle. The Localization Tool also lets you localize code you wrote yourself.

 See Also

[Details on setting localization project options](#)

 See Also

[Details on using the Localization Tool](#)

2. Create resource bundles for locales.

 See Also

[Details on this step](#)

3. Add string values and edit resource bundles.

 See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

By setting your project options, you can specify that Visual Cafe Professional Edition automatically localize the code it generates automatically in the Init portion of the Java source (marked by INIT\_CONTROLS comments). You can also localize code by using the Localization Tool, which also helps you localize code you have written yourself.

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

2. Choose Project  Options

 See Also

Project tab.

3. Select the Localize Generated Code option to specify that you want Visual Cafe to automatically localize the code it generates. Or deselect it to not localize code.

 See Also

**Note** If you deselect this option after it was selected, Visual Cafe undoes localization for auto-generated code, but it saves your resource bundle in the project directory and it continues to appear in the Files view of the Project window. If you turn on localization again, this resource bundle is used, so you do not lose any information, such as string values.

4. Select the Generate Property Files option to specify that you want Visual Cafe to generate the property type of [resource bundle](#). Or deselect it to generate the Java list type of resource bundles.

If you switch types, the resource bundle is converted for you.

5. Click OK.

If you selected the option, Visual Cafe localizes code immediately and continues to localize it as you work on your files.

You can now create resource bundles for separate locales, as described in [Creating resource bundles for different locales](#).

You can also edit the resource bundle, as described in [Editing and specifying values in resource bundles](#).

 See Also

**Note** If you change the resource file type and want to remove resource files of the other type, you need to manually delete them from your project directory.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

With Visual Cafe Professional Edition, you can use the Localization Tool to localize code that you wrote and that was automatically generated by Visual Cafe in the Init portion of the Java source (marked by INIT\_CONTROLS comments).

1. In the Project window, select one or more files you want to localize. Or activate the Form Designer or Source window of a file.

2. Choose Tools  Localization

 See Also

Resource Bundle Strings.

3. In the Localization Options, select the type of [resource bundle](#) you want, and select All or Specify Strings. Then click Generate Resource Bundles.
  - Click All if you want Visual Cafe to localize all of your code, including code you wrote.
  - Click Specify Strings if you want Visual Cafe to localize the automatically generated code, plus let you choose what other strings you want to localize in code you have written.

 See Also

**Note** Clicking Generate Resource Bundles turns on automatic code localization in your project options. If you deselect the project option, auto-generated Init code is unlocalized. Code you wrote remains localized. The resource bundles are not deleted from your project directory or the Files view of the Project window, so they are used for code you wrote that was localized and if you select this project option again.

If you selected All, your code is localized.

If you selected Specify Strings, the Localization Tool appears. Visual Cafe searches for strings in the file you specified, then displays the first string and the line of code it was taken from.

If you switch types, the resource bundle is converted for you.

4. In the Localization Tool, check that the file name and resource bundle are correct. If not, click Browse to specify a different one.
5. For each string, optionally type a variable name for the string in the Resource Key field, a string in the Current String field, or both, then click Convert and Find Next. To skip a string, click Skip and Find Next.

When the entire file is localized, the Current String field displays “No Further Strings Found.”

6. To localize other files, click Browse to specify a Source File. If needed, also specify a resource bundle. Then localize the files as before.
7. When you are finished using the tool, click Close.

You can view the resource bundles in your project by looking at the Files view of the Project window, and double-clicking them to open them in the Source window.

You can now create resource bundles for separate locales, as described in [Creating resource bundles for different locales](#). You can also edit the resource bundle, as described in [Editing and specifying values in resource bundles](#).

 See Also

**Note** If you change the resource file type and want to remove resource files of the other type, you need to manually delete them from your project directory.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

With Visual Cafe Professional Edition, after you have created a default resource bundle by setting your project options or using the Localization Tool, you can add [resource bundles](#) for different locales.

#### To create a locale

The default locale is generally English. You can also specify other locales in separate resource bundles. Locales are generated for the entire project.

 See Also

**Note** The Localize Generated Code option must be set in your project options before you can create locales. You can set the option directly or by choosing Tools  Localization

 See Also

Resource Bundle Strings and localizing.

1. Choose Tools  Localization

 See Also

Add Locale.


2. In the Locale 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.
3. Choose a language.

The list is derived from the ISO 639 codes.

4. Optionally choose a country.

The list is derived from the ISO 3166 codes.

5. If you did not select the Show Only JDK-Supported Locales option and you chose a country, you can type a variant. For example, you could specify a dialect of US English, such as Eastern.
6. Click Add.

A resource bundle specific to that locale is added to your project. The locale now appears in the Tools  Localization

 See Also

Locales menu.

You can edit the resource bundle in Visual Cafe, as described in [Editing and specifying values in resource bundles](#).

#### To delete a locale

1. Choose Tools  Localization

 See Also

Delete Locale.

2. Select one or more locales, then click Delete.

The resource bundle is deleted from the Project window. It remains in your project directory.

#### To choose a locale to use in the Visual Cafe environment

- Choose Tools  Localization

 See Also

Locales

 See Also

locale.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


With Visual Cafe Professional Edition, you can edit a [resource bundle](#) with the Resource Bundle Editor, by typing in the Form Designer or Property List, or by editing the file directly.

#### To edit a resource bundle with the Resource Bundle Editor

 See Also

**Note** The Localize Generated Code option must be set in your project options before you can create locales.

1. In the Project window, select the resource bundle or the form that has a resource bundle.

2. Choose Tools  Localization

 See Also

Edit Resource Bundle.

The Resource Bundle Editor appears. The Resource Tag column shows the resource tags in the bundle. The Default column shows the default value.

3. Edit the resource bundle as needed.

The changes are reflected immediately in the source.

 See Also

**Tip** To rearrange columns, drag a column heading to another location.

#### To add string values to a resource bundle

You can add string values to a resource bundle from within Visual Cafe. Simply choose the locale and type in the Form Designer or Property List. If you have not added locales, the information is put in the default resource bundle.

1. In the Project window, select the file you want to specify a locale-specific information for. Or activate the Form Designer or Source window of the file.

2. Choose Tools  Localization

 See Also

Locales

 See Also


*locale*.

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.

#### To open a resource bundle in the Source window

You can manually edit resource bundles, such as adding other strings. You should not edit the auto-generated comment tags.

- In the Files view of the Project window, double-click the file, right-click the file and choose Open, or select the file and choose Object  Edit Source.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The operating system on your computer might use Asian or European language characters (high ASCII) that the Java compiler does not understand. If these native characters appear in your source code, you need to convert them to Unicode before compilation. Visual Cafe can handle this conversion for you, so you can view native characters in the Source Editor and still compile your code. This feature is useful for converting between the characters used by your system locale and that used by Java, but not for converting a language that does not match your system locale (for example, adding a Chinese resource bundle on a Japanese system); in this case, the feature is disabled.

To use this feature, you need to use the Visual Cafe localization features to create a resource bundle for the locale your operating system is using. Then you can enable automatic conversion between native and ASCII. For example, if a Japanese locale is the current system locale, and you type a native character in this language, Visual Cafe can display the native character in the Source Editor and automatically convert the character to a Unicode character (for example, "u3042") in the source file when you save the file. This lets you compile the file.

If you want to add native characters for a non-US locale that does not match your system locale (for example, for Korean characters if your system locale is Japanese), you need to convert these native characters by hand to ASCII to make sure the code compiles correctly. If you want to add to the project resource bundles that you wrote in native characters, you should convert them to Unicode before using them in the Visual Cafe environment.

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.

 See Also

[Details on this step](#)

2. Choose Tools  Environment Options

 See Also

General tab.

3. Select the conversion options you want.

You most likely want to choose both options together:

- Convert ASCII to native when opening file — When this option is selected, Unicode characters are converted to native characters so you can view these characters in the Source Editor. Otherwise, the Unicode value would appear.
- Convert native to ASCII when saving file — When this option is selected, before saving the file (including before compilation), the native characters are converted to ASCII. The Java compiler requires Unicode characters.

4. Click Apply or OK to save the change.

The change takes effect immediately.


Javadoc

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also

 See Also

Visual Cafe helps you to automatically produce [Javadoc](#) documentation for a project:

- To create Javadoc documentation for an entire project, select a project and choose Project  Produce Javadoc.
- To create documentation every time you compile, select the Always Produce Javadoc When Compiling option in your project options.
- To produce Javadoc for a file and view it, right-click a file in the project and choose View Javadoc.

#### To produce Javadoc documentation for a project

1. Activate the Project window of the project you want to work with.
2. Choose Project  Produce Javadoc.

Your Javadoc options determine what documentation is generated and where (described next).

 See Also

**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 specify Javadoc folders

1. Choose Project  Options

 See Also 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.

If you do not specify a full path, the directory is relative to the project directory.

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 documentation directory, which can take more time but may be what you need for deployment.

If the documentation directory and the system documentation directory are on the same hard disk but are set to be different directories, the generated links are relative. If they are on different hard disks, the links are absolute.

 See Also

**Warning** Do not make the Javadoc Documentation Directory the same as the default Javadoc System Documentation Directory.

4. Click OK.

The change takes effect the next time you produce Javadoc.

#### To set Javadoc options



Producing Javadoc documentation is usually very fast. However, producing the index and tree can take more time and memory. Also make sure you have enough disk space to store the documentation.

1. Choose Project  Options

 See Also 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 for files that have changed only; this reduces compilation time. In addition, the index, tree, and packages files are not generated.
Include Version	Include the Javadoc information specified with the <code>@version</code> tag.
Include Author	Include the Javadoc information specified with the <code>@author</code> tag.
Generate Index	<p>Generate AllNames.html, which 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 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 only used when you choose</p> <p>Project  Produce Javadoc.</p>
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.
Generate Tree	<p>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 Javadoc directories are not 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 only used when you choose</p> <p>Project  Produce Javadoc.</p>
Include Imported System Classes	Include imported system classes (java.*, sun.*, and com.*) in the index, tree, and package files that are generated. It generates 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.
Use Sun's Javadoc	Use the Sun Microsystems Javadoc compiler instead of <b>sj.exe -javadoc</b> . You can specify the name and path of the external Javadoc executable in the VM tab of your Environment Options.
Classes to Document	Choose the option 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 fields are included. Package includes all members without an access qualifier (public, private, or protected), as well as public and protected members.

4. Click OK.

The change takes effect the next time you produce Javadoc.

#### **To set SC.INI initialization options**

The system Java documentation directory is defined in SC.INI as follows:

```
API=%P%\..\JAVA\DOCS
```

The Javadoc generator built into the Symantec compiler uses this variable to determine the relative links that should be generated and to locate the images directory that contains GIF files used by the documentation.

The Javadoc Viewer search path is defined as follows:

```
JAVADOC_VIEWER_PATH=%P%\..\JAVA\DOCS;%P%\..\JFC\DOC\API;%P%\..\JAVA\DOCS\SYMANTEC
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

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, for example, by choosing Project

 See Also

Produce Javadoc.

#### To add or view Javadoc comments for a class, method, or data item

1. In the Files view of the Project window, right-click a file and choose Edit Javadoc Comments. Or activate the Source window or Class Browser Source pane for the file you want to Javadoc, then choose Source  Edit Javadoc Comments or right-click and choose Edit Javadoc Comments..

 See Also

#### 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.
  - You cannot add Javadoc comments to a read-only file. You receive an error message in the Messages window if you try to.
2. In the left pane, choose a member from the tree.

In the tree, click + to expand an item or - 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.

3. In the right pane, type your Javadoc comments in the fields (there might be a tab panel with additional fields).

You can adjust the size of the panels if needed by placing the cursor over the center divider then dragging. You can also resize the window by dragging a side.

 See Also

#### Note

- You can type HTML formatting tags in any field except for the See Also field. See a following table for examples.
- 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 project option settings determine what HTML files are generated when you choose Project  Produce Javadoc.

Field	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   or <P>.
Parameters	Choose a parameter from the drop-down list, then enter a description of this 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 <b>@param listener the listener to add</b> . This field appears when you select a method that has parameters.
Returns	Type a description of the return value 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 you want to appear in

	the <code>@author</code> tag; for example, David Bustin. Separate each author by a comma (,); this inserts separate <code>@author</code> tags for each author. This field appears when you select a class or interface.
Version	Type the information you want to appear in the <code>@version</code> tag; for example, 2.0a March 20, 1998. This field appears when you select a class or interface.
Since	Type the information you want to appear in the <code>@since</code> tag; for example, JDK 1.1.
Deprecated	Type the information you want to appear in the <code>@deprecated</code> tag; for example, "This method was replaced by the newconvert method."
See Also	Type the information you want to appear in the <code>@see</code> tag; for example, JScrollPane. For multiple See Also references, press ENTER at the end of a line to begin each new reference.
Exceptions	Choose an exception from the drop-down list, then enter a description of this exception that you want to appear in the <code>@exception</code> 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 <b><code>@exception myexception something didn't happen</code></b> . This field appears when you select a method.

Here are some HTML tags you might want to use. Consult HTML documentation for more information.

HTML tag	Description
<code>&lt;b&gt;text&lt;/b&gt;</code>	Place the <code>&lt;b&gt;</code> tag at the beginning of text you want to bold, and the <code>&lt;/b&gt;</code> tag at the end.
<code>&lt;i&gt;text&lt;/i&gt;</code>	Place the <code>&lt;i&gt;</code> tag at the beginning of text you want to italicize, and the <code>&lt;/i&gt;</code> tag at the end.
<code>&lt;c&gt;text&lt;/c&gt;</code>	Place the <code>&lt;c&gt;</code> tag at the beginning of text you want to display as sample code, and the <code>&lt;/c&gt;</code> tag at the end.
<code>&lt;p&gt;text</code>	Start a new paragraph with <code>&lt;p&gt;</code> and end it with <code>&lt;/p&gt;</code> .
<code>&lt;br&gt;</code>	Create a line break (a blank line).
<code>&lt;blockquote&gt;text&lt;/blockquote&gt;</code>	Indent text starting with <code>&lt;blockquote&gt;</code> and ending with <code>&lt;/blockquote&gt;</code> .

- To add the comments to the source file, click another member in the tree or close the editor by clicking the 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 are not 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

- Bring the editor to the front and the display is updated.



{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe provides a Javadoc Viewer that lets you quickly locate [Javadoc](#) documentation for files, packages, beans in the Component Library, classes, methods, and fields in the Source Editor, and more.

Anytime you choose the View Javadoc command, 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, or If Visual Cafe has already generated Javadoc for a Java file and you have modified the Java file since then, Visual Cafe asks you to save it first; you must save the file to get updated Javadoc.

#### To open the Javadoc Viewer

- Choose View  Javadoc Viewer.

#### To view the Javadoc for a file

- Right-click a file or object in 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 displayed in the Source window

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 did not generate Javadoc for private methods, the viewer displays the HTML file at the top of the file.

- Right-click in the body or Javadoc comment of a class or method, and choose View Javadoc.
- Right-click a field or Javadoc comment for a field, and choose View Javadoc.

#### 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 on 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 on the path. The Javadoc cannot be in a JAR.

#### 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. The viewer looks at the location set in the Visual Cafe sc.ini file with the JAVADOC\_VIEWER\_PATH variable (by default, *Visual Cafe\Java\Docs*) for the Java API documentation.

#### To go to the 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 your project options, is where the documentation for your project is placed. If Visual Cafe has never generated Javadoc for a Java file, or If Visual Cafe has already generated Javadoc for a Java file and you have modified the Java file since then, Visual Cafe asks you to save it first; you must save the file to get updated Javadoc.

#### To navigate back and forward between pages

- In the Javadoc Viewer, click the Back and Forward buttons.

#### To reload the 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 HTML file

- In the Javadoc Viewer, click the Print button.

#### To set SC.INI initialization options

The Javadoc Viewer search path is defined as follows:

```
JAVADOC_VIEWER_PATH=%@P%\..\JAVA\DOCS;%@P%\..\JFC\DOC\API;%@P%\..\JAVA\DOCS\SYMANTEC
```



## New Bean and Code Development Features

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can now develop and test [beans](#) all within the same project. You no longer have to place your bean in a [JAR](#) and drag it to the Component Library before you can use it in your projects.


After creating a bean in a project so the bean is at the top level of the Objects view of the Project window:

1. Add one or more test [forms](#) or other objects to the project.  
Note that you need to add any support files, including BeanInfo and property editors, to the project before using this feature.
2. Drag the bean onto the test object.  
The bean can be at any level; for example, within a panel. The bean becomes part of that form, and yet remains as a top-level object in the Project window.
3. Choose Project  Update Project Beans to update the bean contained by the object.  
You can now use the bean in that object as you would any bean. For example, for visual beans, you can view and modify it in the Form Designer and Property List. You can also execute a test applet or application.
4. Every time you make changes to the bean source and want to test the bean, choose Project  Update Project Beans.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



You can choose to update top-level beans (as shown in the Objects view of the Project window) when you open a project.

1. Activate the Project window of the project you want to work with.
2. Choose Project  Options.
3. In the Project Options dialog box, click the Project tab.
4. If you want to update top-level beans when you open a project, select Update Project Beans on Project Open. Otherwise, deselect it.
5. Click OK.

The change takes effect the next time you open a project.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Visual Cafe provides a wizard that helps you create a [servlet](#) using the Java Servlet API. This feature is not available in Visual Cafe Standard Edition.

It is recommended that you download the Java Servlet Development Kit from [www.javasoft.com](http://www.javasoft.com) for the most complete information about developing servlets.

1. Choose File  New Project.
2. Select Servlet and click OK.
3. If you receive the Introduction page, select the Don't Show This Page In the Future option 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.

 See Also

**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 is the 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 type, 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 type, 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 [Specifying execution settings for a servlet](#) 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.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```

 See Also

 See Also

To assist you in writing Java code, Visual Cafe provides a contextual helper in the Source Editor that 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. It also helps you enter [Javadoc](#) tags.

#### To activate or deactivate the Java Code Helper

1. Choose Tools  Environment Options

 Editing tab.

2. To activate the Java Code Helper, select Automatically Show Code Helper. Otherwise, deselect it.
3. Click Apply or OK to save the change.

#### To use the Java Code Helper

- As you type in the Source Editor, a drop-down list appears and you can choose items from the list.

#### To start the Java Code Helper at the insertion point

- Press Control-J or choose Source  Code Helper.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

The Visual Cafe Syntax Checker lets you identify problems that the compiler would find, but while you are writing code. This can save time and assist with troubleshooting.

#### To activate or deactivate the Syntax Checker

1. Choose Tools  Environment Options

 See Also 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 more complex errors), select Show Full Compiler Errors.


With the option selected, 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 the change.

#### To use the Syntax Checker

- As you type in the Source Editor, invalid items are highlighted in the error color and a yellow tooltip displays the error.

 See Also

**Tip** To view or set your error color, choose Tools  Environment Options

 See Also Display tab, then select Errors in the Color and Style list.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The 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 Class Wizard:

- Bean option
- Properties page
- Events page

Follow these steps:

1. To define a new class or interface with the Class Wizard, do one of the following:

- Choose Insert 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 in the Environment Options, you can press INSERT from the Class Browser or Hierarchy Editor.

If you select a class, that class becomes the default class to inherit from.

To edit a class or interface with the Class Wizard, do one of the following:

- Select a class or interface in the Classes pane of the Class Browser, then choose Classes Edit Class, or right-click and choose Edit Class.
- Select a class or interface in the Hierarchy Editor, then right-click and choose Edit Class.

2. On the first page of the wizard, specify the options you need:

Option	Description
Type	Select Class if you are creating a <a href="#">class</a> or Interface if you are creating an <a href="#">interface</a> .
Name	Type the name of the class or interface.
Source	Type the complete path to the Java file. Click the Edit button to browse. In the Edit Class File dialog box, you can specify the package and Java file name, and the file path is displayed for you.
Package	Choose a <a href="#">package</a> to add the class or interface to, or None to not add it to an existing package.
Extends	If you are defining a class, choose a class to extend from.
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 <a href="#">inner classes</a> only.
Final or Abstract	Select Final class or <a href="#">Abstract class</a> .
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 <a href="#">bean</a> are that it can be instantiated (it is not an abstract class or interface) and it has a class constructor method that takes zero parameters.

3. Click Finish if you are finished with the definition, or continue by clicking Next.

The next page of the wizard appears, where you can choose the interfaces to implement.

4. From the Available interfaces list, select the interfaces you want to implement and click the downward-pointing arrow.  
To move an interface from the lower list box to the upper one, select the interface and click the upward-pointing arrow.
5. Click Finish if you are finished with the definition, or continue by clicking Next.  
The next page of the wizard appears, where you can choose the [methods](#) to override. Required methods already appear in the Override these methods list box.
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 cannot move methods that are required.
7. Click Finish if you are finished with the definition, or continue by clicking Next.  
The next page of the wizard appears, where you can work with [properties](#).
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.
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 other objects get notified when the property changes value.
Constrained	Selecting this option generates code that takes care of creating notifications so 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 continue by clicking Next.  
The next page of the wizard appears, where you choose [events](#) to generate.
10. From the Available events list, select the events 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 previous pages of the wizard.
12. Click Finish when you are finished with the definition.  
The new class is inserted in the active project.
13. Complete the definition of the class or interface in the Source window or Class Browser.



[Details on using the Source window](#)



[Details on using the Class Browser](#)

## New Interaction Features

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

 See Also


One of the most powerful features of Visual Cafe is the ability to create [interactions](#) between [components](#). The Interaction Wizard lets you graphically build relationships between components, or between a component and itself (for example, double-clicking an item in a list box could remove the item from the list box and add it to another list box). These relationships specify the actions to take when an [event](#) is triggered on a component. For example, in a slide show, a button click could cause the next image to be displayed. Visual Cafe automatically generates the necessary code for the specified relationship. The components must be within the same project.

In this version of Visual Cafe, the Interaction Wizard has been redesigned to provide you with an even more powerful and intuitive interface. In the Form Designer, interactions are now represented by a small line and arrow, which helps you easily identify and edit interactions between components on your [form](#).

 See Also


**Warning** You should not use the Interaction Wizard to create interactions for Swing components placed in an AWT container. These interactions will not work.

#### To start connecting components within the Interaction Wizard

- Select a [component](#) in the Form Designer or Project window (Objects view), then choose Object  Add Interaction.
- Right-click an item in the Form Designer or Project window (Objects view) and choose Add Interaction.

Now complete the Interaction Wizard. You can also use the Interaction tool to visually connect components, as described next.

#### To visually create an interaction

- Click the Interaction tool icon , then drag a line from the trigger component to the action component within the same project. To connect a component to itself, simply double-click the component.

You can connect components displayed in the Project window (Objects view), Form Designer, Menu Designer, or a combination of these windows. If you want to create an interaction between different top-level objects, the interaction must be to a root-level object; for example, you could create an interaction between a menu item within a frame to a top-level dialog.

The action component is highlighted to help you identify which component you selected. If a component will not highlight, the interaction could be inappropriate.

After you release the mouse button, the Interaction Wizard appears. Complete the wizard to add the interaction.

 See Also

**Tip** Before you release the mouse button, you can press ESCAPE to cancel the interaction, or move the interaction line to another component.

#### To complete the Interaction Wizard

1. If you receive the Introduction page, select the Don't Show This Page In the Future option if you do not want to access this page again. Then click Next.

 See Also

**Tip** If you ever want to view the Introduction page again, simply click Back from the next page.

2. In the Interaction Wizard, verify that the trigger component name is correct.  
If the name is wrong, you need to start over.
3. In the Events field, select the [event](#) that activates the interaction.

The event list includes events inherited by other classes and also those defined by the component itself. You can display an alphabetical list of events by deselecting Group Events, or display an event list grouped by event type. The groups are determined by how events are grouped in event sets. In addition, the `boundPropertyChange` group is created by the wizard to group together bound properties.

4. Click Next.
5. Select Perform an Action, Call a Method, or Set a Property. Select the action component in the Available Objects list, then select the action, method, or property.

Your selections specify what you want to happen when the event occurs. The Actions list shows descriptions for connections defined in the BeanInfo file for the action component. The Methods list shows the methods in the action component and methods inherited by the action component. The Properties list is a subset of the Methods list and shows methods applying to setting component properties. When you select an item in the list, a short description appears at the bottom of the wizard.

5. Click Next.

6. If the item you specified takes parameters, specify the parameter then click Next. Repeat for each parameter.

The method is shown at the top of the wizard, with the parameter you are specifying in bold text.

If you select Get It From an Object, select a component in the Available Objects list, then select the parameter.

If you select Let Me Enter the Expression Myself, type the expression in the Type an Expression field. Optionally select Add Quotes. When you click Next, some error checking is done on the expression you entered; if the expression cannot be parsed, you are asked if you want to generate code anyway.

The wizard will convert types for you if needed.


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

In the Java source file, code is generated for the call handler, listener registration, and adapter/listener class.

In the source code, Visual Cafe performs five edits:

- 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.addTypeListener`) after the REGISTER\_LISTENERS tag. 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.
- Fifth, a call to the interaction method is generated in the event handler. If the call already exists, it is used.
- Sixth, an interaction method for the interaction specified in the wizard is generated. If the method already exist, its contents are replaced.

#### To modify what interactions are shown

- While the Form Designer is active, choose Layout  Interactions

 *command.*

Command	Description
Show Interactions From	Show interactions from the selected component (interactions where this component is the trigger component).
Show Interactions To	Show interactions to the selected component (interactions where this component is the action component).
Show Interactions To and From	Show all interactions involving the selected component.
Show All Interactions	Show all interaction lines and arrows on the Form Designer.
Hide Interactions	Do not show interaction lines and arrows in the Form Designer.

#### To identify an interaction

- When you select an interaction in the Form Designer, information is displayed in the Form Designer status bar.

#### To delete an interaction

Press DELETE or choose Edit  Delete for any of the following items:

- the interaction arrow on the Form Designer
- an object that triggers the interaction

The interaction method and the call to that method is removed from the source code. The event binding — the event handler method and the event listener class — will still remain, even if they no longer do anything.

If you delete an object that has an event binding or is involved in an interaction, the event handler method and any interactions it is involved in will be removed from the source code. The call to the handler from the event listener will also be removed. The listener itself remains. In other words, the source code is modified with the minimum amount necessary to remove all references to the deleted interaction or object.

For information on deleting pieces of an interaction in source code, see [Deleting an interaction by editing source code](#).

#### **To edit an existing interaction in the wizard**

- Double-click the interaction arrow in the Form Designer.
- Right-click an arrow and select Edit Interaction.

The interaction is opened in the wizard, which lets you view or change the interaction. Clicking Finish causes the interaction to be regenerated.

#### **To edit an existing interaction in source code**

You can edit interactions in the Source Editor, as described in [Changing an existing interaction by editing source code](#). Visual Cafe will note that you have changed the interaction. The next time you try to edit the interaction in the wizard, one of two things will happen:

- If Visual Cafe was able to convert your modified interaction to a form the wizard can display, the wizard will open and your changes to the source will be reflected there.
- If Visual Cafe was not able to convert the interaction back from source to editable form, either because of an error in the source code or because it cannot decipher the interaction in its new form, you receive a warning dialog box. If you choose to continue in the wizard, the interaction will most likely appear as it was before you made your source changes. Clicking Finish in the wizard will rewrite the interaction in the source, removing your changes.

The general rule for editing interactions in the source code is that if the wizard could have generated the interaction, Visual Cafe can convert it back to editable form. However, if you find Visual Cafe cannot convert it back, once you have edited an interaction in the source you must continue to edit it in the source to avoid losing your changes.

## JFC Swing

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

See Also

See Also

The Java Foundation Classes (JFC) provide powerful and flexible user-interface [components](#) called Swing. These mainly lightweight components are 100% pure Java and have a *pluggable look-and-feel*, which means the component appearance can change based on the computer platform or some other criteria, such as user preference, or have a platform-independent design. Swing is heralded as the future of UI development in Java.

Swing was developed by JavaSoft in cooperation with Netscape Communications. It is based on the Netscape Internet Foundation Classes (IFR) and has minimal dependence on native C code and the Abstract Windowing Toolkit (AWT). For JDK 1.1, the Swing components are located in the com.sun.java.swing package; for JDK 1.2, the Swing components are integrated into the root Java package. If you are using JDK 1.1 and applets, there is a disadvantage that Swing is not yet integrated into the JDK so it is extra overhead.

### Objects in each Swing component

Each Swing component is really made of a group of objects:

- the *component type*, such as the JButton object,
- the *component UI*, such as the ButtonUI object, and
- the *component model*, such as the ButtonModel object.

A Swing component type starts with a J to distinguish it from an AWT component. You often manipulate this object only, because it usually contains all of the methods you need to create and use the component. For example, the JButton class creates the UI and model objects when it creates a button.

The UI object defines the component appearance and handles UI functions such as installing, deinstalling, painting, updating, and preferred, minimum, and maximum sizes. It extends from the ComponentUI abstract class of the Swing package, so the UI object always has some basic methods you can use. This object implements the look and feel.

The model object stores the data and state of the component. For example, the ButtonModel object simply stores the state of a button, either on or off. 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](#) for information on specifying models from the Property List.

Swing components follow the *model-view-controller architecture*. The model (the component model object) stores data, the controller (the component type object) synchronizes data, and the view (the component UI object) displays data.

### JComponent features

Most 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, for example, a raised bevel line surrounding a button or an etched, titled line surrounding items in a panel. The *inset* is the space between the border and the component it contains. See [Specifying a border for a Swing component](#) for information on specifying borders from the Property List.

**Icons.** This feature lets you place any kind of graphics on a component, such as an image or animated image on a button. See [Specifying an icon for a Swing component](#) for information on specifying icons from the Property List.

**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.

**Keystroke handling.** You can associate command key sequences with a component so that when the component window is visible and 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 spacebar, or ALT+CHAR (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.

**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 and optimize how a component redraws itself. In the Property List, you can set the `DebugGraphicsOptions` property to initiate slow-motion rendering and to specify the option you want, including printing a message to the console, flashing graphics, and displaying 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. You can programmatically add this support.

*Actions.* An action is an object that implements the `Action` and `ActionListener` interfaces (often an `AbstractAction` subclass) and is a program function, such as cut, copy, or paste, that can optionally be associated with different components. Actions follow the JDK 1.1 *delegation event model* (see [Understanding component interactions](#) for more information on this model). Actions can be enabled or disabled, and have associated text strings and icons. There is an `Action` component in the Component Library; you can add actions to the Menu Designer (to a `JMenuItem` component) and toolbars (to a `JToolBar` component). For more information, see [Adding actions to a Swing menu or toolbar](#).

## Swing windows and applets

Swing has heavyweight `JFrame`, `JApplet`, `JDialog`, 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
- 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)

Windows have a `DefaultCloseOperation` property, which can cause the window to hide, be disposed of, or let nothing happen for a close event. There are also several `WindowListener` methods that notify components of `WindowEvents`, such as the window being activated, iconified, or closed.

Visual Cafe lets you view your form under different look-and-feels by choosing a menu item.



**Warning** You should not use the Interaction Wizard to create interactions for Swing components placed in an AWT container. These interactions will not work.

## Layout managers

Swing supports the AWT [layout](#) managers, and provides new ones, such as `BoxLayout` and, for the `JViewport` component, `JViewportLayout`. When you use the Visual Cafe visual environment to create a window or applet, the default layout manager is none.

*BoxLayout* is a layout manager that allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized.

Nesting multiple panels with different combinations of horizontal and vertical gives an effect similar to `GridBagLayout`, without the complexity. For example, you could have two panels arranged horizontally, each of which contain three components arranged vertically.

The Box container uses `BoxLayout` (unlike `JPanel`, which defaults to flow layout). You can nest multiple boxes and add components to them to get the arrangement you want.

*OverlayLayout* is a small-footprint layout manager that centers an object in its area. 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.

*ScrollPaneLayout* and *ViewportLayout* are the layout managers used by `JScrollPane` and `JViewport`, respectively.

## Lightweight and heavyweight components

The only heavyweight components used in Swing are:

- `JFrame`
- `JDialog`
- `JWindow`
- `JApplet`

- All AWT components (awt.\*), except those listed in the following list.

All other Swing components are lightweight. Swing's lightweight components are:

- awt.Component
- awt.Container
- JComponent
- All other Swing classes (swing.\*) and all packages under it, except for those noted in the previous list.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

A Swing [component](#) that extends from `JComponent` can have a *border*. A border is a visual container for a component, for example, a raised bevel line surrounding a button or an etched, titled line surrounding items in a panel. The *inset* is the space between the border and the component it contains.

In Visual Cafe, border objects are invisible components that appear in the Objects view of the Project window at the same level as the component that created it. You can share border objects within the same project across multiple components.

 See Also

**Note** You can manually delete unused border objects from your project. Visual Cafe does not remove them for you.

1. To display the Property List, choose View  Property List.
2. Select a component in the Form Designer, Project window, or from the Property List pull-down menu.
3. Find the Border property, then choose an item from the drop-down list.

The Visual Cafe Property List can display a Border property if it is appropriate for a certain 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 property will 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.

 See Also

**Note** Press ESCAPE to cancel an edit and return the property to its previous value.

5. Use the Property List to specify the properties of the border object.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

Many Swing [components](#) that extend from `JComponent` can have an *icon*. This feature lets you place any kind of graphics on a component, such as an image or animated image on a button. You can specify icons in the Icons properties in the Property List.

In Visual Cafe, icon objects are invisible components that appear in the Objects view of the Project window at the same level as the component that created it. You can share icon objects within the same project across multiple components.

 See Also

**Note** You can manually delete unused icon objects from your project. Visual Cafe does not remove them for you.

1. To display the Property List, choose View  Property List.
2. Select a component in the Form Designer, Project window, or from the Property List pull-down menu.
3. Find the Icon property, then choose an item from the drop-down list.

The Visual Cafe Property List can display an Icon property if it is appropriate for a certain component. Initially, the Icon property's drop-down menu contains types of borders; when you choose a icon type, a new icon object is added to your project and the Icon property will show the name of this new object. Every time you choose a icon type for the Icon property in the Property List, a new icon object is created and added to the drop-down list.

4. Press ENTER or click somewhere else to make the change.

 See Also

**Note** Press ESCAPE to cancel an edit and return the property to its previous value.

5. Use the Property List to specify the properties of the icon object.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Each Swing [component](#) is really made of a group of objects:


- the *component type*, such as the JButton object,
- the *component UI*, such as the ButtonUI object, and
- the *component model*, such as the ButtonModel object.

The model object stores the data and state of the component. For example, the ButtonModel object simply stores the state of a button, either on or off. For more complex objects, such as JTable, you may need to manipulate the model object to describe the data.

In Visual Cafe, model objects are invisible components that appear in the Objects view of the Project window at the same level as the component that created it. You can share model objects within the same project across multiple components.

 See Also

**Note** You can manually delete unused model objects from your project. Visual Cafe does not remove them for you.

1. To display the Property List, choose View  Property List.
2. Select a component in the Form Designer, Project window, or from the Property List pull-down menu.
3. Find the Model property, then choose an item from the drop-down list.

The Visual Cafe Property List can display a Model property if it is appropriate for a certain component. Initially, the Model property's drop-down menu contains types of models; when you choose a model type, a new model object is added to your project and the Model property will show the name of this new object. Every time you choose a model type for the Model property in the Property List, a new model object is created and added to the drop-down list.

4. Press ENTER or click somewhere else to make the change.

 See Also

**Note** Press ESCAPE to cancel an edit and return the property to its previous value.

5. Use the Property List to specify the properties of the model object.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe makes it easier for you to add actions to Swing menus and toolbars. There is an Action [component](#) in the Component Library; you can easily add actions to menus through the JMenuAction component and to toolbars through the JActionButton component.

An action is an object that implements the Action and ActionListener interfaces (often an AbstractAction subclass) and is a program function, such as cut, copy, or paste, that can optionally be associated with different components. Actions follow the JDK 1.1 *delegation event model* (see [Understanding component interactions](#) for more information on this model). Actions can be enabled or disabled, and have associated text strings and icons.

In general, you follow these steps to add an action to a Swing menu or toolbar. However you can vary the step order if it better suits your development process.

1. Add a JMenuBar and JMenu component, or a JToolBar component, to a form in your project.



[Details on creating a menu bar](#)

2. Add a JActionMenuItem component to the menu or a JActionButton component to the toolbar.
3. In the Actions group of the Component Library, locate the Action component and drag it to the top level of the Objects view of the Project window.
4. In the Objects view of the Project window, drag the Action object onto the form.

An instance of the action is created in the form. The Action object remains at the top level in the Project window.

5. In the Property List, set the Action property of the JActionMenuItem or JActionButton component to the Action object.
6. In the Property List, set the properties of the action instance and the Action object as needed.
7. Add code to the Action object so it performs the action, and set its properties in the Property List.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



Visual Cafe provides [layouts](#) (also called layout managers) to help you organize components on a [form](#) or in a [panel](#). You can group components in panels on your form and use different layout managers to suit the components contained by the panel.

BoxLayout lets you arrange components vertically (X-axis) or horizontally (Y-axis). Unlike FlowLayout, the components do not wrap to the next line, even when the form is resized. Components are arranged from left to right or from top to bottom in the order they were added to the container.

You can use nested Box containers to get an arrangement you want. Another possibility is to use a combination of panels, each with a vertical or horizontal BoxLayout; this arrangement can serve as a substitute for the more complex GridBagLayout.

#### **To choose a layout manager**

1. Open the Property List for a form or panel.
2. 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. Or, you might have to position components first.

3. Rearrange the components in the Form Designer, if needed.

#### **To arrange components in a Swing BoxLayout**

1. Choose BoxLayout for the Layout property of a form or panel.
2. Set the Axis property to X or Y.
3. If components are already on the form or panel, rearrange components, as needed.
4. Test your layout by running it at different form sizes and resolutions. For example, you can resize the form when you run it from Visual Cafe.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can toggle the display of Swing expert and read-only properties in the Property List. Whether a property is expert is defined by the programmer.

1. Choose Tools  Environment Options

 See Also Property List tab.

2. Select or deselect the Show Expert Properties and Show Read-Only Properties as needed.
3. Click Apply to make the change, or OK to make the change and close the Environment Options dialog box.

## Menu Designer

{ewc HLP25632,HLP256\_UPPER\_LEFT,can.bmp}

 See Also

 See Also


With Swing, you can add menu bars any valid container, such as to frames, applets, and dialogs. Swing menu items are auto-sizing and can contain both text and icons that can be aligned.

To create a menu bar, you can do the following:

- Add a JMenuBar component to a form.
- Add one or more JMenu components to the JMenuBar container.
- Add one or more JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem, JMenuItem, or JSeparator components to a JMenu container. You could also add other components, such as a JTree.
- Add one or more submenus to a menu item.
- Add command key equivalents to menu items.
- Add [interactions](#) between menu items and other components in your project. For example, specifying that a menu item open a dialog within the same project.
- Bind [custom code](#) to menu items.

The JFrame component in the JFC Application template already has a menu bar, which you can modify and enhance. The Swing Menu Designer makes it easier 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 JMenuBar component in the Project window or Form Designer.
- In the Project window, select the JMenuBar component, then choose Object  Edit JMenuBar or right-click and choose Edit JMenuBar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,cup.bmp}

 See Also

 See Also

With Swing, you can add menu bars anywhere they make sense, such as to frames, applets, and dialogs. The Swing Menu Designer makes it easier to create a menu bar by letting you edit a visual representation of the menu bar. Remember to add Swing components, which start with a J, instead of AWT components.

In general, to design your menu bar and menus, you follow these basic steps:

1. Add a menu bar to a form.

If you used the JFC Application project template, you already have a menu bar in the frame it created for you.

 See Also

[Details on this step](#)

2. Add menus to your menu bar.

 See Also

[Details on this step](#)

3. Add menu items to your menus.

 See Also

[Details on this step](#)

You can also add submenus.

 See Also

[Details on this step](#)

4. Edit the menu structure as needed.

 See Also

[Details on this step](#)

5. Associate command keys with menu items, as needed.

 See Also

[Details on this step](#)

6. Add [interactions](#) between menu items and other components in your project, as needed. For example, specifying that a menu item open a dialog within the same project.

 See Also

[Details on this step](#)

7. Bind [custom code](#) to menu items, as needed.

 See Also

[Details on this step](#)

 See Also

**Tip** You can save time by copying menu bars and menus from other projects. See [Copying a component in a project](#) for more information. You can also add commonly used menu bars and menus to the Component Library and Palette. See [Using the Component Library](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



When you right-click in the Menu Designer, Visual Cafe displays a pop-up menu with these commands. The commands change depending on the item you right-click. Also, any component that extends from JMenuitem or JSeparator and is in the Component Library can also appear in this menu.

#### **Insert JMenu**

Inserts a menu in the menu bar; you can then add menu items to the menu. Also lets you create a submenu for the selected menu.

#### **Insert JMenuitem**

Inserts a menu item into the selected menu that is in a menu bar.

#### **Insert JCheckboxMenuitem**

Inserts a checkbox menu item into the selected menu that is in a menu bar.

#### **Insert JRadioButtonMenuitem**

Inserts a radio button menu item into the selected menu that is in a menu bar.

#### **Insert JActionMenuitem**

Inserts an action menu item into the selected menu that is in a menu bar.

#### **Insert JSeparatorMenuitem**

Inserts a separator into the selected menu that is in a menu bar.

#### **Edit Source**

Opens the source file for the menu bar so you can bind code to menu items.

#### **Add Interaction**

Opens the Interaction Wizard or Editor so you can define an interactive behavior between the selected menu item and another component in the same project.

#### **Properties**

Displays the list of properties for the selected menu component.

#### **Cut/Copy/Paste**


These edit commands perform standard Windows functions.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also


You can add the JMenuBar component to a Swing container, such as a JFrame, JApplet, or dialog, in the following ways:

- Drag a JMenuBar component from the Component Library (Menus and Menu Items group) or Palette into the Project window or Form Designer.
- Use an Insert menu item:
  - While the Project window or Form Designer is active, choose Insert  Component.
  - Right-click the frame in the Project window and choose Insert Component.
- Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

To open the Menu Designer:

- Double-click a JMenuBar component in the Project window or Form Designer.
- In the Project window, select the JMenuBar component, then choose Object  Edit JMenuBar, or right-click and choose Edit JMenuBar.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

To add a Swing menu to a Swing menu bar, add a JMenu [component](#) to a JMenuBar component.

1. Add a JMenu component to a JMenuBar component in one of the following ways:
  - Use an Insert menu item:
    - Right-click the menu bar in the Menu Designer and choose Insert JMenu.
    - Right-click the menu bar in the Project window and choose Insert Component.
    - While the menu bar is selected in the Project window or the Menu Designer is the active window, choose Insert

 See Also

Component.

- Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

2. In the Property List, set the menu properties, including the menu name in the Name property.

 See Also

**Tip** To add more menus, you can select the Type Here menu in the list and type its properties in the Property List.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can add the following Swing menu item [components](#) to a JMenu component:



- JMenuItem
- JCheckboxMenuItem
- JRadioButtonMenuItem
- JActionMenuItem
- JSeparator

Visual Cafe provides a Swing Menu Designer that lets you visually design your menus with these Swing menu items. In addition, if you have added a component to the Component Library that has extended from JMenuItem or JSeparator, you can use the Menu Designer to add it to a menu as well. The component appears in the Menu Designer menus.

To create a menu item that can have submenus, see [Adding submenus](#).

 See Also

**Note** By writing code, you can add other components, such as a JTree component, to a menu.

1. Add a Swing menu item component to a JMenu component in one of the following ways:
  - Use an Insert menu item:
    - While a menu item is selected in the Menu Designer, right-click and choose Insert *J-menu-item*.
    - While a menu item is selected in the Menu Designer, choose Menu  Insert *J-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 Insert  Component.
  - Copy and paste it within the same project or between projects.

 See Also

[Details on this step](#)

2. In the Property List, set the menu item properties, including the menu item name in the Name property.

 See Also

**Tip** To add more menu items, you can select the Type Here menu item in the list and specify its properties in the Property List.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



To create a submenu, place a JMenu [component](#) within another JMenu component, then add Swing menu items to the nested JMenu component. Here is how you can add a submenu by using the Menu Designer:

1. In the Menu Designer, right-click a menu item and choose Insert JMenu.  
A menu that can have submenus appears.
2. While the new menu is selected, in the Property List set the menu properties, including the name of the menu in the Name field.
3. Click the submenu box, then add a menu item.



[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

You can quickly add command keys to menu items. Some components also have default keyboard key assignments defined in a standard look-and-feel.

1. Select a menu item in the Project window or Menu Designer.
2. In the Property List, specify the command keys in the Accelerator property.
3. Verify your command keys by running your Java program.

 See Also

[Details on this step](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also


 See Also

You can change the menu structure as needed.

**To move items in the menu structure**

- Move items in the Menu Designer or Project window to visually change it.

**To delete a menu or menu item**

- Select the component in the Menu Designer or Project window, then press DELETE or choose Edit  Delete.

 See Also

**Note** If you delete a component from a container, you must manually delete any [custom code](#), [interactions](#), and [event bindings](#) involving that component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also

Code can be [bound](#) to [components](#). Visual Cafe assists you in binding these [events](#) with the Add Event dialog box.

1. From the Project window (Objects view), Form Designer, or Menu Designer, double-click a component that is not at the top level. Or right-click a component and choose Bind Event.

The Add Event dialog box appears. The component name is at the top of the dialog box and in the list box are all possible events for this component. Events that are bold are already bound.

 See Also

**Warning** The dialog box does not appear if you double-clicked a component and if, when you last saw the dialog box, you deselected Show This Dialog Next Time. Instead, the Source window appears *and the default event is automatically bound to the object*. The dialog box also does not appear if there is not a default event defined for the component; in this case, no event is generated.

2. Select the event you want to bind to. If you select a bold event, you can edit it.

 See Also

**Tip** Select Group Events to group events by type in a tree view, or deselect it to show an alphabetical list of all possible events.

3. Click Add to add the event, click Open Source to view source code containing a declaration of the component, or click Cancel to cancel the operation.

If you clicked Add, the Source window appears at the [event handler](#). A comment shows you where you can add code.

If you clicked Open Source, the Source window appears and no event is generated.

OBSOLETE

 See Also

 See Also

Visual Cafe Professional Edition assists you in using resource bundles to localize your Java programs. It can automatically create resource bundles and replace strings in your files, either automatically as in code Visual Cafe generates or using a Localization Tool. For example, while you are creating a Java program, Visual Cafe can search for strings in your file, then replace them with a reference to a resource bundle as well as create one or more resource bundles that translators can use to specify words in different languages for different countries.

For example, this line:

```
textField1.setText("Hello");
```

Might be replaced with this line:

```
textField1.setText(resourceBundle.getString("textField1_text"));
```

And 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 wizard.

Visual Cafe also lets you specify resource bundles for locales, which can include a language, country, and variant (which might be a dialect of a language). The default is the locale configured for your computer. The translation for a word is searched in this 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

Visual Cafe can automatically generate two types of resource bundles, a Java type or a properties type. Here is 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"}
    };
}
```

Here is an example of a properties type of resource bundle called Applet1Bundle.properties:

```
label1_text=hello
```

A Java type of resource bundle contains a class definition that gets compiled into the object that is being localized. As a result, it is faster to use once the Java program is compiled and used outside the Visual Cafe environment. The properties type of resource bundle can be easier to replace during development, if you want to modify the file outside of the Visual Cafe environment. You can use one type for all resource bundles in a project. If you switch types during development, the new resource bundles are generated, if needed.

### To localize auto-generated code

Visual Cafe can automatically localize the code it generates automatically.

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

2. Choose Project  Options.

3. In the Project Options dialog box, click the Project tab.
4. Select the Localize Generated Code option to specify that you want Visual Cafe to automatically localize the code it generates. Or deselect it to not localize code.



**Note** If you deselect this option after it was selected, Visual Cafe undoes localization for auto-generated 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 to 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 option, Visual Cafe localizes code immediately and continues to localize it as you work on your files.

### To localize code with the Localization Tool

You can use the Localization Tool with code that you wrote or that was automatically generated by Visual Cafe.

1. In the Project window, select one or more files you want to localize. Or activate the Form Designer or Source window of a file.
2. Choose Project Localize Strings.
3. In the Localization Options, select Help Localize Non-Generated Strings then click OK.  
The Localization Tool appears. Visual Cafe searches for strings in the file you specified, then displays the first string and the line of code it was taken from.
4. In the Localization Tool, check that the file name and resource bundle are correct. If not, click Browse to specify a different one.
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 to generate the Java type of resource bundles.
6. For each string, optionally type a variable name for the string in the Resource Key field, then click Convert. To skip a string, click Find Next.  
When the entire file is localized, the Current String field displays “No Further Strings Found.” To go back to a string you skipped, you can use the Find Previous button.
7. To localize other files, click Browse to specify a Source File. If needed, also specify a resource bundle. Then localize the files as before.
8. When you are finished using the wizard, click Close.

You can view the resource bundles in your project by looking at the Files view of the Project window.

### To specify a locale

The default value is the locale configured for your computer. You can also specify other locales in separate resource bundles. You have to choose to automatically localize auto-generated code before you can add locales.

1. In the Project window, select the file you want to specify a locale for. Or activate the Form Designer or Source window of the file.
2. Choose Project Locale



Add Locale.

3. In the Locale 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 English, such as Eastern.
7. Click Add.

A resource bundle specific to that locale is added to your project. The locale now appears in the Project Locale menu.



8. Click Close after you have added all of the locales you want.

#### 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 of the file.
2. Choose Project Locale Delete Locale.
3. Select one or more locales, then click Delete.

#### To add 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. You need to add the locale before you can add information to it.

1. In the Project window, select the file you want to specify a locale-specific information for. Or activate the Form Designer or Source window of the file.
2. Choose Project Locale locale.
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.

#### To edit a resource bundle with the Resource Bundle Editor

You can edit a resource bundle from within Visual Cafe. You have to choose to automatically localize auto-generated code before you can edit a resource bundle with the editor, and you need to add a locale before you can add information to it.

1. In the Project window, select the resource bundle or the form that has a resource bundle.
2. Choose Object Edit Resource Bundle.  
The Resource Bundle Editor appears. The Resource Tag column shows the resource tags in the bundle. The Default column shows the default value.
3. Edit the resource bundle as needed.
  - To add another column for a locale, click Add Column then click the triangle icon and choose a locale.
  - To rearrange columns, drag a column heading to another location.
  - To remove a column, drag a column to the far right, then click Remove Column.
  - To add another resource tag, type in the bottom empty row.
4. Click Apply to apply changes.
5. Click Close to close the editor.


{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Bean Editor lets you quickly and efficiently develop JavaBeans components and classes. The editor gives you real-time feedback by using dynamic introspection. That means that changes you make in other Visual Cafe windows, such as the Source Editor and Property List, are shown in the Bean Editor, so it stays in sync with the code.

The Bean Editor lets you see class fields, inner classes, methods, properties, and events. It also shows superclasses and introspected bean information (BeanInfo). The editor provides you with the functionality to delete, expose, add, go to source, and override class items.

### To start the JavaBean Editor

- In the Files view of the Project window, select a file you want to edit, then choose View  Bean Editor.

The Bean Editor appears. It shows the classes contained within the file through a tree view. The starting point of the tree lists the class name and additional attributes of the class. The title bar displays the file being edited. If a BeanInfo file is not already present for the file, it is automatically created and added to your project.

### To navigate within the tree view

The tree view is organized with the classes contained within the source file being displayed at the top or root level. If additional items are contained within the classes, you will see a + displayed to the very left of the class name. Clicking the symbol expands the tree to show the titles of any additional items contained within the class. The headings you may see here are:

Classes	Any inner classes defined within the class.
Fields	Any variables defined in the classes scope.
Events	Any events that have been dynamically introspected.
Methods	Any methods defined within the class.
Properties	Any properties that have been dynamically introspected.

Under each of these headings you might find these additional headings:

Reflected	Items that have been recognized using dynamic reflection. Under this heading you may also see the Super heading. The Super heading represents information found in the superclass of this class.
Introspected	Items listed under this heading have been recognized using dynamic introspection of the BeanInfo file. Under this heading may be the heading Additional. The Additional heading represents items introspected from the superclass of the beaninfo class.

When an item other than a heading is selected, more detailed information is provided in the Property List for that item. Some of the items can be edited within the Property List.

### To delete an item

You can delete items contained within your classes source file or BeanInfo file. Items listed under the Super or Additional headings cannot be deleted because they reside in class files.

- Right-click an item, then choose Delete.
- Shift-click to select multiple items, move the cursor off the selected items, then right-click and choose Delete.

### To expose an item in the BeanInfo file

This operation works only on items listed under the Reflected or Additional headings, and the item must be an event, method, or property. Exposing events, methods, and properties causes a descriptor to be generated in the BeanInfo source file.

- Select an item, then right-click and choose Expose.

#### **To add an item**

- Select a class, then right-click and choose an Add menu item:

Event	Create a new event. See <a href="#">Using the Define Event dialog box</a> for more information.
Field	Create a new field. See <a href="#">Using the Define Field dialog box</a> for more information.
Interface	Create stub methods. See <a href="#">Using the Define Interface dialog box</a> for more information.
Method	Create a method. See <a href="#">Using the Define Method dialog box</a> for more information.
Property	Create a property. See <a href="#">Using the Define Property dialog box</a> for more information.

#### **To override a method**

The Bean Editor lets you easily generate a method stub in a source file.

- Select a method, then right-click and choose Override.

#### **To locate an item in a source file**

- Select an item, then right-click and choose Go To Source.
- Shift-click to select multiple items, move the cursor off the selected items, then right-click and choose Go To Source.

The Source window appears. The cursor is placed at the first line of source code for the selected item. For example, if you select a property, the cursor is placed at the accessor, manipulator, indexed accessor, or indexed manipulator method, depending on which appears first in the source file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



In the Bean Editor's Define Event dialog box, you can specify two types of events:

- *General Event*. Generate support for a general event, such as an action.
- *Specific Property Event*. Generate a specific event for a property.

#### To generate a general event

1. Select General Event.
2. Type an event name or choose an event from the list of commonly used events in the java.AWT.Event package.
3. Optionally select Unicast, which means there is a one-to-one relationship between the event source and the event listener (or at least an upper limit on the number of listeners). Otherwise, the event is multicast, which means there can be many listeners for one event. (Most events are multicast.)
4. Click OK.

Visual Cafe adds three items to the source code: an `addeventListener`, a `removeeventListener`, and a field of type `eventListener`, where *event* is the event name. `java.awt.AWTEventMulticaster` is the multicast support class used for adding and removing listeners and triggering events.

#### To generate a property event

1. Select Specific Property Event.
2. Type a property name or choose a property from the list of existing properties.
3. Select the options you want: Bound, Constrained, and Unique.

A *bound property* provides notifications to an applet, application, or bean based on changes in its value. A *constrained property* lets an applet, application, or bean perform validation on a new property value before accepting the modification.

If you select Unique, Visual Cafe creates the `PropertyChangeSupport` field, `VetoableChangeSupport` field, or both, and uses them within the appropriate methods. This helps you minimize event traffic. If you do not select Unique, Visual Cafe uses a common field for supporting the property events.

4. Click OK.

Visual Cafe creates up to seven separate items in the source code:

- `addpropbListener`
- `removepropbListener`
- `firepropbChange`
- `addpropcListener`
- `removepropcListener`
- `firepropcChange`
- `java.beans.PropertyChangeSupport` field
- `java.beans.VetoableChangeSupport` field

Where *propb* and *propc* represent the name of a property; *propb* relates to bound events and *propc* relates to constrained events.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Bean Editor's Define Field dialog box helps you generate a new field (variable) in your source code.

1. Type the field name.
2. Type or choose a field type.
3. Choose the access you want, one of Private, Protected, Public or default access (Package).
4. Select the other modifiers you want:

Modifier	If selected...
final	The field value will not change during runtime.
transient	The field is not part of the object's persistent state.
static	The field is a class variable.
volatile	The field values and calculations should not be optimized.

5. Click OK.

Visual Cafe generates the field in the source code.

```
{ewc HLP25632,HLP256_UPPER_LEFT,press.bmp}
```



The Bean Editor's Define Interface dialog box helps you generate stub methods in your source code (it creates a "skeleton" method where you can fill in the body).

1. Type or choose the interface or class you want to use as a template to generate the stub methods.
2. Click OK.

Visual Cafe generates stub methods of the class or interface in your source code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Bean Editor's Define Method dialog box helps you generate a new method in your source code.

1. Type the method name.
2. Type or choose a method return type.
3. Choose the access you want, one of Private, Protected, Public or default access (Package).
4. Select Abstract, Static, or None.

Modifier	If selected...
abstract	The method is specified with a name, parameter types, and return type, but no implementation.
static	The method is not associated with a class instance and it cannot be overridden. Called a <i>class method</i> .
none	The method is neither abstract or static.

5. Select the other modifiers you want:

Modifier	If selected...
final	The method cannot be overridden in a descendent.
native	The method is implemented in the platform's native code.
synchronized	The method can only be accessed by one thread.

6. Click OK.

Visual Cafe generates the method in the source code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}



The Bean Editor's Define Property dialog box helps you generate a new field in your source code.

1. Type the property name.
2. Type or choose a property return type.
3. Select the attributes you want.

Attribute	If selected...
accessor	Generate a read method for this property.
manipulator	Generate a write method for this property.
bound	Generate the code to fire property change notifications.
constrained	Generate the code to fire and handle a vetoable change notification.
indexed accessor	Generate an indexed read method for this property.
indexed manipulator	Generate an indexed write method for this property.
storage field	Generate a field to hold a value of this property.
unique event objects	Generate a specific PropertyChangeSupport object, VetoableChangeSupport object, or both.

4. Click OK.

Visual Cafe generates the property in the source code.

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}

 See Also

 See Also



The Interaction Editor lets you easily specify [interactions](#) for your Java programs, even very complex interactions. You can create interactions with the Visual Cafe Interaction Editor or Interaction Wizard. The components must be within the same project.

 See Also

**Note** Code is generated every time the interaction is completely specified and valid.

In the Form Designer, interactions are represented by a small arrow.


#### To start connecting components within the Interaction Editor

- Select a [component](#) in the Form Designer or Project window (Objects view), then choose Object  Add Interaction With Editor.
- Right-click an item in the Form Designer or Project window (Objects view) and choose Add Interaction With Editor.
- Choose View  Interaction Editor, then click the Event tool. (Tooltips identifies the different buttons.)

Complete the Interaction Editor as described in a following procedure, “To Specify an Interaction You Have Started.” You can also use the Interaction tool to visually connect components, as described next.

#### To visually create an interaction

The last tool you used, either the Interaction Editor or the Interaction Wizard, will appear when you visually connect components using the Interaction tool.

- Click the Interaction tool icon , then drag a line from the trigger component to the action component within the same project. To connect a component to itself, simply double-click the component.

You can connect components displayed in the Project window (Objects view), Form Designer, Menu Designer, or a combination of these windows.

The action component is highlighted to help you identify which component you selected. If a component will not highlight, the interaction could be inappropriate.

After you release the mouse button, the Interaction Editor appears.

 See Also

**Tip** Before you release the mouse button, you can press ESCAPE to cancel the interaction, or move the interaction line to another component.

#### To specify an interaction you have started

1. In the Interaction Editor, verify that the component names are correct.  
The trigger component should appear in the Event box; if you want a different component, choose a different one.  
The action component should appear in the green box; if you want a different component, you can choose it later.
2. Choose the type of action in one of these ways:
  - Right-click on the green box, then choose a Change Action menu item, including Call Method, Change Object, Set Property, Set Variable, or If statement.
  - Select the green box, then click a green button in the Interaction Editor toolbar. (Tooltips identifies the button.) Or drag the button to the green box.
  - Select the green box, then choose a Change Action menu item from the Interaction menu.

 See Also

**Tip** To change the action type, choose a different one. Or you can delete the action and start over.

3. Complete the green box :
  - For a method action type, choose a method. All methods for the action component class are listed.
  - For an object action type, choose an interaction. These interactions are specified in the BeanInfo file by the bean developer.
  - For a property action type, choose a property. Then specify a value (see the next step).
  - For a custom property action type, choose a property you want to set with a customizer. Then specify a value and click Set to open the custom property editor.

- For a variable action type, choose a variable you want to set. Then specify a value (see the next step).
- For an if statement, specify an action (as described previously) and a value (described in the next step).

If the action takes no parameters, the interaction is finished. If the action takes parameters, a burgundy data box appears for each parameter.

4. If you have a burgundy data box prompting for a value, specify the value type then choose the data items you want:
  - Right-click on the burgundy box, and choose a Change Value menu item, including Method Return, Object Data, Property, Variable, Text Expression, or Operator.
  - Select the burgundy box, then a burgundy button in the Interaction Editor toolbar. (Tooltips identifies the button.) Or drag a button to the burgundy box.
  - Select the burgundy box, then choose a Change Value menu item from the Interaction menu.

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.

In the source code, Visual Cafe minimally performs five edits:

- 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.addTypeListener*) after the REGISTER\_LISTENERS tag. 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.
- Fifth, the interaction specified in the wizard is generated in the event handler.

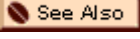
#### To edit an existing interaction


- Open the editor and edit the interaction.
- Double-click the interaction arrow in the Form Designer.

The last tool you used, either the Interaction Wizard or Editor, appears. However, if the interaction is too complex for the Interaction Wizard to handle, the Interaction Editor appears.

#### To modify what interactions are shown

- While the Interaction Editor is active, choose Interaction  Show Interactions From, Show Interactions To, Show Interactions To and From, Show All Interactions, or Hide Interactions.

- While the Form Designer is active, choose Layout  Interactions

 Show Interactions From, Show Interactions To, Show Interactions To and From, Show All Interactions, or Hide Interactions.

#### To delete an interaction

Press DELETE or choose Edit  Delete for any of the following items:

- the interaction arrow on the Form Designer
- the Event box in the Interaction Editor
- the action box (the outer blue box outline) in the Interaction Editor
- an object that triggers the interaction

{ewc HLP25632,HLP256\_UPPER\_LEFT,press.bmp}


 See Also


 See Also

The Swing Tab Order Editor lets you construct a tab order for a [form](#) that is different than the z-order (which is the order the [components](#) are listed in the Objects view of the Project window). This is useful if you want some components to be excluded from the tab order and when it is difficult to construct the tab order you want by changing the z-order, for example, if you have components grouped in various panels.

When constructing a tab order, remember that the virtual machine determines which component is the first component to get focus. This is often the component listed first in the z-order. For best results, you should have this component first in your tab order and not exclude this component from the tab order. When a component that is excluded has focus and a user presses the Tab key, the z-order is followed until a component is reached that is in your defined tab order; at this point, your defined tab order is followed. The last component in the tab order points to the first component in the tab order, so you can think of the tab order as a loop.

#### To access the Swing Tab Order Editor

- Select the form you want to edit, then choose Tools  Custom Tools

 See Also Swing Tab Order Editor.

To select a form, you can select a form in the Project window, or activate a Form Designer or Source window. All components contained by the form appear in the list. Note that you can press Tab to move through items in the list.

#### To change the order to the z-order

- Click Default Order.

The order of all components in the list changes to the z-order.

#### To change the order to top to bottom, then left to right, order

- Click Natural Order.

The order of included components in the list changes to natural order.

#### To exclude a component

- Select the component or shift-click to select multiple components, then click Omit.

The component name becomes grayed and moves to the bottom of the component list.

#### To include a component

- Select a grayed component or shift-click to select multiple grayed components, then click Include.

The component name is no longer grayed and moves into the list of components in the tab order.

#### To move components up or down in the tab order

- Select a component or shift-click to select multiple components, then click the up or down arrow or press the left or right arrow key on your keyboard.

## Component Date/History Tracking

1/16/97: Added dummy topics for Pro testing -- Date, Time, Timestamp.

2/1 - 3/3 - updated topics, added popups, removed jumps to API, remapped Event button

3/3/97 Sent for engineering review.

4/15/97 Remove DateMask, TimeMask, and DateTimeMask from index/browse string. Components were deleted from the release.

6/11/97 Modified RadioBox and RadioButton components, added DateTimeStamp Component

8/6/97 Update/check all components for 1.1

12/04/97 - Added JDBC components

4/27/98 -separated out dbAWARE/JDBC comps and moved them into another file.

8/98 - updated to new look and feel



{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 Swing Components

Visual Cafe helps you to design Java applets and applications, without writing source code by using a library of pre-written components.

In Java, the objects you use to create a visual interface are called components. You place your components into objects called containers, which define a window and determine how each component displays in its window.

For example, a Frame is the parent container used to build a visual interface for an application. To build a Frame in Java, you extend a Frame class and instantiate and configure that subclass object, and then instantiate, configure and add components, like Buttons or TextFields, into your Frame container. This is a complex process.

Visual Cafe simplifies this process. In Visual Cafe, you build an application project using the New Project window, drag component icons from the Palette to your Form Designer window, and configure the component using its Property List window. Visual Cafe does the rest for you.

**More information:** For more information about standard component classes, see the Java API Reference manual. API reference manuals are available from the Visual Cafe Help menu.

#### **Additional component information**

To access reference information about an individual component, perform a search in the Help system index or select the component on the Component Palette or in the Component Library and press F1.

#### **Also see**

[Understanding the Component class](#)

[Understanding the Container class](#)

## Components

Container objects are found in the Panel and Forms categories.

Standard	Additional	Additional (cont'd.)	Forms	Panels
<a href="#">Button</a>	<a href="#">ComboBox</a>	<a href="#">LabelButton</a>	<a href="#">AboutDialog</a>	<a href="#">BorderPanel</a>
<a href="#">Canvas</a>	<a href="#">DateMaskedField</a>	<a href="#">LabelHTMLLink</a>	<a href="#">Applet</a>	<a href="#">ImagePanel</a>
<a href="#">Checkbox</a>	<a href="#">DaySpinner</a>	<a href="#">Label3D</a>	<a href="#">AttentionDialog</a>	<a href="#">KeyPressManagerPanel</a>
<a href="#">Choice</a>	<a href="#">DirectionButton</a>	<a href="#">ListSpinner</a>	<a href="#">Dialog</a>	<a href="#">RadioButtonGroupPanel</a>
<a href="#">Horizontal Scrollbar</a>	<a href="#">FormattedTextField</a>	<a href="#">MonthSpinner</a>	<a href="#">Frame</a>	<a href="#">ScrollingPanel</a>
<a href="#">Label</a>	<a href="#">HorizontalSlider</a>	<a href="#">MultiList</a>	<a href="#">Open File Dialog</a>	<a href="#">SplitterPanel</a>
<a href="#">List</a>	<a href="#">ImageButton</a>	<a href="#">NumericSpinner</a>	<a href="#">PasswordDialog</a>	<a href="#">TabPanel</a>
<a href="#">Panel</a>	<a href="#">ImageHTMLLink</a>	<a href="#">RollOverButton</a>	<a href="#">ProgressDialog</a>	<a href="#">ToolBarPanel</a>
<a href="#">RadioButton</a>	<a href="#">ImageListBox</a>	<a href="#">ScrollingText</a>	<a href="#">Save File Dialog</a>	<a href="#">ToolBarSpacer</a>
<a href="#">TextArea</a>	<a href="#">InvisibleButton</a>	<a href="#">StateCheckBox</a>	<a href="#">Window</a>	
<a href="#">TextField</a>	<a href="#">InvisibleHTMLLink</a>	<a href="#">StatusBar</a>		
<a href="#">Vertical Scrollbar</a>		<a href="#">VerticalSlider</a>		
		<a href="#">WrappingLabel</a>		
Menus & Items	Multimedia	Shapes	Utility	Predefined TextFields
<a href="#">CheckboxMenuItem</a>	<a href="#">Animator</a>	<a href="#">Circle</a>	<a href="#">Calendar</a>	<a href="#">IntLongDistPhoneNumber</a>
<a href="#">Menu</a>	<a href="#">Emblaze20</a>	<a href="#">Ellipse</a>	<a href="#">ProgressBar</a>	<a href="#">LocalPhoneNumber</a>
<a href="#">MenuBar</a>	<a href="#">Firework</a>	<a href="#">HorizontalLine</a>	<a href="#">StatusScroller</a>	<a href="#">LongZipCode</a>
<a href="#">MenuItem</a>	<a href="#">ImageViewer</a>	<a href="#">Line</a>	<a href="#">Timer</a>	<a href="#">PostalCode</a>
	<a href="#">MovingAnimation</a>	<a href="#">Rect</a>	<a href="#">TreeView</a>	<a href="#">SocialInsuranceNumber</a>
	<a href="#">NervousText</a>	<a href="#">Square</a>	<a href="#">Wizard</a>	<a href="#">SocialSecurityNumber</a>
	<a href="#">Plasma</a>	<a href="#">VerticalLine</a>		<a href="#">USLongDistPhoneNumber</a>
	<a href="#">SlideShow</a>			<a href="#">ZipCode</a>
	<a href="#">SoundPlayer</a>			

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The Component class is the parent [class](#) from which all visual [components](#) and [containers](#) extend. This class provides a protocol defining [objects](#) that will possess size and position, can be rendered onto the screen, and can respond to [events](#).

The visual components present on the Visual Cafe palette are subclasses of class [Components](#). The [Container](#) class is also a subclass of Component. Specialized containers for applets, pop-up windows, and applications are subclasses of class Container. Both Component and Container are abstract classes and cannot be instantiated without first subclassing them.

Use a component in one of three ways:

- You can write an entirely new component by subclassing the Component class directly. Be sure to override all abstract members.
- You can instantiate one of the standard visual components (like Button or WrappingLabel) directly. Your code can handle that component's events by registering a listener with it. See [events](#) for more information.
- You can subclass one of the standard visual components (like Button or WrappingLabel). Your subclassed component will inherit all public Component data members and [methods](#), and can directly access the members it needs. Your code can handle that component's events by registering a listener with it (preferred), or by using the old event "inheritance" model. See [events](#) for more information.

By default, Visual Cafe provides an Applet component for creating [applets](#) and a [Frame component](#) for creating [applications](#).

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

A Container is a visual component that can hold other visual components. You build a Java GUI by placing components and containers into a parent Container extended from this class.

The Container class provides a calling protocol that defines objects that can display on the screen, show or hide themselves and their contents, receive or surrender focus, provide default responses to user [events](#) and hold objects of the type component.

Use a container in one of three ways:

- You can write an entirely new container by subclassing the Container class directly. Be sure to override all abstract members. The Window container typically provides this basic functionality.
- You can instantiate one of the standard Containers (like Frame or BorderPanel) directly. Your code can handle that container's events by registering a listener with it. See [events](#) for more information.
- You can subclass one of the standard Containers (like Frame or BorderPanel). Your subclassed container will inherit all public Container data members and [methods](#), and can directly access the members it needs. Your code can handle that container's events by registering a listener with it (preferred), or by using the old event "inheritance" model. See [events](#) for more information.

In Visual Cafe, applications are built on Frame containers and Applets are built on Applet containers. Double-clicking on one of these containers in the Project window opens a Form Designer window for visual editing of that container and its components. Other specialized containers are listed below:

<a href="#">Window</a>	A blank modal container that must be set into a Frame.
<a href="#">Frame</a>	A top-level window. Extends Window. It supports a title and menu bar, and can be minimized. It cannot be placed inside other containers.
<a href="#">Panel</a>	A basic organizational container. It can be placed inside other containers including other panels.
<a href="#">Applet</a>	A Panel that can be run by another program and supports multimedia features. Applet containers are parents to applet programs. Applets cannot be nested.

 See Also

#### Tips:

To build a GUI using Visual Cafe, create an [application](#) or an [applet](#) project using the Project window. Depending on what you selected, Visual Cafe automatically provides you with either a Frame container or an Applet container as the parent container for your project. Continue by dragging container and component icons from the Palette or Component Library to the Form Designer window as necessary.

To build a GUI in project source code, instantiate a Container object and all the Component objects you need. Call the container's [setLayout](#) method to add a [layout](#) manager, if desired. Then call that container's `Add` [method](#) to place Component objects into your Container object. Finally, call the container's [setVisible](#) method to make the container and all components visible.

## Standard components

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

Help is not currently available for this component.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



## Description

A labelled square box that the user toggles to turn on or off an option. To toggle, either click on the Checkbox's box or label, or press the spacebar when it has the focus.

Use a Checkbox to

- display a true/false value to the user and optionally allow it to be changed,
- display a possible range of options to the user and optionally allow one or more to be chosen.

If a Checkbox is a member of a group (by having a name is specified in the [Group property](#) field), it is a [RadioButton](#) and is used to select one of a possible range of options. See [RadioButton](#) for more information.

See also: [StateCheckBox](#)

See also: [RadioButton](#)

## Runtime modification

To add a [CheckBox](#) to a checkbox group at run time, call [setCheckboxGroup](#) to add the component to a particular group. If you are using a [RadioGroupButtonPanel](#), you can just add the component to the panel.

## Properties

To set the default state, either checked (true) or cleared (false), use the [State property](#).

Use the Interaction Wizard to have another component perform some action when the Checkbox value changes and generates an [ItemEvent](#) type event. Actions can be triggered when the Checkbox generates other standard events, too.

To group Checkbox components, use the [Group property](#) to specify a common group name. When the Group property is specified, the Checkbox behaves like a [RadioButton](#). Only one of the components in the Checkbox group can be "on" at a time. This allows the user to select on of a possible range of options. See [RadioButton](#) for more information.

## Coding the component

In project source code, use this syntax to create a new [CheckboxGroup](#):

```
group1 = new CheckboxGroup();
```

Where *group1* is the variable name used to identify the [Checkbox](#) group in code.

In project source code, use this syntax to create a new [Checkbox](#):

```
checkboxbox1 = new java.awt.Checkbox(String label, CheckboxGroup group, boolean state);
```

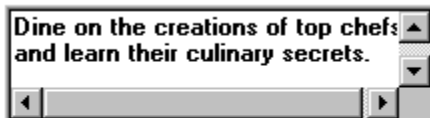
Part	Type	Description
checkboxbox1	Checkbox	The variable name used to identify the checkbox in code.
label	String	The text to display next to the checkbox.
group	CheckBoxGroup	The <a href="#">CheckboxGroup</a> , or null for no group.
state	boolean	The initial state of the check box: true is "checked"; false is "unchecked".

## Example

The following screen shot shows a single [CheckBox](#) component below a [TextArea](#) component.

**Description**

Dine on the creations of top chefs  
and learn their culinary secrets.

A rectangular text area with a thin border. It contains two lines of text: "Dine on the creations of top chefs" and "and learn their culinary secrets." To the right of the text is a small square icon with a downward-pointing arrow. Below the text is a horizontal scrollbar with a slider.

☐ Set as default

## Checkbox Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Group](#) (standard only)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Label](#)

[Name](#)

[State](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

A text box with an attached list box. Users can either type the desired value into the text box or select their choice using the list box.

Use a ComboBox to

- compactly display a list of items,
- manage a list of items longer than the display window size,
- change display fonts and colors and add edit and search functions.

When you include a ComboBox in a container, make sure that the attached list box doesn't cover up other components at run time.

### Properties

To add an item to a ComboBox list box, use [List Items property](#) in the Property List window. To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item.

To create a ComboBox that includes a scroll bar, set the [Show Horizontal Scrollbar](#) and/or [Show Vertical Scrollbar property](#) to true. You do not need to associate scroll bars manually. Vertical Scrollbars can be used to scroll through items when a ComboBox has a larger number of items in the list than can be displayed in the dropdown area.

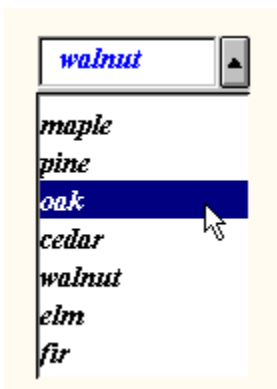
To use a different font in the ComboBox, set the [Font property](#).

A searchable ComboBox allows the user to enter text into the text field as long as it matches one of the existing list items. This allows the user to select from a non-editable list by typing instead of dropping-down the list and selecting with the mouse. To make the ComboBox searchable, set the [Searchable property](#) to true.

Use the [Text property](#) to set the initial value.

### Example

This ComboBox example has the ComboBox Font property set to Times Roman and the Foreground property set to Blue.



## ComboBox Properties

[Background](#)

[Bounds](#)

[Case-sensitive](#)

[Class](#)

[Cursor](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[List Down](#)

[List Items](#)

[Name](#)

[Searchable](#)

[Show Horizontal Scrollbar](#)

[Show Vertical Scrollbar](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A text field which has currency formatting logic applied to the user input.

Use a CurrencyTextField to:

- set the character used as a currency symbol.
- place a currency symbol before or after a value.
- limit the number of characters displayed after the decimal point.
- set the character used as a decimal point.

### Properties

To specify the required format for the text entry, use the [Leading Symbol property](#), [Symbol property](#), [DecimalPoint property](#), [DigitsAfterDecimal property](#), [Separator property](#) and the [SeparatorEnabled property](#).

To specify a text string to display in the field by default, use [Text property](#).

To make the default text editable, use the [Editable property](#). You do not need to write code or use the Interaction Wizard to connect the CurrencyTextField component with any other component to display text.

### Coding the component

To create a CurrencyTextField in project source code, use this syntax:

```
currencyTextField1 = new com.symantec.itools.awt.CurrencyTextField();
```

Part	Type	Description
CurrencyTextField1	CurrencyTextField	The variable name used to identify the currency text field in code.

## CurrencyTextField Properties

[Background](#)[Bounds](#)[Class](#)[Columns](#)[Cursor](#)[DecimalPoint](#)

[DigitsAfterDecimal](#)

[EchoChar](#)[Editable](#)[Enabled](#)[Font](#)[Foreground](#)[InheritBackground](#)[InheritFont](#)[InheritForeground](#)[NameSeparator](#)[Separator](#)[Enabled](#)

[Symbol](#)

[TextVisible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Extends MaskedTextField.

A text field which has edit masking logic applied to the user input for dates and times. The masking logic ensures that a match between the input character and the relevant filter for the cursor position is made. Based on the DateMask properties settings and the current locale setting, the component gets a date pattern ( like "d/MM/yy" ) and translates this pattern to a mask.

Use a DateMaskedField to:

- limit the type of text that can be entered in a text field.
- display text captured from the keyboard.
- edit a line of text.
- post an [event](#) based on text input from the keyboard.

### Properties

Use the [DateMask properties](#) to set the date or time type, the formatting style of the date or time, and whether the field should recognize literals (date and time separators) passed to it.



**Note:** Changes made to DateMask properties are not reflected at design-time.

### Coding the component

In project source code, use this syntax to create a new DateMaskedField:

```
dateMaskedField1 = new com.symantec.itools.awt.DateMaskedField();
```

Part	Type	Description
DateMaskedField1	DateMaskedField	The variable name used to identify the component

## DateMaskedField Properties

[Background](#)[Bounds](#)[Class](#)[Columns](#)[Cursor](#)[DateMask](#)[DateMask\\_properties](#)

[EchoChar](#)[Editable](#)[Enabled](#)[Font](#)[Foreground](#)[InheritBackground](#)[InheritFont](#)[InheritForeground](#)[Name](#)[Text](#)[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A text field which has text formatting logic applied to the user input.

Use a FormattedTextField to

- limit the type of text that can be entered in a text field,
- display text captured from the keyboard,
- edit a line of text,
- post an [event](#) based on text input from the keyboard.

To take action based on FormattedTextField events or text, use the Interaction Wizard.

If the text box already contains text, the user can select the default text and delete or edit it.

 See Also

**Note:** If the mask is set after the text is initialized, the text in the edit field vanishes. You need to call the [setText](#) method after [setMask](#).

FormattedTextField does not support logical AND, OR or XOR constructs and does not interactively prompt the user to retry input upon error. You must write project source code to accomplish these tasks.

### Properties

To specify the required format for the text entry, use the [Mask property](#).

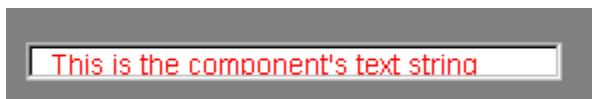
To specify a text string to display in the field by default, use [Text property](#).

To make the default text editable, use the [Editable property](#). You do not need to write code or use the Interaction Wizard to connect the FormattedTextField component with any other component to display text.

To hide the text that displays in the component, use the [Echo Char property](#) to specify a character to display in place of the characters in the string.

### Example

This screen shot shows a FormattedTextField with Foreground red, a 14 pt font, and a predefined Text string. The Field's parent container's background is gray.



## FormattedTextField Properties

[Background](#)

[Bounds](#)

[Class](#)

[Columns](#)

[Cursor](#)

[Echo Char](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Mask](#)

[Name](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

A platform-independent displayable image. An ImageViewer can be initialized from a URL pointing to a GIF or JPEG image, or through an Image object.

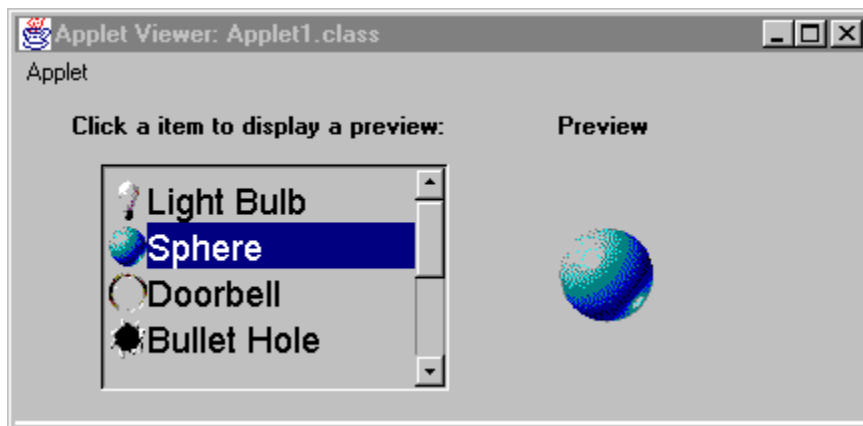
Instantiate an Image object through the [getImage](#) or [createImage](#) methods. The Image's [getSource](#) method returns an ImageProducer object that produces the image data. [getGraphics](#) returns a Graphics object. [getProperty](#) returns the value of a specified property.

### Properties

For non-database applications, use the [Image URL property](#) and [Image Style property](#) to identify the image and define the image's centering. These properties are not supported in the dbAWARE version of the ImageViewer.

### Example

This screen shot shows an [ImageListBox component](#) (left) and an ImageViewer component (right, under Preview).



## ImageViewer Properties

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Image Style](#)

[Image URL](#)

[Name](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

 See Also

### Description

A text string that is usually attached to another visual component (like a TextField or Button).

Use this component to:

- label a TextArea, or any other component that does not have a viewable label,
- display static, explanatory text,
- display a simple status area.

An application or applet can change the label text string, but a user cannot edit it.

### Properties

To specify the label text, type the text in the [Text property](#) .

### Coding the component

In code, use this syntax to create a new Label:

```
label1 = new java.awt.Label(label, alignment);
```

Part	Type	Description
label1	Label	The variable name you can use to identify the component in code.
label	String	The text to display in the label.
alignment	int	Positions text within the label bounding box. If no alignment is specified, the label is automatically left-aligned. Legal values are LEFT, RIGHT, and CENTER.

### Example

This example shows two standard Label components for two TextFields.



## Label Properties

[Alignment](#)

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

## Description

A box containing a scrollable and selectable list of items.

When a List item is selected or deselected, an ItemEvent is generated (see [item state changed](#)). When a user double-clicks on an item, the DbClicked action performed event is generated (see [action performed](#)). Use the Interaction Wizard to perform some action triggered by these (and other) events.

A List automatically displays with scroll bars when the display area is not large enough to display all the list items.

## Properties

To specify the initial items to use to populate the List, use the [Items property](#). To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item.

To allow your user to select either one item exclusively, or to select multiple items at once, set the [Multiple Mode property](#).

Use the Interaction Wizard to trigger actions based on user activity.

## Runtime modification

To:	Use:
Add items to a list	<a href="#">addItem(String item)</a>
Add items at a particular position	<a href="#">addItem(String item, int index)</a> . The item is placed before the index, for example, inserting an item at position 0 adds the new item to the very start of the list box. Use an index of -1 to place the item at the end of the list.
Remove all items from the list	<a href="#">removeAll()</a>
Delete an item	<a href="#">delItem(int position)</a> Position 0 is the first item in the list.
Get the current selected item	<a href="#">getSelectedIndex()</a> or <a href="#">getSelectedIndexes()</a> , for lists with multiple selection turned on.
Remove all selected items	A code sequence like the following: <pre>int[] selectedItems; selectedItems=list1.<a href="#">getSelectedIndexes()</a>; for (int index=selectedItems.size; index&gt;=0; index--)     list1.<a href="#">delItem(selectedItems[index])</a>;</pre>

## Coding the component

To create a new List component, use this syntax:

```
list1 = new java.awt.List(rows, multipleSelections);
```

This creates a new scrolling list initialized to display the specified number of rows. If the multipleSelections argument is true, then the user can select multiple items at a time from the list. If it is false, only one item at a time can be selected.

Part	Type	Description
list1	List	The variable name used to refer to the component in code.
rows	int	The number of items to display in the list. If this parameter is omitted, the list height is zero, and therefore not visible.

`multipleSelections`      `boolean`      Whether the list allows multiple selections.

The settings for *multipleSelections* are:

<b>Value</b>	<b>Description</b>
<code>true</code>	Multiple selections are allowed.
<code>false</code>	(Default) Only one item can be selected at a time.

## List Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Items](#)

[Multiple Mode](#)

[Name](#)

[Visible](#)

[Visible Rows](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also [Event256.bmp](#)

 See Also

**Description**

A text field which has Mask formatting logic applied to the user input.

Use a MaskedTextField to

- limit the type of text that can be entered in a text field
- display text captured from the keyboard
- edit a line of text
- post an [event](#) based on text input from the keyboard

**Properties**

To specify the required format for the text entry, use the [Mask property](#).

To specify a text string to display in the field by default, use [Text property](#).

To make the default text editable, use the [Editable property](#). You do not need to write code or use the Interaction Wizard to connect the MaskedTextField component with any other component to display text.

**Coding the component**

To create a MaskedTextField component, use this syntax:

```
maskedTextField1 = new com.symantec.itools.awt.MaskedTextField();
```

Part	Type	Description
MaskedTextField1	MaskedTextField	The variable name used to identify the component in code.

## MaskedTextField Properties

[Background](#)[Bounds](#)[Class](#)[Columns](#)[Cursor](#)[EchoChar](#)

[Editable](#)[Empty Means Null](#)

[Enabled](#)[Font](#)[Foreground](#)[Inherit](#)[Background](#)[Inherit](#)[Font](#)[Inherit](#)[Foreground](#)[Mask](#)

[Name](#)[Text](#)[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



#### Description

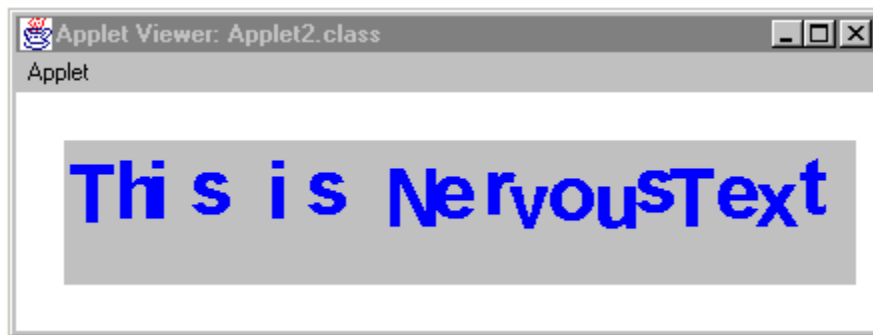
Creates animated text in which each letter moves independently of all other letters. This multimedia component is provided for novelty and programming convenience.

#### Properties

To add the text that appears within the component, use the [Text property](#) (for the standard version), or use the RelView Name and the Projection Name (for the dbAWARE version).

#### Example

This is a screen shot of NervousText at runtime.



## **NervousText Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Paused](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A round button used to select one choice from several options.

Use A RadioButton to:

- manage a selection-deselection user task,
- display a selection choice,
- post an event to another component,
- determine that only one choice in a set of choices is selected.

Typically, a RadioButton will be a member of a group. In a group of radio buttons, only one can be selected at a time. When the user selects one radio button, the other radio buttons in the group are automatically deselected.

When you add RadioButtons to your form they are automatically added to group, created as needed. Standard RadioButtons use a group of class `CheckboxGroup`.

Use the Interaction Wizard to trigger an action when a RadioButton is clicked.

### Properties

To set the default state, checked or cleared, of the RadioButton, use the [State property](#).

### Coding the standard component

In project source code, a radio button is identical to a check box, except it belongs to a `CheckboxGroup`.

To create a radio button, you must first have a `CheckboxGroup` to which you add the radio button. The code sample below shows the syntax required for creating `group1` a new `CheckboxGroup`.

```
group1 = new CheckboxGroup();
```

To create the radio button, create a check box and add it to a check box group. Use this syntax to create a radio button:

```
radioButton1 = new java.awt.Checkbox(label, Group1, state);
```

Part	Type	Description
radioButton1		Checkbox The variable name used in code to refer to the component.
label	String	The text to display next to the component.
group1	CheckboxGroup	The CheckboxGroup, or null for no group (resulting in a check box, not radio button).
state	boolean	The initial state: true is "checked"; false is "unchecked".

### Example

The following shows RadioButton and TextField components at runtime.



RadioGroupBox

## RadioButton Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Group](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Label](#)

[Name](#)

[State](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

## Description

An editable multi-line display window with two scroll bars. The display window can scroll through an amount of text that is too large to display at one time.

You cannot remove the scrollbars.

Use a TextArea to:

- display large sections of text,
- edit a block of text,
- scroll through a large block of text,
- post an event based on text input from the keyboard.

Use the Interaction Wizard to trigger one or more actions when the TextArea generates events.

The TextArea provides methods to add, delete and select text, but does not support text editing. You must write custom code to provide new editing functions.

 See Also

**Note:** For many purposes, you can use the Interaction Wizard to bind the KeyTyped event in order to receive notification whenever a user presses any key in the text area. This is often good enough to replace a true change notification.

## Properties

To make the TextArea text read-only, use the [Editable property](#).

## Coding the component

To select text, use TextArea's [select](#) and [selectAll](#) methods in your project source code.

To edit text, use TextArea's [append](#), [insert](#) and [replaceRange](#) methods.

To use the TextArea as a read-only text area, use the [setEditable](#) call.

By default, TextArea behavior accepts keyboard input and displays that input in its display pane. You can use the Interaction Wizard to change this default behavior.

When the TextArea receives or loses the focus, the [Focus Gained](#) or [Focus Lost](#) events are triggered.

To create a new TextArea component in project source code, use the following syntax:


```
text1 = new java.awt.TextArea(text, rows, cols);
```

Part	Type	Description
------	------	-------------

text1	TextArea	A variable which you use to reference the component in code.
text	String	The text to display in the text field.
cols	int	The width (in characters) that the text field can display and accept as input.
rows	int	The number of rows of text that the text area can display.

## Example

This screen shot shows a TextArea component that is populated with data from a database column.

 See Also

## TextArea Properties

[Background](#)

[Bounds](#)

[Class](#)

[Columns](#)

[Cursor](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Rows](#)

[Scrollbar Visibility](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A box in which your user can type text.

Use TextField to:

- display text captured from the keyboard,
- edit a line of text,
- post an event based on text input from the keyboard.

Use the Interaction Wizard to trigger one or more actions when the TextField generates events.

If the text box already contains text, the user can select the default text and delete or edit it.

The TextField provides methods to add, delete and select text, but does not support text editing. You must write custom code to provide new editing functions.

### Properties

Use the [Editable property](#) to make the default text editable. You do not need to write code or use the Interaction Wizard to connect the TextField component with any other component to display text.

 See Also

**Note:** For convenience, TextField supports several formatted components, which prompt for telephone numbers, zip codes and other formatted user entry strings. See the component listing under Predefined TextFields in the Component Library.

### Coding the component

To select text, use TextField's [select](#) and [selectAll](#) methods in your project source code.

To use the TextField as a read-only text area, use the [setEditable](#) call.

When the TextArea receives or loses the focus, the [Focus Gained](#) or [Focus Lost](#) events are triggered.

To create a new TextField component in project source code, use the following syntax:

```
text1 = new java.awt.TextField(text, cols);
```

Part	Type	Description
------	------	-------------

text1	TextField	A variable which you use to reference the component in code.
text	String	The text to display in the text field.
cols	int	The width (in characters) that the text field can display and accept as input.

### Example

The following shows RadioButton and TextField components at runtime.



**RadioGroupBox**

## **TextField Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Columns](#)

[Cursor](#)

[Echo Char](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A dialog box where you can display general information about your program.

You can insert an AboutDialog component into a Window container, a Frame container or an Applet project.

You can extend this dialog by adding components. This is done by dragging components and containers to the Form Designer window as necessary.

 See Also

### Tips:

- Use the Interaction Wizard to connect a component to the AboutDialog, so that your user can display the dialog box. Commonly, the AboutDialog is connected to the About menu item in a menu bar.

When you create a new project with the Basic Application template, a project is created automatically for you that contains a Frame, an AboutDialog, and QuitDialog (a custom built Dialog component). The frame contains a menu bar that has menu items preconnected to the dialog boxes.

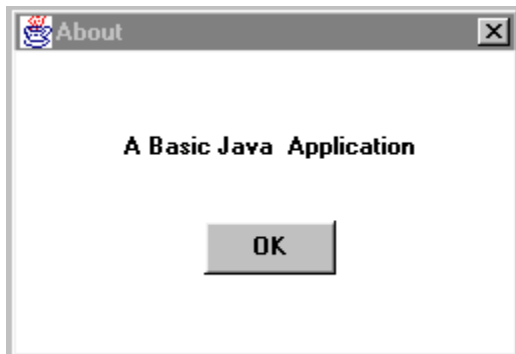
- The easiest way to draw an Interaction Wizard “connection” line from a menu item to a dialog box component is to draw the line in the Project window.

### Properties

To edit the text string that displays in the About dialog box header bar, use the [Title property](#). To change the About dialog title dynamically at runtime, use the Interaction Wizard to display the About dialog box and create the title based on the parent frame’s title or the frame’s warning string.

### Example

This screen shot is of the default AboutDialog that you can create with Visual Cafe.



## Common Dialog Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Resizable](#)

[Title](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

An animation that is created by displaying a series of images in sequence. This animation is static (there is no user interaction.) The sequence of images is drawn within the component using a [0,0] origin. Typically each image in the sequence is the same size.

To allow the user, or the program, to start and stop the animation, use the Interaction Wizard.

**See:** [MovingAnimation component](#) . The MovingAnimation component moves the image frames from left to right (or right to left) a predetermined amount each step.

### Properties

To specify the animation sequence, double-click on the [URL List property](#) to display the URL List dialog box. In the dialog box, enter a list of JPEG or GIF graphic files. List the files in the order you want them to be displayed.

To run an animated sequence one or more times set the [Loop Count property](#) .

Ignore the loop count and run an animated sequence forever by setting the [Repeat Mode property](#) to true.

To view your animation at design time, set the [Preview Component property](#) to true.

Ensuring the [Clear Frame property](#) is set to false prevents flickering caused by repainting the component background before drawing the component image.

### Example

There is a sample application that uses the Animator component in the release's Samples directory, under the subdirectory Animation. This example uses 37 GIF files to achieve the animation.

## **Animator Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Clear Frame](#)

[Cursor](#)

[Delay](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Loop Count](#)

[Name](#)

[Preview Component](#)

[Repeat Mode](#)

[URL List](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

An applet is the parent container that holds all components comprising an applet's GUI.

Use an Applet container to create an applet program.

Build an applet in Visual Cafe by choosing the Basic Applet icon displayed in the New Project window. Then to create the applet's GUI interface, drag the visual components you want from the Palette onto the Form Designer window attached to that applet project.

Applets support the following functionality:

- Applet programs are run by another program.
- Applet components include some built-in support for [threads](#). (For simple applets, you do not need to consider how your applet will handle threads.)
- Applets contain [methods](#) which manage sounds, pictures, and URL addresses.
- Applet programs can write to files only if the applets are located on the host system of the managing Web page. An applet can not write a file over a network.

Your applet can send and receive [events](#). Use the Interaction Wizard to post predefined events between your Applet container and the components it contains.

An applet program is a self-contained block of running Java code that a Web browser launches and runs from a Web page. Applets are called using the HTML `</Applet>` tag. To display an applet on a Web page, use an HTML file containing a reference to your applet. In HTML, a simple applet tag looks like this:

```
<APPLET CODE="Applet1.class" WIDTH=100 HEIGHT=100></APPLET>
```

 See Also

**Tip:** An applet does not have menus, but a [Frame component](#) that is invoked from an applet *can* have menus.

### Properties

You can change the layout manager using the [Layout property](#).

## Applet Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Extends Dialog

Creates a pop-up window that displays a message. Place an AttentionDialog container into a Window, Frame or Applet container.

Use AttentionDialog to display a program message requiring immediate attention. AttentionDialog windows are modal.

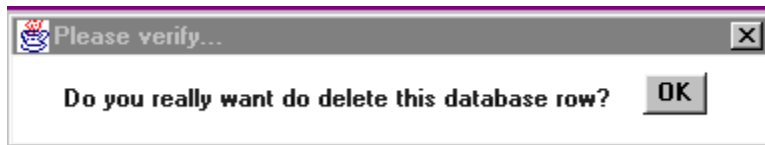
### Properties

To set the dialog box window title, use the [Title property](#) .

AttentionDialog provides layout managers, which you can choose by using the [Layout property](#).

### Example

This example is a slightly customized AttentionDialog. The interaction that was created with the Interaction Wizard to display this dialog specifies the text string for the label to the left of the OK button. The component was also resized for the label length.



{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Creates a container with a border, that holds visual components and other panels.

Use `BorderPanel` to display a decorative border and specifically to

- create a labelled subcontainer that organizes container space within an Applet, Frame or Dialog container. This simplifies your component layout task.
- hold other specialized Panel containers.

### Properties

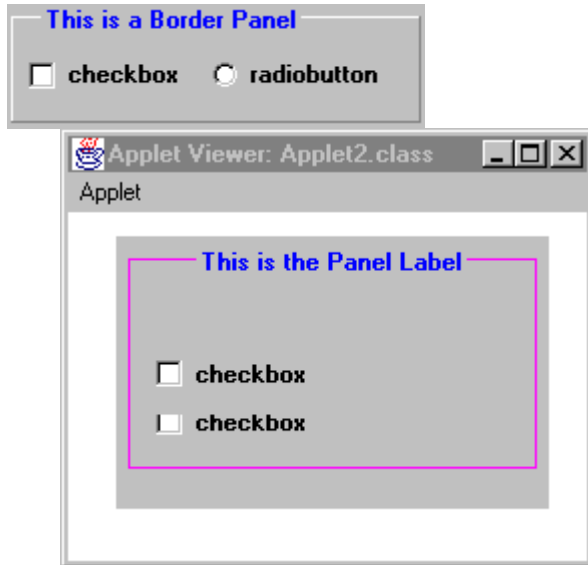
`BorderPanel` provides layout managers, which you can choose by using the [Layout property](#).

Use the [Border Color property](#) to set the color of the panel's border.

### Example

The first example is a `BorderPanel` with a light gray Background, blue Label Color, Label Alignment of `ALIGN_LEFT`, a `FlowLayout`, and `BEVEL_RAISED` Style.

The second screen shot is a panel with light gray Background, blue Label Color, Label Alignment `ALIGN_CENTER`, Padding Bottom 20, Style of `BEVEL_LINE`, and magenta Border Color.



## BorderPanel Properties

[Background](#)

[Border Color](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Inset Padding Bottom](#)

[Inset Padding Sides](#)

[Inset Padding Top](#)

[Label](#)

[Label Alignment](#)

[Label Color](#)

[Layout](#)

[Name](#)

[Padding Bottom](#)

[Padding Left](#)

[Padding Right](#)

[Padding Top](#)

[Style](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

Creates a simple button that initiates an action. A button usually displays a text label. The Button component is a standard Java awt component. A [LabelButton component](#) is a Symantec component that has additional features, such as a [Bevel Style property](#) for 3D appearance, and an [Alignment Style property](#) for flexible button label alignment.

Use a Button to:

- accept or yield focus. Buttons accept and yield focus automatically by default.
- respond to a user event. Buttons accept clicked events automatically by default.
- send an action event to another visual component. To send an action event to another component, use the Interaction Wizard. Optionally, you can register an event listener with the button in project source code.

A button appears outlined when it possesses input focus. This indicates that it can receive a user event. When the user presses the button it get redrawn so that it appears depressed. When the user then releases the mouse button while still over the button component, a click event is generated.

Visual Cafe also provides the specialized button components listed below. These buttons can generate repeated action event signals as long as they detect a MouseDown event.

[DirectionButton](#)

[InvisibleButton](#)

[ImageButton](#)

[LabelButton](#)

[RollOverButton](#)

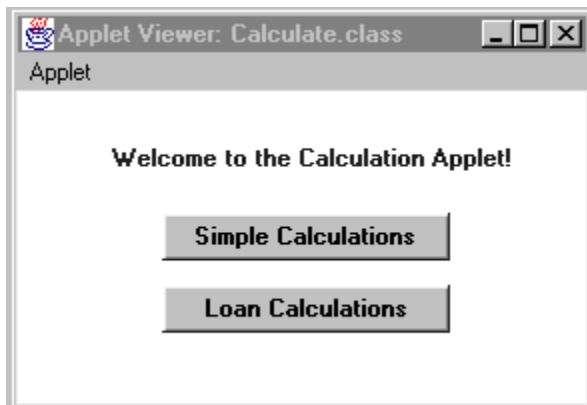
### Properties

To specify text for the label, use the [Label property](#) .

When a button is disabled, it is 'grayed out.' This indicates that it can not receive user input. Set the enable state using the [Enabled property](#) .

### Example

These are Button components at runtime.



## Button Properties

[Action Command](#)

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Label](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

A calendar that displays a month at a time.

Displays a calendar with text boxes. These text boxes have up and down arrows and contain a list of years ([NumericSpinner component](#)) and a list of the months in the year ([MonthSpinner component](#)). At run time, the calendar displays with the selected date text as grayed out.

The Calendar component sends an action event when the user clicks on a date or changes the date.

You can use the Interaction Wizard to reset Calendar properties and to enable/disable and show/hide the component.

### Properties

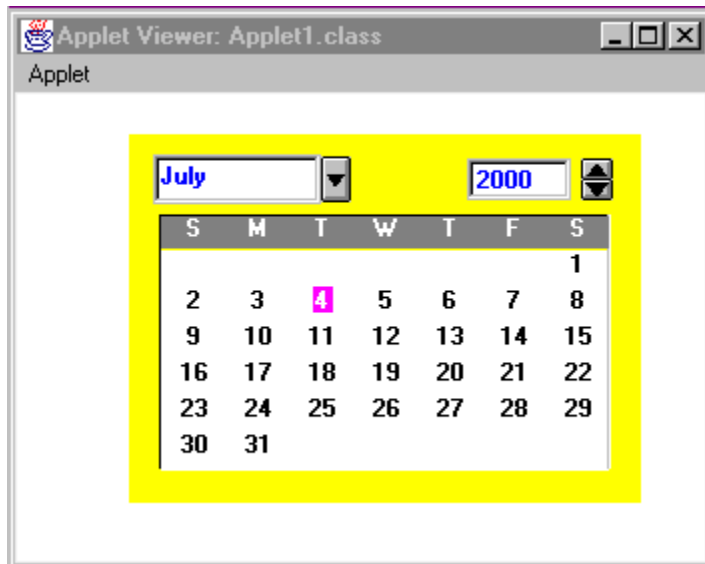
To specify a start date, use the [Date property](#) . The string accepts many date syntaxes; including the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT"

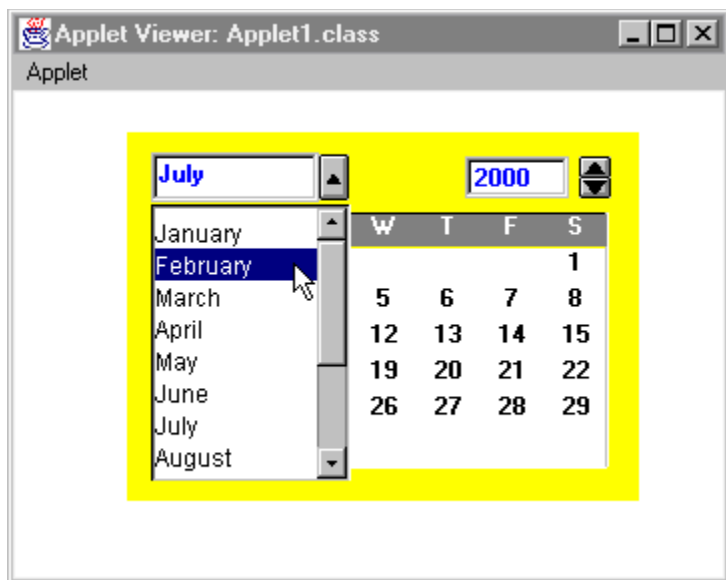
To set the color of the text in the month and year text field, use the [Foreground property](#) .

To set the color of the selected/highlighted date on the calendar, use the [Selection Color property](#) .

### Example

The first screen shot shows the Calendar component with Foreground property set to Blue, Selection Color property set to Magenta, and Background property set to Yellow. The second screen shot shows the dropdown list for month selection.





## Calendar Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Date](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Selection Color](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Canvas provides the visual component interface for a graphics object, so your GUI can size, position, show, hide, draw, and redraw that object, and respond to user events.

Visual Cafe provides many specialized Canvas components for programming convenience. These specialized components provide predefined Paint and Events methods and do not have to be subclassed.

Visual Cafe provides the following Canvas subclasses:

<a href="#">Animator</a>	<a href="#">Circle</a>	<a href="#">DirectionButton</a>
<a href="#">Ellipse</a>	<a href="#">Firework</a>	<a href="#">HorizontalLine</a>
<a href="#">HorizontalSlider</a>	<a href="#">ImageButton</a>	<a href="#">ImageHTMLLink</a>
<a href="#">ImageViewer</a>	<a href="#">InvisibleButton</a>	<a href="#">InvisibleHTMLLink</a>
<a href="#">Label3D</a>	<a href="#">LabelButton</a>	<a href="#">LabelHTMLLink</a>
<a href="#">MovingAnimation</a>	<a href="#">Nervous Text</a>	<a href="#">Plasma</a>
<a href="#">ProgressBar</a>	<a href="#">Rect</a>	<a href="#">ScrollingText</a>
<a href="#">Square</a>	<a href="#">StateCheckBox</a>	<a href="#">ToolBarSpacer</a>
<a href="#">VerticalLine</a>	<a href="#">VerticalSlider</a>	<a href="#">WrappingLabel</a>

You must override Canvas methods to receive and send user events, to draw corresponding objects on the Canvas component, and redraw the Canvas when it changes.

To use Canvas in your project, write this project code:

1. Subclass Canvas, and add the subclass to your container.
2. Override the Canvas [paint](#) method. Draw your component using the Graphics object parameter.
3. Write and register listeners for the desired events. This can be done by selecting your new Canvas subclass in the Project window and using the Interaction Wizard.

## Canvas Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a check box that can be included in a menu. Clicking the associated menu item toggles its state between checked (marked with a check mark) and unchecked.

Use `CheckboxMenuItem` whenever you include a menu item that you want to toggle between on and off.

Use the Interaction Wizard to trigger an action when the `CheckboxMenuItem` is checked or unchecked.

### Properties

Use the [Checkbox property](#) to set the initial menu state, either checked or unchecked.

The [Label property](#) sets the menu item text.

### Coding the component

Use the [setState](#) method to check or uncheck a menu item.

The Java language does not require you to create a variable to reference a menu item. Instead, you can refer to the menu item by its label.

In project source code, use this syntax to create a new `CheckboxMenuItem`:

```
menuItem1 = new java.awt.CheckboxMenuItem(label);
```

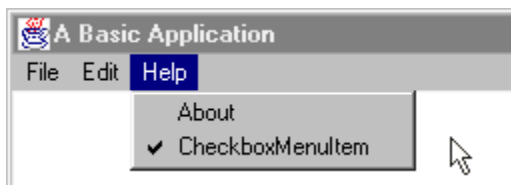
To add a check box menu item to an existing menu, call the menu component's `add` method:

```
menu1.add( new java.awt.CheckboxMenuItem(label) );
```

Part	Type	Description
menuItem1		<code>CheckboxMenuItem</code> The variable name used to refer to the component in code.
label	String	The label for the check box menu item, or null for an unlabeled menu item.
menu1	Menu	The variable name of the menu component to which you are adding the menu item.

### Example

This example shows a single `CheckboxMenuItem` that is activated at runtime.



## CheckboxMenuItem Properties

Checkbox

Class

Enabled

Label

Menu Shortcut

Name

Separator

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A text box with a descriptive label and a button. Clicking on the button displays a drop down list of choices, which are mutually exclusive.

In Visual Cafe, a Choice is a ComboBox without scrollbars. This component is non-editable. Use a [ComboBox component](#) to provide editable properties.

Use a Choice component to:

- specify that your user can select only one choice from a list of choices.
- issue an action event to another component when the user clicks on a choice.

Use the Interaction Wizard to trigger an action when a Choice item is selected.

### Properties

Specify the contents of the choice list by using the [Items property](#). To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item. By default, the first item added to the choice menu becomes the selected item. To make a different item become the selected item, you can change the [Selected Index property](#).

### Coding the component

To create a new Choice containing no text labels in project source code, use this syntax:

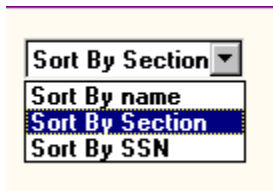
```
choice1 = new java.awt.Choice();
```

Part	Type	Description
------	------	-------------

choice1	Choice	The variable name used to refer to the component in code.
---------	--------	---

### Example

This example shows a Choice component with three elements.



## Choice Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Items](#)

[Name](#)

[Selected Index](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

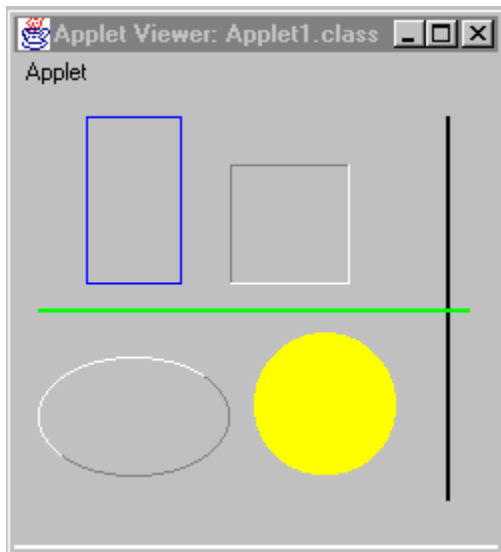
A simple circle. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example

This screen shot shows six predefined Visual Cafe shapes; [Rect](#) , [Circle](#) , [Square](#) , [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.



## Shape Properties

[Background](#)

[Border Style](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Fill Color](#)

[Fill Mode](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A text box that shows one day of the week, and has up and down arrows to the right that allow selecting any day of the week. Use this component to allow your user to select a day of the week.

At run time, only the selected value is displayed in the text box.

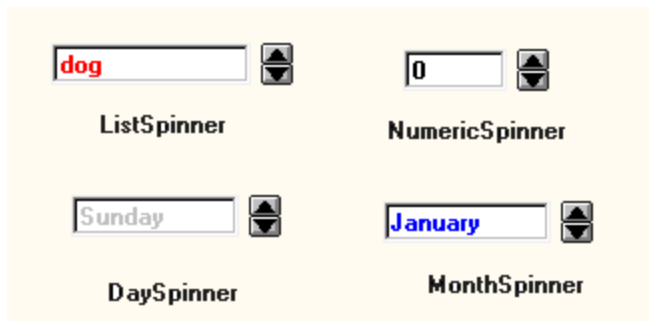
### Properties

Use the [Value property](#) to set the initial day of the week. 0 corresponds to Sunday, 1 to Monday, etc.

Use the [Maximum and Minimum properties](#) to set the maximum and minimum allowed day of the week. 0 corresponds to Sunday, 1 to Monday, etc.

### Example

This screen shot shows the four types of Visual Cafe spinner components. Each component has a label component under it. Various colors were chosen for the Foreground to display the colored text.



{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A dialog box that can be shown and dismissed apart from its parent window. When you add a Dialog component to your project, a new Form Designer window opens. Add components to the Dialog by dragging components and containers to the Form Designer as necessary.

Dialog boxes are intended to be temporary window objects. They present specific timely information to your user, or they allow your user to set options for the current operation. A Dialog can be modal or non-modal. A modal Dialog receives all mouse, keyboard, and focus events that occur while it is open. The user must interact with a modal Dialog window and dismiss the window before returning to its parent window.

Use the Interaction Wizard to open a Dialog in response to an action such as a button press or a menu selection. The Interaction Wizard allows you to open the dialog as modal or non-modal.

Use Dialog to

- prompt the user for information or to make a decision.
- interrupt program processing until your user has provided information or read the message.
- display status on an activity taking place.

Dialog boxes commonly contain the following GUI elements: CheckBox, ComboBox, Button, Choice, BorderPanel, Label, List, Horizontal/Vertical Slider, Numeric/etc Spinner, TabPanel, TextField, TextArea, and more.

Dialogs are empty when created. You must add components, set properties and add interactions as necessary, to create functionality. For programming convenience, Visual Cafe provides the following predefined Dialog containers:

[AboutDialog](#)

[SaveFileDialog](#)

[AttentionDiaog](#)

[PasswordDialog](#)

[OpenFileDialog](#)

[ProgressDialog](#)

### Coding the component

To create a Dialog window in project source code, use this syntax:

```
dialog1 = new java.awt.Dialog(parent, title, modal);
```

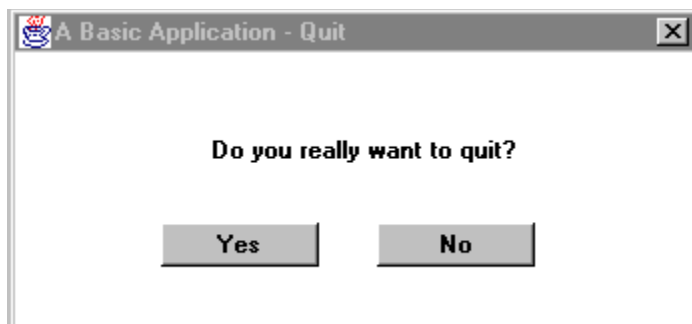
Part	Type	Description
dialog1	Dialog	The variable name used to refer to the component in code.
parent	Frame	The main application window from which the Dialog can be shown.
title	String	The text to display in the dialog box's title bar.
modal	boolean	Whether the dialog is modal or not.

The settings for *modal* are:

Value	Description
true	The dialog box blocks input to other windows. While it is visible dialog box retrieves all user input.
false	The dialog box is modeless. Other windows can be activated and used while the dialog box is open.

### Example

This Dialog component has been customized to act as an Exit verification dialog box. This Quit dialog box is provided for you automatically when you create a new project using the Basic Application template.



{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

An arrow button. At runtime, the button has a raised look when not pressed and a lowered look when pressed.

This component is usually used in conjunction with a combo or list box to indicate a list that the user can view by clicking the arrow.

Use the Interaction Wizard to trigger an action when this button is pressed.

### Properties

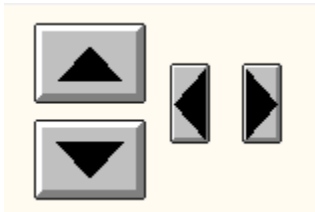
To set the direction of the arrow, use the [Direction property](#).

To define the margin between the arrow and the button sides, use the [Arrow Indent property](#). Type a new value in pixels.

To send a continuous stream of action events when the button is clicked, set the [Notify While Pressed property](#) to “true”, and specify a [Notify Delay property](#) value in milliseconds.

### Example

In the this screen shot, there are four DirectionButtons. The left and right arrow buttons show the default bevel style and arrow look. The up and down arrow buttons have a Bevel Height of 3 and an Arrow Indent of 5.



## DirectionButton Properties

[Arrow Color](#)

[Arrow Indent](#)

[Bevel Height](#)

[Border Color](#)

[Bounds](#)

[Button Color](#)

[Class](#)

[Cursor](#)

[Direction](#)

[Enabled](#)

[Frame Name](#)

[HTML Link URL](#)

[Name](#)

[Notify Delay](#)

[Notify While Pressed](#)

[Show Focus](#)

[Show Link URL Status](#)

[Use Offset](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

An ellipse. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example

This screen shot shows six predefined Visual Cafe shapes; [Rect](#) , [Circle](#) , [Square](#) , [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Creates a box that displays a Emblaze animation file (.BLZ.).

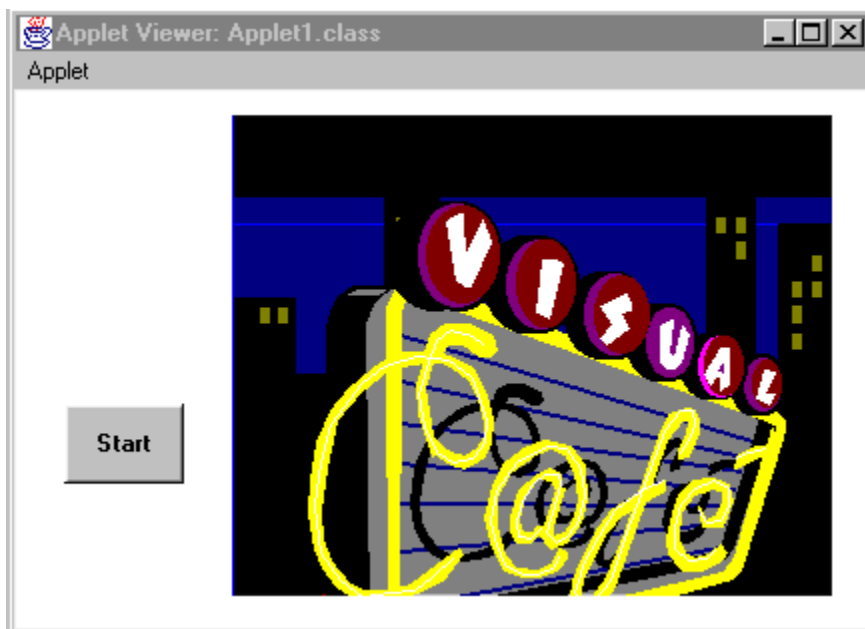
To run the animation, create a button and then use the Interaction Wizard to connect the button to the Emblaze20 component. Select the button trigger action as clicked and the Emblaze action as Start the animation. This connection sends the start method to the Emblaze component when the button is clicked at runtime.

### Properties

To add a Emblaze animation file, double-click on the [URL property](#) to open the URL selection dialog box and specify a .BLZ file.

### Example

This screen shot shows the Emblaze file that is shipped in the Sample directory running in an applet. The button starts the animation.



## Emblaze20 Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[URL](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

 See Also

### Description

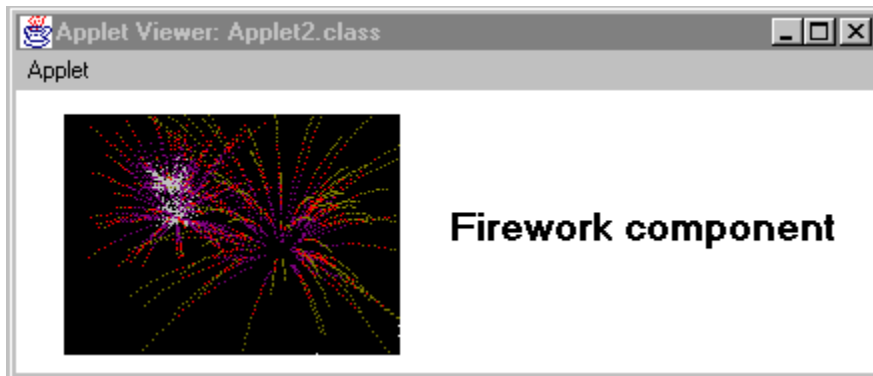
An animated fireworks image. This multimedia component is provided for novelty and programming convenience.

### Properties

There are no properties to control the speed or display of the Firework image.

### Example

The following is a simple example of the Firework component. See the Sample directory for another example.



## Firework Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description


A frame is a top-level window that supports a title bar and menu bar. Visual Cafe uses the frame component to build applications. Frames cannot be placed inside another component.

 See Also

**Note:** You must place a Window component into a Frame. Also, a Dialog component can be placed into a Frame, while an Applet cannot.

Use a frame whenever you need to:

- build a GUI for an application. Frame is the parent container for application GUIs.
- build a container that supports a menu bar.

To create a Frame, select File  New Project and choose Basic Application project. Visual Cafe automatically provides a Frame and a main method, and updates both Frame methods, and property and event defaults as necessary.

You can also drag a Frame from the Palette to the Form Designer window, or from the Component Library to the Project window. Adding a Frame from the Palette or the Component Library may require you to provide a `main` method and override other Frame methods manually.

To add a [MenuBar component](#) to a Frame, drag the MenuBar icon to the Form Designer window.

When a Frame is active, it receives all mouse, keyboard, and focus events that occur over it.

## Frame Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[MenuBar](#)

[Name](#)

[Resizable](#)

[Title](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A horizontal line. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example

This screen shot shows six predefined Visual Cafe shapes; [Rect](#) , [Circle](#) , [Square](#) , [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Scrollbar components preset to display left-to-right and up-down respectively.

A scrollbar is a component used when the contents of a window or box takes up more space than is allocated to that window or box. This, for example, commonly occurs when a large amount of text is displayed in a window. The scrollbar indicates three things. The total length of the contents (corresponding to the length of the scrollbar). The location of the contents shown in the window (corresponding to the position of the scrollbar thumb). And the amount of the contents shown in the window (corresponding to the length of the scrollbar thumb). A different part of the contents may be shown in the window by moving the scrollbar's thumb. The thumb can be moved by dragging, clicking an arrow button, or by clicking between the thumb and an arrow.

Use a ScrollBar to:

- scroll through a data buffer that is too large to display in an associated view window.
- respond to a user event.
- send an action event to another visual component.

Use the Interaction Wizard to trigger an action when the scrollbar's value has changed. A typical event might be painting the appropriate part of the data a window is displaying into that window, thus making the scrollbar "scroll" the data in the window.

### Properties

To change the orientation of a ScrollBar, set the [Orientation property](#) to VERTICAL or HORIZONTAL.



**Note:** HorizontalScrollbar and VerticalScrollbar differ only in the default value of this property. They are actually both components of type `java.awt.Scrollbar`.

To set the location of the scroll thumb in the scroll bar, set the scroll bar's [Value property](#). The [Minimum](#) and [Maximum](#) properties set the range of values the scrollbar may have.

In code, you can move the scroll thumb to it's minimum position by calling the [setValue](#) method for the scrollbar and use the [Minimum property](#) setting as the parameter.

### Coding the component

The three examples below instantiate a scrollbar in project source code:

```
scrollbar1 = new java.awt.Scrollbar(orientation, value, visible, minimum, maximum);
```

Part	Type	Description
scrollbar1	Scrollbar	The variable name used to reference the component in code.
orientation	int	An integer value indicating the orientation of the scroll bar, VERTICAL or HORIZONTAL.
value	int	The initial value of the scroll bar. This parameter controls the starting location of the scroll thumb. This number should be between the minimum and maximum values.
visible	int	The size represented by the thumb of the scrollbar. The scroll bar uses this value when paging up or down by a page.
minimum	int	The minimum value of the scrollbar, when the scroll thumb is in the top or leftmost position.
maximum	int	The maximum value of the scrollbar, when the scroll thumb is in the bottom or rightmost position.

## Scrollbar Properties

[Background](#)  
[Block Increment](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[Enabled](#)  
[Font](#)  
[Foreground](#)  
[Inherit Background](#)  
[Inherit Font](#)  
[Inherit Foreground](#)  
[Maximum](#)  
[Minimum](#)  
[Name](#)  
[Orientation](#)  
[Unit Increment](#)  
[Value](#)  
[VisibleVisible Amount](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also (h)

See Also (v)

### Description

A horizontal or vertical slider component.

A slider is used to select one value from a continuous range of values. It has a movable thumb in front of a gauge with ticks marks on it.

### Properties

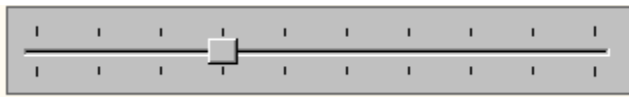
To specify the range of values that the slider indicates, change the [Min Value property](#) and the [Max Value property](#).

To change the number of tick marks on the slider, set the [Tick Frequency property](#).

Use the [Value property](#) to set the initial value of the slider.

### Example

This HorizontalSlider component has a Tick Style of Both and a light gray Background.



## Slider Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Max Value](#)

[Min Value](#)

[Name](#)

[Show Border](#)

[Tick Frequency](#)

[Tick Style](#)

[Value](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A button that displays an image on its face instead of a text label.

Use an ImageButton to:

- display an image instead of text on a button.
- generate a continuous series of action events when the user clicks the button.

### Properties

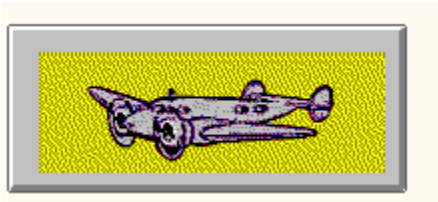
To select and display an image on the button, use the [Image URL property](#). Specify a file name for the JPEG or GIF image you want displayed. Then, if desired, scale the image to fit in the button by using the [Image Style property](#).

To send a continuous series of action events when the button is clicked, set the [Notify While Pressed property](#) to “true”, and specify a [Notify Delay property](#) value in milliseconds.

Use the Interaction Wizard to trigger some action when the button is pressed.

### Example

This ImageButton example has a Bevel Height of 3, Image Style is IMAGE\_CENTERED, and Background property is set to light gray. The button's container's Background property is also gray.



## **ImageButton Properties**

[Bevel Height](#)

[Border Color](#)

[Bounds](#)

[Button Color](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Frame Name](#)

[HTML Link URL](#)

[Image Style](#)

[Image URL](#)

[Name](#)

[Notify Delay](#)

[Notify While Pressed](#)

[Show Focus](#)

[Show Link URL Status](#)

[Use Offset](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Creates a rectangular button, with an image label, that calls a URL address when clicked.

Specifically, use ImageHTMLLink to:

- go to an HTML page when the button is clicked.
- display an image on a button.
- generate a continuous series of action events when the user clicks the button.

Use the Interaction Wizard to trigger an action when this button is clicked.

 See Also

**Note:** This component can only show HTML files if your program is running in a browser.

### Properties

To select and display an image on the button:

1. Using the [Image URL](#), specify the file name of the JPEG or GIF image you want displayed.
2. Scale the image to fit in the button, as desired, by using the [Image\\_Style](#) property.
3. Set the [HTML\\_Link\\_URL](#) to the URL that you want displayed when the button is clicked.

To send a continuous series of action events when the button is clicked, set the [Notify While Pressed property](#) to True, and then specify a [Notify\\_Delay property](#) value in milliseconds.

## ImageHTMLLink Properties

[Bevel Height](#)

[Border Color](#)

[Bounds](#)

[Button Color](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Frame Name](#)

[HTML Link URL](#)

[Image Style](#)

[Image URL](#)

[Name](#)

[Notify Delay](#)

[Notify While Pressed](#)

[Show Focus](#)

[Show Link URL Status](#)

[Use Offset](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

 See Also

### Description

A box containing a list of images and text.

Use an ImageListBox to display a set of images and text that the user can select. Text may be added to the ImageListBox using the Property window or Interaction Wizard. Images and images with text must be added via program code. Use the [addItem](#) method of the ImageListBox component to add images.

### Properties

Use the [ComboBox Mode property](#) to change the highlight action of the ImageListBox. When ComboBox Mode is true list selections will be highlighted anytime the mouse cursor is over them. When false, the normal ListBox highlight is used.

The [Enabled property](#) effects the entire ImageListBox component. Individual list items may be enabled and disabled in program code using the [setEnabled](#) method.

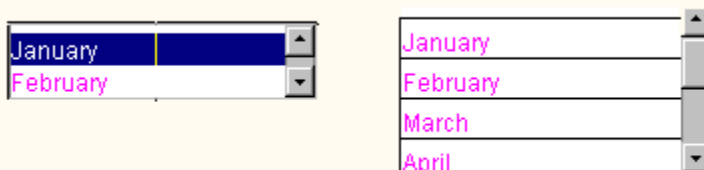
Use the [Rows to Display property](#) to determine the minimum number of rows to display when this component is automatically laid out. This affects the dimensions returned by `getPreferredSize` and `getMinimumSize`.

### Example

This screen shot shows an ImageListBox component (left) and an [ImageViewer component](#) (right, under Preview).

 See Also

This next picture shows an ImageListBox component with no images (on the left). This component has Display Cell Borders set to False and Border Style set to BORDER\_REGULAR. The ImageListBox component on the right has Display Cell Borders set to True and Border Style set to BORDER\_NONE. Both components have the same set of values in the List Items property.



## ImageListBox Properties

[Background](#)

[Border Style](#)

[Bounds](#)

[Class](#)

[Columns to Display](#)

[ComboBox Mode](#)

[Cursor](#)

[Display Cell Borders](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[List Items](#)

[Multiple Selections](#)

[Name](#)

[Rows to Display](#)

[Show Horizontal Scrollbar](#)

[Show Vertical Scrollbar](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

The ImagePanel component is similar to a regular panel except that it displays an image within the panel. The image to use is specified with a URL.

### Properties

Use the [Image URL property](#) to specify the URL of the image to display.

Use the [Image Style property](#) to specify how to display the image.

## ImagePanel Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Image Style](#)

[Image URL](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A `FormattedTextField` in which your user can type text that is limited to an international phone number format (thirteen digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 999/-999/-999/-9999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

An invisible area, usually within an image, that initiates an action when clicked. Specifically, use InvisibleButton to

- create a "hot spot" on an image or on a component
- accept or yield focus
- respond to a user event
- send an action event to another component

Button tips:

- Buttons accept and yield focus automatically by default.
- Buttons accept clicked events automatically by default.
- Use the Interaction Wizard to trigger an action when the button is clicked.

### Example

The following screen shot shows the SlideShow component that is used in the Visual Cafe Tutorial that ships with your release. The SlideShow component displays four package pictures, and an ImageViewer component displays the art work at the bottom that is used as the slide show controller. The SlideShow is "activated" by two InvisibleButtons on the slide controller ImageViewer component. One invisible button overlays the "previous" arrow and the other overlays the "next" arrow. The interactions between the buttons and SlideShow were created with the Interaction Wizard. The text to the right of SlideShow displays is a WrappingLabel component. An interaction is defined between the SlideShow and the WrappingLabel so that the text changes as each slide displays.

See Also

**Tip:** Overlapping components - browsers handle components layered on each other differently. In this SlideShow example, some browsers display/layer the InvisibleButtons on top of the ImageViewer component, while others displayed them underneath. Therefore, this example required two sets of InvisibleButtons, one set on top of ImageViewer and the other underneath.



**Hiking through the rainforest and touring botanical gardens are some of the many ways to explore the Bungbusi tropical paradise.**





## InvisibleButton Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

An invisible rectangular button, usually within an image, that links to a URL address when clicked. Specifically, use InvisibleHTMLLink to:

- Show an HTML document when clicked on.
- Accept or yield focus.
- Respond to a user event.
- Send an action event to another component.

Button tips:

- Buttons accept and yield focus automatically by default.
- Buttons accept clicked events automatically by default.
- Use the Interaction Wizard to trigger an action when the button is clicked.

 See Also

**Note:** This component will only access HTML links when it is running within a Java enabled browser.

### Properties

Use the [HTML Link URL property](#) to specify the address that the browser should access when this link is clicked.

Use the [Frame Name property](#) to specify how the linked document will be displayed.

## InvisibleHTMLLink Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Frame Name](#)

[HTML Link URL](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A container which holds visual components and other panels that the user can TAB through to change focus. You can also use this component to set default buttons (for when the user presses the ENTER key), and to handle OK and CANCEL buttons automatically.

The tab focus order is based on the order in which the components were added to the KeyPressManagerPanel. Each component receives focus in turn, but the mouse cursor is not relocated. Look at the Project window to see the tab order of components within the KeyPressManagerPanel. You can change the tab order by moving the component names in the Project window list.

Tabbing from the last component in the panel changes focus back to the first component. To make the focus leave the KeyPressManagerPanel and continue to the next traversable component in the application, use CTRL+TAB.

Components respond to default events when they receive focus. For example, the TextField component displays text input from the keyboard, and the Button component issues an action event when it is clicked.

Use KeyPressManagerPanel to create a panel whose elements can be tabbed through, and specifically to:

- create a subcontainer that organizes container space within an Applet, Frame or Dialog container. This simplifies your component layout task.
- hold other specialized Panel containers.

### Properties

If desired, use the [Layout property](#) to select a layout manager for this container.

## KeyPressManagerPanel Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A text string in a rectangle with a three-dimensional visual effect. It is usually attached to an option, box, or button.

An application or applet can change the label text string, but a user cannot edit it. Use the [setText](#) method at runtime to change a label's text.

 See Also

**Note:** For most components, labels are usually created by specifying text for the Label property of that component.

### Properties

To specify the label text, type the text in the [Text property](#) field on the Property List window.

You can also specify text alignment, font, color, bevel style, border color and border indent using properties in the Property List window.

### Example

This set of Label3D components show Bevel Styles - Lowered, Raised, Line, and None - and different Alignment Styles. The Border Color in the label with the Line bevel style is blue. Each component has a Border Indent of INDENT\_TWO.



## Label3D Properties

[Alignment Style](#)

[Background](#)

[Bevel Style](#)

[Border Color](#)

[Border Indent](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Text](#)

[Text Color](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A button that has a 3D appearance and whose text can be aligned left, center, or right. In addition, the button can generate a continuous stream of action events while pressed.

A LabelButton appears outlined when it possesses input focus. This indicates that the component can receive a user event. When the user presses the button it gets redrawn so that it appears depressed. When the user then releases the mouse button while still over the button component, a click event is generated.

When a LabelButton is disabled, it is 'grayed out.' This indicates that it can not receive user input.

The button accepts clicked events automatically by default.

Use a LabelButton to:

- Accept or yield focus.
- Respond to a user event.
- Send a train of action events to another component.

Use the Interaction Wizard to trigger an action when the button is clicked.

### Properties

To specify the label text, use the [Text property](#).

To specify the label text alignment, use the [Alignment Style property](#).

To specify the 3D appearance, use the [Bevel Height property](#).

The text string accepts and yields focus automatically by default. To prevent the component from accepting focus, use the [Enabled property](#).

To send a continuous series of action events when the button is clicked, set the [Notify While Pressed](#) property to True, and use the [Notify Delay](#) property to specify a value in milliseconds.

### Properties

To specify the label text, type the text in the [Text property](#).

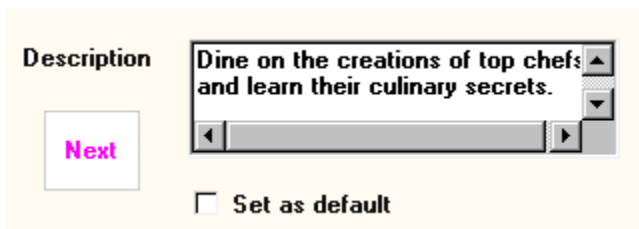
To specify the destination URL, double-click the [HTML Link URL property](#) and select the target file.

Use the [Show Link URL Status property](#) to determine whether the link URL will be displayed in the status area when the mouse is over the button.

To specify the label text alignment, use the [Alignment Style property](#).

### Example

This screen shot includes a LabelButton (Next). The label has a magenta text color.



## LabelButton Properties

[Alignment Style](#)

[Bevel Height](#)

[Border Color](#)

[Bounds](#)

[Button Color](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Frame Name](#)

[HTML Link URL](#)

[Inherit Font](#)

[Name](#)

[Notify Delay](#)

[Notify While Pressed](#)

[Show Focus](#)

[Show Link URL Status](#)

[Text](#)

[Text Color](#)

[Use Offset](#)

[Vertical Alignment Style](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

Creates a button with a text label that displays the document at a URL address when clicked. This component only works from within a Java enabled browser. An applet can change the label text string, but a user cannot edit it.

Use LabelHTMLLink whenever you want to create “hot” text, that links to a URL address when clicked.

### Properties

To specify the label text, type the text in the [Text property](#).

To specify the destination URL, double-click the [HTML Link URL property](#) and select the target file.

## LabelHTMLLink Properties

[Alignment Style](#)

[Bevel Height](#)

[Border Color](#)

[Bounds](#)

[Button Color](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Frame Name](#)

[HTML Link URL](#)

[Inherit Font](#)

[Name](#)

[Notify Delay](#)

[Notify While Pressed](#)

[Show Focus](#)

[Show Link URL Status](#)

[Text](#)

[Text Color](#)

[Use Offset](#)

[Vertical Alignment Style](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A line with variable thickness. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Line Thickness](#) property to vary the width of the line.

Use the [Positive Slope](#) and the [Bounds](#) properties to specify the line endpoints.

## Line Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Line Thickness](#)

[Name](#)

[Positive Slope](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A text box that shows one of a list of items, and has up and down arrows to the right. Use this component to allow your users to move through a set of fixed values.

At run time, only the selected value is displayed in the text box.

### Properties

To add items to the list, use the [List Items property](#).

Use the [Orientation property](#) to determine whether spinner arrows are displayed one above the other (vertically) or side-by-side (horizontally).

### Example

This screen shot shows the four types of Visual Cafe spinner components. Each component has a label component under it. Various colors were chosen for the Foreground property to display the colored text.



## ListSpinner Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Dynamic Resizing](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[List Items](#)

[Maximum](#)

[Minimum](#)

[Name](#)

[Orientation](#)

[Value](#)

[Visible](#)

[Wrappable](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to an local phone number format (seven digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 999/-9999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

## Predefined TextField Properties

[Background](#)

[Bounds](#)

[Class](#)

[Columns](#)

[Cursor](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Mask](#)

[Name](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to an long zip code format (nine digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 99999/-9999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Use this component to create a drop-down menu containing a list of commands.

Menus must be placed within a [MenuBar component](#) . Only Frame components support MenuBars, Menus or MenuItems.

Visual Cafe provides a Menu Editor to help you create menus.

See: [Adding Menus to a Form](#)

### Properties

A menu can remain on the screen after the mouse button has been released by using the [Tear Off property](#).

### Coding the component

Each item in a menu must belong to the [MenuItem](#) class. This can be an instance of MenuItem, a submenu (an instance of Menu), or a check box (an instance of [CheckboxMenuItem](#)).

Create a new menu using the syntax below. You can create a menu with a specified label. Optionally, you can specify that the menu behave like a pop-up menu.:

```
menu1 = new java.awt.Menu(label, tearOff);
```

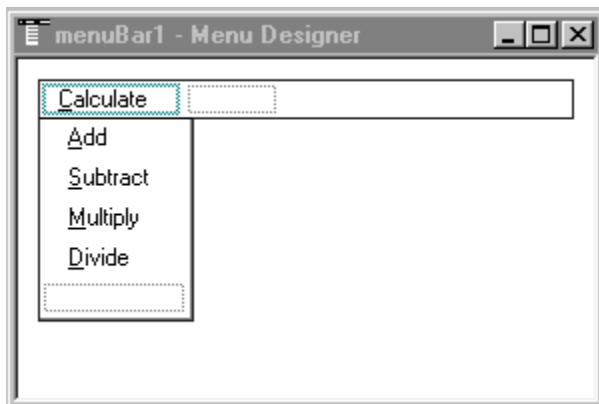
Part	Type	Description
menu1	Menu	The variable name used to reference the component in code.
Label	String	The text that displays in the menu on the menu bar.
tearOff	boolean	A setting that determines whether or not the menu is a tear-off menu.

The settings for *tearOff* are listed below:

Value	Description
true	The menu can be torn off. The menus remains on the screen after the mouse button has been released.
false	(Default.) The menu cannot be torn off.

### Example

In this example of a menu being developed at design time, the element “Calculate” is the Menu component. See Also [MenuBar component](#).



## Menu Properties

[Class](#)

[Help Menu](#)

[Label](#)

[Name](#)

[Tear Off](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Creates a platform-specific horizontal bar containing a list of menus. Use MenuBar to group your menus within a horizontal bar.

Only Frame containers can accept MenuBar components. To display a menu bar in an application, add the MenuBar component to a Frame window.

See: [Adding Menus to a Form](#)

### Runtime modification

At run time, you can display a different menu bar by calling the Frame's [setMenuBar](#) method.

### Coding the component

To create a new menu bar in source code, use this syntax:

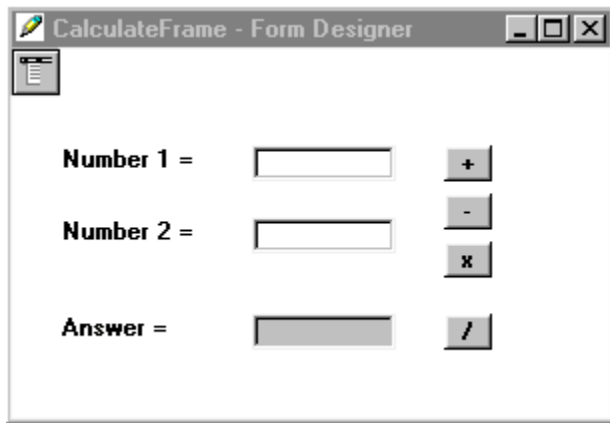
```
mb = new java.awt.MenuBar();
```

Part	Type	Description
------	------	-------------

mb	MenuBar	The variable name used to reference the component in code.
----	---------	--

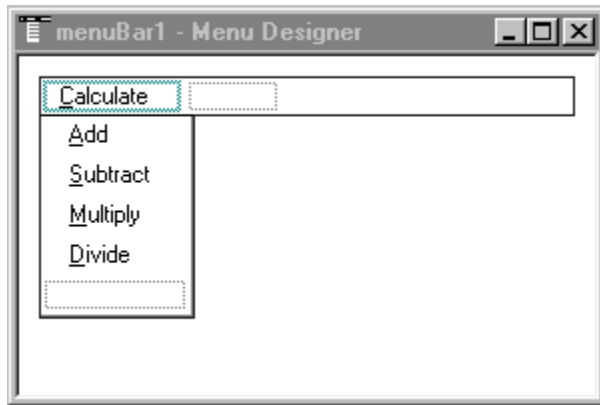
### Example

The following two screen shots show a menu bar at design time and then at runtime.



 See Also

When you double-click on the MenuBar component, the Menu Designer opens so that you can add Menu and MenuItem components. Click on the appropriate placeholder on the menu bar and enter the text that should appear in that component's Label property.



## MenuBar Properties

Class

HelpMenu

Name

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

 See Also

### Description

An item in a menu. Menu items are the commands that appear on a pull-down menu.

Visual Cafe creates menu items for you automatically in the Menu Editor as you edit a [MenuBar component](#).

See: [Adding Menus to a Form](#)

 See Also

**Tip:** When using the Interaction Wizard to connect a MenuItem component to another component, it may be easiest to start the interaction line from the MenuItem in the Menu Designer and drag the line to the appropriate component in the Project window.

### Properties

To create a menu separator, enter a single hyphen (-) in the Label property, or set the [Separator property](#) to true.

### Coding the component

The Java language does not require you to create a variable to reference a menu item. Instead, you can refer to the menu item by its label.

To create a menu item with the specified label in project code, use this syntax:

```
menuItem1 = new java.awt.MenuItem(label);
```

Part	Type	Description
menuItem1	MenuItem	The variable name used to reference the component in code.
label	String	The label for the menu item, or null for an unlabeled menu item.
Menu1	Menu	The variable name of the menu component to which you are adding the menu item.

To create a menu item labelled “Undo” and add it to a Menu with the variable name editMenu, do the following:

```
undoMenu = new java.awt.MenuItem("Undo");  
editMenu.add(undoMenu);
```

Alternatively you could have the editMenu automatically create the MenuItem for you:

```
editMenu.add("Undo");
```

### Example

In the example of a menu being developed at design time, the elements “Add,” “Multiply,” “Subtract”, and “Divide” are MenuItem components.

 See Also

## MenuItem Properties

Class

Enabled

Label

Menu Shortcut

Name

Separator

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A text box that shows one month of the year, and has up and down arrows to the right that allow selecting any month of the year. Use this component to allow your user to select a month.

At run time, only the selected value is displayed in the text box.

### Properties

Use the [Value property](#) to set the initial month. 0 corresponds to January, 1 to February, etc.

Use the [Maximum and Minimum properties](#) to set the maximum and minimum allowed month. 0 corresponds to January, 1 to February, etc.

### Example

This screen shot shows the four types of Visual Cafe spinner components. Each component has a label component under it. Various colors were chosen for the Foreground property to display the colored text.

 See Also

## Month/DaySpinner Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Dynamic Resizing](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[List Items](#)

[Maximum](#)

[Minimum](#)

[Name](#)

[Orientation](#)

[Value](#)

[Visible](#)

[Wrappable](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

Creates an animation by displaying a series of images in sequence. This animation moves within the component while the images display. The initial image is drawn at the left edge of the component. Each subsequent image is drawn within the component, but the origin of the image is shifted left or right by the amount specified by the [Shift Offset property](#).

### Properties

To specify the animation sequence, double-click on the [URL List property](#) to display the URL List dialog box. In the dialog box, enter a list of JPEG or GIF graphic files. List the files in the order you want them to be displayed.

Run an animated sequence one or more times by setting the [Loop Count property](#).

Ignore the loop count and run an animated sequence forever by setting the [Repeat Mode property](#) to true.

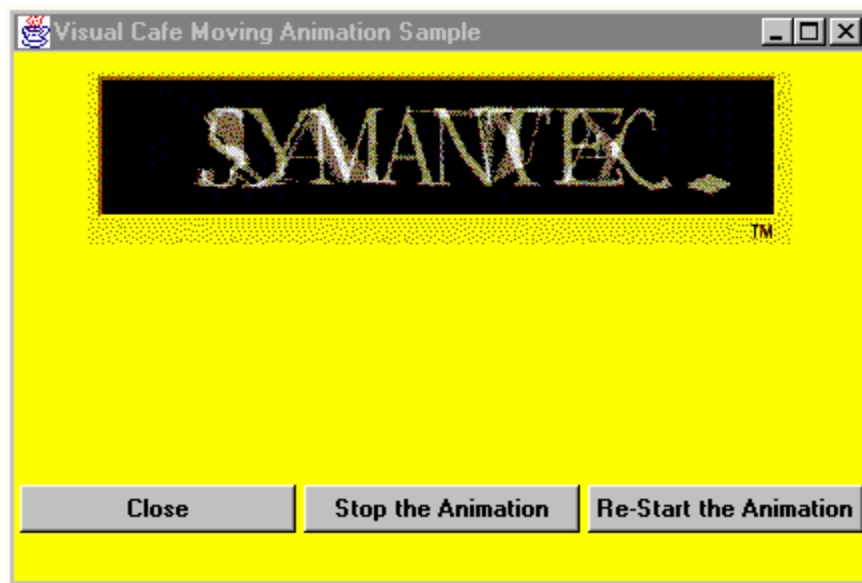
To view your animation at design time, set the [Preview Component property](#) to true.

To specify the movement of the animated image, use the [Shift Offset property](#). The value is pixels per frame. A negative value shifts the image to the left.

To allow the user, or the program, to start and stop the animation, use the Interaction Wizard.

### Example

This screen shot is of the MovingAnimation component in the Samples directory.



## **MovingAnimation Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Clear Frame](#)

[Cursor](#)

[Delay](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Loop Count](#)

[Name](#)

[Preview Component](#)

[Repeat Mode](#)

[Shift Offset](#)

[URL List](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

 See Also

### Description

Creates a box that displays a matrix of items which the user can select. The user cannot type or edit a selection in a list box.

The user can resize a column at run-time by dragging the column boundary to a new position.

Use the Interaction Wizard to trigger an action when a MultiList row is selected. You can not use visual tools to create an interaction between a list item and another component, this can only be done in source code.

See: [Creating a MultiList](#)

### Properties

To create columns, use the [Column Headings property](#) to specify a label for each column. List the columns from left to right. By default all columns have the same width, and headings are clipped if necessary.

Use the [Heading Font property](#) to specify the font used in the headings.

To change the default column widths, use the [Column Sizes property](#) to specify the width of each column in pixels.

Use the [Column Alignments property](#) to specify how the text in each column is justified.

To include an item in a column, use the [List Items property](#). Place a semicolon (;) in between the text for each column.

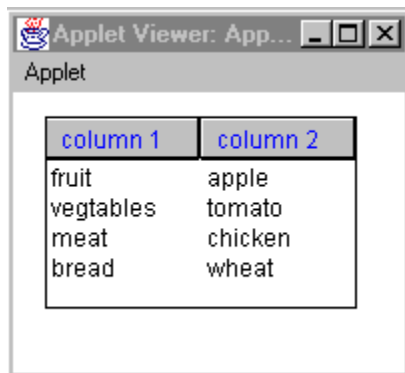
Use the [Allow Sorting property](#) to specify whether the list gets sorted when a column heading is clicked.

Use the [Multiple Mode property](#) to allow multiple rows to be selected at once.

### Example

The following MultiList component has two Column Heading property values, "column1" and "column2", and the following text entry in the List Item property.

```
fruit;apple  
vegetables;tomato  
meat;chicken  
bread;wheat
```



## MultiList Properties

[Allow Sorting](#)

[Background](#)

[Bounds](#)

[Cell BackGround](#)

[Cell Font](#)

[Cell Foreground](#)

[Class](#)

[Column Alignments](#)

[Column Headings](#)

[Column Sizes](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Heading Background](#)

[Heading Font](#)

[Heading Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[List Items](#)

[Multiple Mode](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A text box that shows one of a list of numbers, and has up and down arrows to the right that allow selecting any one of a range of fixed values.

At run time, only the selected value is displayed in the text box.

### Properties

Use the [Value property](#) to set the initial value.

Use the [Maximum and Minimum properties](#) to set the maximum and minimum allowed values.

### Example

This screen shot shows the four types of Visual Cafe spinner components. Each component has a label component under it. Various colors were chosen for the Foreground to display the colored text.



### Example

This screen shot shows the four types of Visual Cafe spinner components. Each component has a label component under it. Various colors were chosen for the Foreground to display the colored text.



## **NumericSpinner Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Editable](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Increment](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Maximum](#)

[Minimum](#)

[Name](#)

[Orientation](#)

[Value](#)

[Visible](#)

[Wrappable](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A modal dialog that allows the user to specify a file to be opened. The exact look and behavior of this dialog depends on the native operating system the program is running under and cannot be changed.

 See Also

**Note:** Both `OpenFileDialog` and `SaveFileDialog` are objects of type `java.awt.FileDialog`. They only differ in the value of the [Mode property](#). `OpenFileDialogs` have a mode of “LOAD”, while `SaveFileDialogs` have a mode of “SAVE”.

 See Also

**Note:** Both `OpenFileDialog` and `SaveFileDialog` cannot be used in Applets for security reasons. They can only be used in applications.

### Properties

Use the [Default Directory property](#) to specify the initial directory that is shown when the dialog is displayed.

Set the [Default File Name property](#) to specify an initial file name or file name template to use when the dialog is displayed.

## FileDialog Properties

Class

Default Directory

Default File Name

Mode

Name

Title

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A generic container. A panel can contain other visual components and panels. Panels do not support menus.

Use Panel to

- create a subcontainer that organizes container space within an Applet, Frame, TabPanel, Dialog, etc. This simplifies your component layout task.
- form a basis for a layout style.
- hold other specialized Panel containers.

The following predefined subclasses of Panel are available.

<a href="#">Applet</a>	<a href="#">BorderPanel</a>	<a href="#">Calendar</a>
<a href="#">ComboBox</a>	<a href="#">DaySpinner</a>	<a href="#">KeyPressManagerPanel</a>
<a href="#">ImageListBox</a>	<a href="#">ImagePanel</a>	<a href="#">MultiList</a>
<a href="#">ListSpinner</a>	<a href="#">MonthSpinner</a>	<a href="#">ScrollingPanel</a>
<a href="#">NumericSpinner</a>	<a href="#">RadioButtonGroupPanel</a>	<a href="#">SplitterPanel</a>
<a href="#">ScrollPane</a>	<a href="#">SlideShow</a>	<a href="#">ToolBarPanel</a>
<a href="#">StatusBar</a>	<a href="#">TabPanel</a>	
<a href="#">TreeView</a>	<a href="#">Wizard</a>	

### Properties

Use the [Layout property](#) to set the desired layout manager.

### Coding the component

To create a new Panel, use the following syntax:

```
panel1= new java.awt.Panel();
```

Part	Type	Description
panel1	Panel	The variable name used to reference the component in code.

## Panel Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A dialog that prompts the user for a user name and password. This is a simple dialog with user id and password fields. When you add a PasswordDialog component, a new Form Designer window opens. You can extend this dialog by adding components. This is done by dragging components and containers to this Form Designer window as necessary.

Use PasswordDialog to prompt the user for a name and password. Dialog boxes are intended to be temporary window objects.

### Properties

Set the password display character by selecting the password field and entering a character in the [echo property](#).

### Example

This is the default Password dialog box.

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

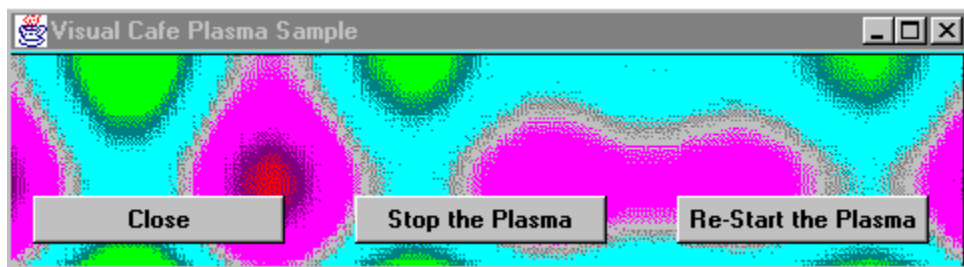
Creates an animation of colored amorphous shapes, where colors gradually modulate as shapes merge and separate. This multimedia component is provided for novelty and programming convenience.

### Properties

To turn off the animation of the Plasma component during design time, use the [Preview Component property](#).

### Example

This is a screen shot of the Plasma component. Buttons have been added and connected to the Plasma component by using the Interaction Wizard to start and stop the display.



## Plasma Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Preview Component](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to an postal code format (a mix of characters and numbers). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to A9A/-9A9. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

A bar that displays the percentage completed of a particular process. It is commonly used to indicate the percentage completed of a printing task or installing a program. The bar has no user interaction.

See also: [ProgressDialog](#)

### Properties

To show the percent complete as a percentage in addition to graphically as a bar, use the [Show Percentage property](#).

To set the percent complete a process is, use the [Value property](#). For example, a value of 33 would indicate 33% complete.

To specify the color of the progress bar, use the [ProgressBar Color property](#).

To specify whether the progress bar is drawn as a single solid bar or a series of boxes, use the [Box Bar Style property](#).

To specify the box width when the progress bar is drawn as a series of boxes, use the [Box Width property](#).

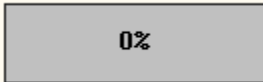
To specify the size of the gap between boxes when the progress bar is drawn as a series of boxes, use the [Gap Width property](#).

### Coding the component

While a process is underway, periodically note the percentage complete it is by calling the [setProgressPercent](#) method.

### Example

The following is a default ProgressBar component.



## ProgressBar Properties

[Alignment Style](#)

[Background](#)

[Bevel Style](#)

[Border Color](#)

[Border Indent](#)

[Bounds](#)

[Box Width](#)

[Class](#)

[Cursor](#)

[Draw as Boxes](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Gap Width](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[ProgressBar Color](#)

[Show Progress](#)

[Value](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A simple dialog that displays the progress of a lengthy task. You can extend this dialog by adding components. This is done by dragging components and containers to this Form Designer window as necessary.

Use a ProgressDialog to

- Display print status
- Show percent of a file copied
- Indicate progress of any task

The progress dialog contains a [ProgressBar component](#). Use the interaction Wizard to update the percentage shown on the progress bar by getting the value from another component or an expression. At runtime, call the ProgressBar's [setProgressPercent](#) method to update the display.

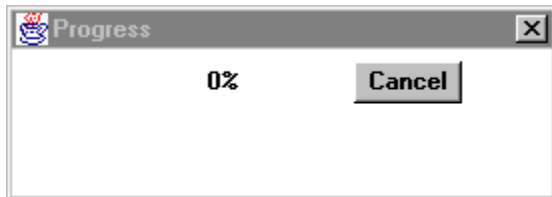
See also: [ProgressBar component](#)

### Properties

To change the dialog window title, use the [Title property](#).

### Example

This example is of the default Progress dialog box.



{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Extends Panel.

Creates an rectangular panel area that automatically groups the radio buttons that it contains.

Add radio buttons to the panel by dragging them from the Component Palette onto the panel.

 See Also

**Note:** Since CheckBoxes and RadioButtons are actually the same base component, any CheckBoxes that are dropped into the RadioButtonGroupPanel turn into RadioButtons.

All the radio buttons within the RadioButtonGroupPanel act together. Only one of the radio buttons can be "on" at a time. This is accomplished by making each RadioButton added to the panel a member of the panel's private CheckboxGroup.

### Example

This example shows a RadioButtonGroupPanel (in gray) that contains two RadioButton components.



## RadioButtonGroupPanel Properties

[Background](#)  
[Border Color](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[Enabled](#)  
[Font](#)  
[Foreground](#)  
[Inherit Background](#)  
[Inherit Font](#)  
[Inherit Foreground](#)  
[Inset Padding Bottom](#)  
[Inset Padding Sides](#)  
[Inset Padding Top](#)  
[Label](#)  
[Label Alignment](#)  
[Label Color](#)  
[Layout](#)  
[Name](#)  
[Padding Bottom](#)  
[Padding Left](#)  
[Padding Right](#)  
[Padding Top](#)  
[Selected RadioButton](#)  
[Style](#)  
[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A simple rectangle. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example

This screen shot shows six predefined Visual Cafe shapes; [Rect](#) , [Circle](#) , [Square](#) , [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

This is a button that allows three different states and displays the document with a given URL when clicked. Each button state has its own associated image. Each image is specified by providing the file name or the URL of the file containing the image. The document to show is specified by providing the "LinkURL" of that document.

The "standard" state is when the mouse is not over the button. If the standard image is not specified, the button will be transparent when in the standard state, otherwise the specified image will be displayed.

The "over" state is when the mouse cursor is over the button. This state also displays the LinkURL if it is specified.

The "down" state is when the mouse button is pressed inside the button. When there is a MOUSE\_UP in the button, it will attempt to show the document at the LinkURL unless it is not specified. This state also displays the LinkURL if it is specified.

### Properties

Use the [Standard Image property](#) to specify the image for when the mouse is not over the button. If the standard image is not specified, the button will be transparent when in the standard state.

Use the [Mouse Over Image property](#) to specify the image for when the mouse cursor is over the button.

Use the [Mouse Down Image property](#) to specify the image for when the mouse button is pressed inside the button.

Ensuring the [Clear Frame property](#) is set to false prevents flickering caused by repainting the component background before drawing the component image.

To center the image in the button, use the [Center Mode property](#).

## **RollOverButton Properties**

[Background](#)

[Bounds](#)

[Center Mode](#)

[Class](#)

[Clear Frame](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[HTML Link URL](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Mouse Down Image](#)

[Mouse Over Image](#)

[Name](#)

[Standard Image](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A modal dialog that allows the user to specify a filename to be used for saving a file. The exact look and behavior of this dialog depends on the native operating system the program is running under and cannot be changed.

 See Also

**Note:** Both `OpenFileDialog` and `SaveFileDialog` are objects of type `java.awt.FileDialog`. They only differ in the value of the [Mode property](#). `OpenFileDialogs` have a mode of “LOAD”, while `SaveFileDialogs` have a mode of “SAVE”.

 See Also

**Note:** Both `OpenFileDialog` and `SaveFileDialog` cannot be used in Applets for security reasons. They can only be used in applications.

### Properties

Use the [Default Directory property](#) to specify the initial directory that is shown when the save dialog is displayed.

Set the [Default File Name property](#) to specify an initial file name or file name template to use when the save dialog is displayed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A simple container with scroll bars.

The ScrollingPanel is a panel with automatic scroll bars. It may contain only one component/container/panel/etc. It automatically tracks the size of this (usually large) component and provides scroll bars so the entire component may be viewed within a panel of limited size. Typically, the added component would be a panel (or some other container) which would then contain a variety of other components.

Typically a ScrollingPanel is used to view a portion of a larger container or component which is contained within the ScrollingPanel. Scroll bars are then used to change the portion of the container or component which is in view. A small ScrollingPanel could be used to display a portion of a large image, for example, and the user would use the scroll bars to move the view to other parts of the image.

While a ScrollingPanel can only contain one object, this object can be a Panel full of other components.

Use this component to display a region that allows the user to scroll through all items in the panel, and specifically to:

- create a subcontainer that organizes container space within an Applet, Frame or Dialog container. This simplifies your component layout task.
- hold other specialized Panel containers.

See: [Creating a ScrollingPanel](#)

 See Also

**Note:** JDK 1.1 now has a very similar component, [ScrollPane](#). It is recommended that you use that component unless you need some of the enhanced properties of the ScrollingPanel Component. That includes properties: [Minimum Height and Width](#), and [Scroll Increment](#).

To specify the minimum dimensions returned by the `getMinimumSize` method, use the [Minimum Height and Width properties](#)

To specify the increment, in pixels, that the panel scrolls when a scrollbar arrow is clicked, use the [Scroll Increment property](#).

## ScrollingPanel Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Minimum Height](#)

[Minumum Width](#)

[Name](#)

[Scroll Increment](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A simple container with scroll bars.

The ScrollPane is a panel with automatic scroll bars. It may contain only one component/container/panel/etc. It automatically tracks the size of this (usually large) component and provides scroll bars so the entire component may be viewed within a panel of limited size. Typically, the added component would be a panel (or some other container) which would then contain a variety of other components.

Typically a ScrollPane is used to view a portion of a larger container or component which is contained within the ScrollPane. Scroll bars are then used to change the portion of the container or component which is in view. A small ScrollPane could be used to display a portion of a large image, for example, and the user would use the scroll bars to move the view to other parts of the image.

While a ScrollPane can only contain one object, this object can be a Panel full of other components.

Use this component to display a region that allows the user to scroll through all items in the panel, and specifically to:

- create a subcontainer that organizes container space within an Applet, Frame or Dialog container. This simplifies your component layout task.
- hold other specialized Panel containers.

Use the [Scrollbar Display Policy property](#) to choose whether the scroll bars get displayed as needed, or are always/never displayed.

## ScrollPane Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Scrollbar Display Policy](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

A scrolling text component. It forms a text banner that scrolls specified text horizontally. The component allows multiple messages and links.

Use the [Messages property](#) to enter a list of messages to display. To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item.

Use the [Message Links property](#) to enter a list of URLs to link to. To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item.

Use the [Scroll Direction property](#) to specify the direction the messages should scroll.

To set the delay, in milliseconds, between scroll steps, use the [Scroll Interval property](#).

To set the amount the text moves, in pixels, each scroll step, use the [Scroll Unit property](#).

## ScrollingText Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Frame Name](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Message Hilite Color](#)

[Message Links](#)

[Messages](#)

[Name](#)

[Scroll Direction](#)

[Scroll Interval](#)

[Scroll Unit](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Displays a series of images in an applet or application. You can provide descriptive text for each image.

 See Also

**Note:** When encountering an incorrect URL, SlideShow sends an error to standard output and goes to the next URL.

### Properties

To specify a list of files containing displayable images, double-click in the [URL List property](#) field to display the URL List dialog. Images display in the order that they are placed on the list.

To associate a text string with a specified URL address, enter the text in the Description column of the URL List dialog box.

### Coding the component

Use this syntax to create a slide show component in project source code:

```
slideShow1 = new symantec.itools.awt.SlideShow();
```

Part	Type	Description
------	------	-------------

slideShow1	SlideShow	The variable name used to reference the component in code.
------------	-----------	--

### Example

The following screen shot shows the SlideShow component that is used in the Visual Cafe DDE Tutorial. The SlideShow component displays four package pictures, and an ImageViewer component displays the art work at the bottom that is used as the slide show controller. The SlideShow is "activated" by two InvisibleButtons on the slide controller ImageViewer component. One invisible button overlays the "previous" arrow and the other overlays the "next" arrow. The interactions between the buttons and SlideShow were created with the Interaction Wizard. The text to the right of SlideShow displays is a WrappingLabel component. An interaction is defined between the SlideShow and the WrappingLabel so that the text changes as each slide displays.

 See Also

**Tip:** Since components do not recognize an initialization event, to display the first slide's description in the WrappingLabel, the first text string was entered in the label's Text property as the default text.

 See Also

**Tip:** Overlapping components - browsers handle components layered on each other differently. In this SlideShow example, some browsers display/layer the InvisibleButtons on top of the ImageViewer component, while others displayed them underneath. Therefore, this example required two sets of InvisibleButtons, one set on top of ImageViewer and the other underneath.

 See Also

## SlideShow Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[URL List](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to an social insurance number format (nine digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 999/-999/-999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to an social security number format (nine digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 999/-99/-9999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Creates a nonvisual class that plays a Sun AU format sound file. The design time component looks like this



The SoundPlayer component is not visible at runtime. Use other components, such as radio buttons and image buttons, to control the playing of the sound. Make the component connections by using the Interaction Wizard.

 See Also

**Note:** The target system must have been configured with optional Sun Java classes in order for this component to work.

### Properties

Use the [URL List property](#) to specify the sounds clips to play.

Use the [Synchronized Mode property](#) to specify whether the list of sound clips play serially or all at once. If true, the sound clips play serially, otherwise they all play at once.

Use the [Repeat Count property](#) to specify the number of times the list of sound clips is played.

## SoundPlayer Properties

[Class](#)

[Name](#)

[Repeat Count](#)

[Synchronized Mode](#)

[URL List](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A container that can be divided into a number of subpanels, which holds visual components and other panels.

Use `SplitterPanel` specifically to:

- create a subcontainer that organizes container space within an Applet, Frame or Dialog container. This simplifies your component layout task.
- hold other specialized Panel containers.

The `SplitterPanel` component is the parent Panel containing a set of subpanels. You must write code to divide the parent panel into subpanels and to populate those subpanels. See the “Coding the component” sample below to see how this is done.

`SplitterPanel` automatically recognizes the `MouseDown` event as the command to resize the appropriate subpanel boundary.

### Runtime modification

Subpanels can be resized at runtime by dragging panel borders with the mouse.

### Coding the component

This code example splits a `SplitterPanel` into four subpanels, and adds a button to two of the subpanels.

```
// divide the parent panel into subpanels
splitterPanel1.split(splitterPanel1.SPLIT_HOR);
splitterPanel1.getTopPanel().split(splitterPanel1.SPLIT_VER);
splitterPanel1.getBottomPanel().split(splitterPanel1.SPLIT_VER);
// add a button to the top left subpanel
splitterPanel1.getTopLeftPanel().add(new Button("Top Left"));
// add a button to the lower right subpanel
splitterPanel1.getBottomRightPanel().add(new Button("Bottom Right"));
```

Individual subpanels can be accessed in code using any of these methods:

- `getBottomLeftPanel`
- `getBottomPanel`
- `getBottomRightPanel`
- `getLeftPanel`
- `getRightPanel`
- `getSub2Panel`
- `getSubPanel`
- `getTopLeftPanel`
- `getTopPanel`
- `getTopRightPanel`

## **SplitterPanel Properties**

[Allow Dynamic Moving](#)

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Enforce Minimum Dimension](#)

[Font](#)

[Foreground](#)

[Gap Color](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Propagate Resize](#)

[Use 3D Border](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A simple square. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example


This screen shot shows six predefined Visual Cafe shapes; [Rect](#) , [Circle](#) , [Square](#) , [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.

 See Also

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A two- or three-state checkbox that doesn't have an associated text label. -- . A two-state checkbox behaves just like the standard checkbox. In three-state mode clicking the checkbox can switch the checkbox to a third state that appears both gray and checked.

The third state is often used to indicate that an attribute of the current selection varies across the selected items. For example, a checkbox used to indicate "bold text" might use the third state if the currently selected text included both bold and plain characters. It is also used to indicate a default action or state.

This checkbox does not have a text string associated with it.

Display an option with fixed states, and specifically to

- manage a selection-deselection user task.
- display a check mark.
- post an event to another component.
- determine that only one choice in a set of CheckBoxes is selected.

Use the Interaction Wizard to trigger an action when the check box is clicked.

### Properties

Specify the number of option states, either two or three by using the [Style property](#). Style determines the number of "states" that the checkbox can have.

Set the initial state, either checked, unchecked, or default (the third state), by using the [State property](#).

## StateCheckBox Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[State](#)

[Style](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A rectangular area suitable for displaying status text.

Use a StatusBar to show document status and other information, like the meaning of a button or other interface element in a window. Typically, a status bar appears at the bottom of a window.

### Runtime modification

Use the [setStatusText](#) method to change the status bar text when the application or applet is running.

### Properties

Set the text to display in the status bar by editing the [Status Text property](#).

Make the StatusBar border raised, inset, plain, or absent by changing the [Bevel Style property](#).

To align a StatusBar to the bottom of a window, set the containing window's Layout property to BorderLayout, and then set the StatusBar's [Placement property](#) to South.

To change the distance between the drawn border and the usable display area within the border, use the [Inset properties](#).

To change the distance between the drawn border and the actual bounds of the component, use the [Padding properties](#).

### Example

This simple StatusBar component has a Style of Lowered and a predefined status text string.



## StatusBar Properties

[Background](#)  
[Bevel Style](#)  
[Border Color](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[Enabled](#)  
[Font](#)  
[Foreground](#)  
[Inherit Background](#)  
[Inherit Font](#)  
[Inherit Foreground](#)  
[Inset Padding Bottom](#)  
[Inset Padding Sides](#)  
[Inset Padding Top](#)  
[Label](#)  
[Label Alignment](#)  
[Label Color](#)  
[Name](#)  
[Padding Bottom](#)  
[Padding Left](#)  
[Padding Right](#)  
[Padding Top](#)  
[Status Text](#)  
[Status Text Color](#)  
[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Displays a scrolling message in the status window of a browser or applet viewer.

### Runtime modification

Use the [setString](#) method to change the status bar text when the application or applet is running.

### Properties

Set the text to display in the StatusScroller by editing the [String property](#).

To scroll the message repeatedly, set the [Repeat property](#).

To specify the direction the message should scroll, use the [Right To Left property](#).

To specify whether the scrolling will automatically start when the applet is loaded, use the [Auto Start property](#).

To specify whether the message will scroll completely off before scrolling on again, use the [Scroll Clean property](#).

## StatusScroller Properties

[Auto Start](#)

[Class](#)

[Delay](#)

[Name](#)

[Repeat](#)

[Right To Left](#)

[Scroll Clean](#)

[String](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

The TabPanel is a Panel extension which provides for a tabbed dialog effect. Along the top (by default) of the panel is a series of file folder-like tabs, each with a text label. Each tab is associated with a panel or component that gets shown when the user clicks on the tab. The TabPanel automatically manages swapping panels when a tab selected. This arrangement conserves window space.

See: [Creating a TabPanel](#)

### Properties

To define the tab text strings, use the [Tab Labels](#) property.

To display tabs at the bottom of the TabPanel component, set the [Tabs On Bottom](#) property to true.

To set which subpanel is initially shown, use the [Active Tab property](#).

To change the order of the subpanels, change their order in the Project window.

### Coding the component

In code, to create a tab, a tab label, and an associated panel, call the [addTabPanel](#) method. Call the [addTabPanel](#) method for each tab you create.

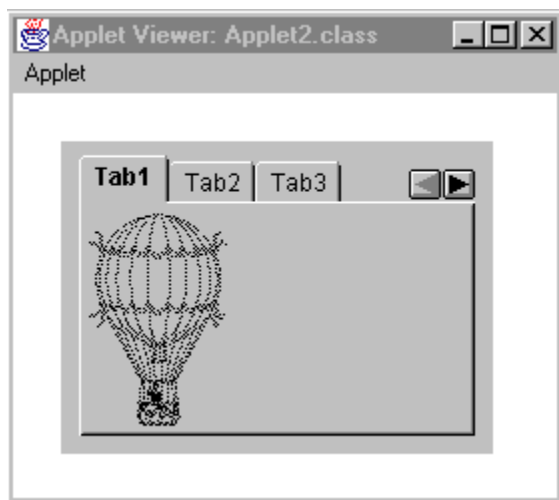
To access any tabbed subpanel in project source code, use the [getTabPanel](#) method.

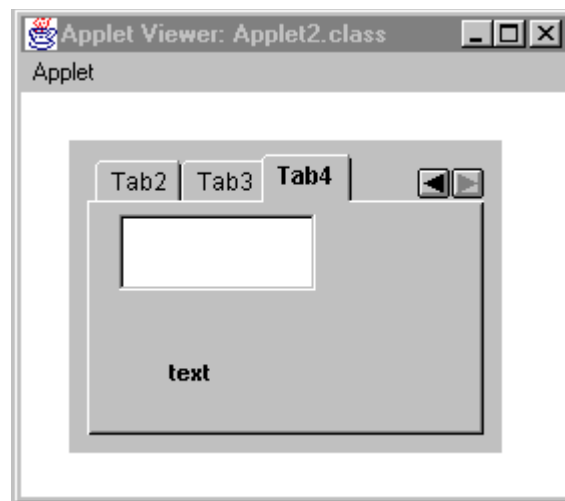
To get the index of the currently active subpanel in project source code, use the [getCurrentPanelNdx](#) method.

### Examples

This first screen shot is of a simple five tab panel at runtime. The first tab panel contains an [ImageViewer component](#). This also shows the “scroll” arrows that display automatically when all tabs cannot be displayed within the component.

The second screen shot shows a tab that contains a default Label and TextField component. Remember, a Panel component must be added to the tab prior to the other standard components for grouping ability.





## **TabPanel Properties**

[Active Tab](#)

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Tab Labels](#)

[Tabs On Bottom](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

### Description

A Timer is an invisible component that generates action events after waiting a specified number of milliseconds.

Use the Interaction Wizard to trigger an action when the timer generates its action event(s).

The Timer displays in the Form Designer at design time, as an icon, but does not display at runtime.



You can also instantiate a Timer component in your project source code.

### Properties

To set the timer, specify the “wait” time in the [Time property](#).

To specify that the timer run, reset, and start continuously, use the [Repeat Automatically property](#).

### Coding the component

Timer does not extend the Component class. It therefore does not have common Component methods, such as `setBounds`, `setVisible`, etc.

Use this syntax to create a Timer component in project source code:

```
timer1 = new symantec.itools.util.Timer(delay, repeat);
```

Part	Type	Description
timer1	Timer	The variable name used to reference the component in code.
delay	int	The delay before the timer fires an action event, in milliseconds.
repeat	boolean	Whether the timer repeatedly generate action events.

## Timer Properties

Class

Enabled

Name

Repeat Automatically

Time (ms)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

This component creates a panel to which you can add buttons to create a toolbar in a window. This toolbar can allow users access to common functionality in your application or applet. The toolbar is fixed and does not float.

To add space between items on the toolbar, add a [ToolBarSpacer component](#) between elements.

 See Also

**Tip:** The easiest way to add the ToolBarSpacer is to drag the component to the correct position in the Project window.

Toolbars commonly contain buttons, but a ToolBarPanel can hold other types of components like static text, check boxes, even images.

Use the Interaction Wizard to trigger an action when a toolbar's button (or other component) is pressed.

### Properties

To align a ToolBarPanel at the top of a window, set the parent window's [Layout property](#) to BorderLayout, and then set the ToolBarPanel's [Placement property](#) to North. To align the toolbar on the left side of the window, set its Placement property to West.

To make the toolbar border raised, inset, or plain, specify the appropriate bevel look in the [Bevel Style property](#).

To title the toolbar, use the [Label property](#), the [Label Alignment property](#) and the [Label Color property](#).

To change the distance between the drawn border and the usable display area within the border, use the [Inset Padding properties](#).

To change the distance between the drawn border and the actual bounds of the component, use the [Padding properties](#).

### Example

The following is a ToolBarPanel that contains three ImageButton components and two ToolBarSpacer components.



## **ToolBarPanel Properties**

[Background](#)

[Bevel Style](#)

[Border Color](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Inset Padding Bottom](#)

[Inset Padding Sides](#)

[Inset Padding Top](#)

[Label](#)

[Label Alignment](#)

[Label Color](#)

[Name](#)

[Orientation](#)

[Padding Bottom](#)

[Padding Left](#)

[Padding Right](#)

[Padding Top](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

Use ToolBarSpacer to add a gap between elements in a [ToolBarPanel component](#).

 See Also

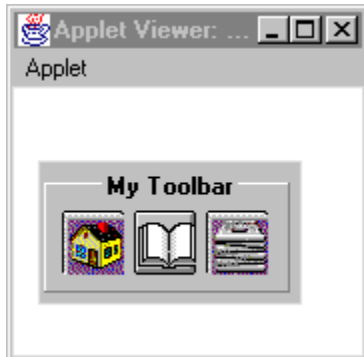
**Tip:** Although you can drag the ToolBarSpacer on to the ToolBarPanel in the Form Designer, you may find it easier to drag the ToolBarSpacer into the Project Window and insert it between objects.

### Properties

To vary the width of the spacer in pixels, use the [Space](#).

### Example

The following is a ToolBarPanel that contains three ImageButton components and two ToolBarSpacer components.



## ToolBarSpacer Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Space](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

See Also

### Description

An "outline view" of text headings and, optionally, images. The headings are arranged in a hierarchical fashion, and can be expanded to show their sub-headings or collapsed, hiding their sub-headings.

A TreeView is typically used to display information that is organized in a hierarchical fashion like an index or table of contents.

Use the Interaction Wizard to trigger an action in response to TreeView actions. A TreeView generates an action event when a node is expanded/collapsed, double-clicked, or ENTER is pressed.

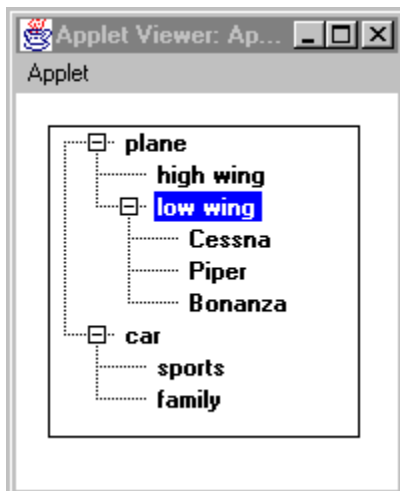
See: [Creating a TreeView](#)

### Properties

To add items to a TreeView, double-click the [Items property](#) to display the editing pane and enter the text you want for the label. Entering one or more spaces before the text label indents that item one or more levels in the resulting tree hierarchy. To include more than one item in the list, press CTRL+ENTER (on PCs) after typing each item.

### Example

The following screen shot is of a TreeView component with nine entries in the Items property.



## TreeView Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Items](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Extends PhoneNumber.

Creates a box in which your user can type text that is limited to a US Domestic long distance phone number (ten-digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

To post TextField events or text to other components, use the Interaction Wizard.

### Properties

To specify the required format for the text entry, use the [Mask property](#) in the Property List window. The allowed format is 999/-999/-9999.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

 See Also

### Description

A vertical line. This component is provided for programming convenience, so that you don't have to draw the shape.

### Properties

Use the [Border Style](#) , [Fill Mode](#) , [Fill Color](#) , [Background](#) , and [Foreground](#) properties to achieve the desired component look.

### Example

This screen shot shows six predefined Visual Cafe shapes; [Rect](#), [Circle](#), [Square](#), [Ellipse](#) , [VerticalLine](#) , and [HorizontalLine](#) . The component use a variety of property settings to generate the different effects.

 See Also

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A simple top-level window that has no border, no title, and no menubar. It behaves modally, blocking input to other windows when visible.

When you add a Window component to a project, a new Form Designer window opens. You can add other components to the Window by dragging components and containers to the Form Designer as necessary.

Visual Cafe provides frames, dialogs and other window subclasses for you. You must use a Frame component to create an application. You must use the Applet component to create an applet program.

### Runtime modification

Use the [show](#) method to make the Window visible at run time.

### Coding the component

To create a Window in project source code, use the following syntax:

```
window1 = new java.awt.Window(Frame parent);
```

Part	Type	Description
<i>window1</i>	Window	The variable name used to reference the component in code.
<i>parent</i>	Frame	The Frame that is the parent of the Window. A Window requires a parent <a href="#">Frame</a> in order to be displayed.

## Window Properties

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

The Wizard component simplifies the task of creating a “wizard dialog.” A wizard dialog leads the user through a sequence of steps needed to accomplish some task.

Follow these common steps to create your wizard:

- Create a new Frame or Dialog (typically a Dialog).
- Drop a Wizard component on the Dialog.
- Drop components on the Wizard. Typically you drop Panels on the upper part of the Wizard. Each of these panels would contain and organize a page of components. You can also drop components onto the button bar.
- Each time a component is dropped it is shown and becomes the active page. The state of the Back and Next buttons are updated. It is possible to use the Back and Next buttons to change the current page while designing your Wizard.
- Pages can be moved and removed in the Project window.

At run-time, the Wizard’s behavior can be extensively customized. See the [Wizard’s API documentation](#) for more information.

### Properties

Use the [Buttons Alignment property](#) to align the wizard’s buttons left, center, or right.

Use the [Combined Button](#) to combine the Next and Finish buttons into a single button.

Use the [Help Button Visible](#) to hide the Help button so that it can’t be used.

## Wizard Properties

[Background](#)

[Bounds](#)

[Buttons Alignment](#)

[Class](#)

[Combined Button](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Help Button Visible](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A multi-line text label. Text is displayed on one or more lines, wrapping as needed to fit in the available horizontal space.

Use a WrappingLabel when you want to display a paragraph of static (not user-editable) text.

An application or applet can change the label text string using the [setText](#) method, but a user cannot edit it.

### Properties

To specify the displayed text, type the text in the [Text property](#).

To specify how the text is justified within the bounds of the wrapping label, use the [Alignment Style property](#).

Use the Property List window to specify a font, color, and point size.

### Example

See the [SlideShow component](#) example.

 See Also

**Tip:** Since components do not recognize an initialization event, to display the first slide's description in the WrappingLabel, the first text string was entered in the label's Text property as the default text.

## WrappingLabel Properties

[Alignment Style](#)

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Inherit Background](#)

[Inherit Font](#)

[Inherit Foreground](#)

[Name](#)

[Text](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A FormattedTextField in which your user can type text that is limited to a zip code number format (five digit number). Text formatting logic is applied to the user input. If the text box already contains text, the user can select the default text and delete or edit it.

Use the Interaction Wizard to trigger an action when the value of this component is changed.

### Properties

The mask format is unchangeable, and set to 99999. See: [Mask property](#).

Use the [Editable property](#) to prevent the user from being able to edit the number.

## Change history

3/17/97 - deleted MaxRows, Sharable, And ReadOnly per Carl T.

8/6/97 - Update/check all properties for 1.1

12/04/97 - Added properties for JDBC components

4/28/98 - seperated dbAWARE and JDBC properties from standard into separate file

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also


 See Also

 See Also


 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Properties are those defaults and resources which define how a visual [component](#) appears and behaves when it first displays. All visual components inherit properties defined in the Component abstract [class](#). [Containers](#) also inherit properties defined in the container abstract class. Most visual components either override inherited properties, or add unique properties to their properties list.

You can view and set properties for a component from the Property List window. At runtime, most property values can be set or modified with the `set()` and `get()` methods.

For information on a specific property

- highlight the property name in the Property List and press F1
- access the component's help topic, click on the Properties button, and select a property

## General Properties

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The command name of the action event fired by this component.

**Runtime modification**

Use the `setActionCommand` method to modify this property.

**Settings**

Any string is valid. The default for the Button component is “button”.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The zero-relative index of the subpanel and tab to show.

#### Runtime modification

Use the `setCurrentPanelNdx` method to modify this property.

#### Settings

Valid Values	Description
-1	(Default)
integer	The zero-relative index of the subpanel and tab to show.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Positioning property. Controls the justification of text drawn in a Label component, and positions components within a container using the FlowLayout manager.

#### Settings

Valid values	Description
CENTER	(Default for Containers) The text/component should be centered.
LEFT	(Default for Labels) The text/component should be placed towards the left.
RIGHT	The text/component should be placed towards the right.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the justification of text drawn in a component.

#### Runtime modification

At run time, use the `setAlignStyle` method to modify this property.

#### Settings

Valid values	Description
ALIGN_CENTERED	Center text within the component.
ALIGN_LEFT	(Default) Justify the text to the left edge of the component.
ALIGN_RIGHT	Justify the text to the right edge of the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the moving of a split in a splitter panel will be in real time, or delayed until mouse released.

#### Runtime modification

At run time, use the `setAllowDynamicMoving` method to modify this property.

#### Settings

Valid values	Description
true	When a split is moved it will dynamically resize the panels the split affects.
false	(Default) When a split is moved the resizing will happen when the mouse is released.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether the MultiList is sorted when a column heading is clicked.

#### Settings

Valid values	Description
true	(Default) Clicking on a column heading sorts the MultiList.
false	Clicking on a column heading does not sort the MultiList.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [Grid Bag Constraints](#) . It is only accessible when the component is inside a container that uses a GridBagLayout layout. To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Determines where to place the component within a component's display area The system invokes this property when a component is smaller than its display area.

#### Runtime modification

At run time, use the `setConstraints` method to modify this property.

#### Settings

Constant	Value	Description
CENTER	10	Puts the component in the center of its display area.
EAST	13	Puts the component on the right side of its display area, centered vertically.
NORTH	11	Puts the component in the top of its display area, centered horizontally
NORTHEAST	12	Puts the component in the top right corner of its display area.
NORTHWEST	18	Puts the component in the top left corner of its display area.
SOUTH	15	Puts the component in the bottom of its display area, centered horizontally
SOUTHEAST	14	Puts the component in the bottom right corner of its display area.
SOUTHWEST	16	Puts the component in the bottom left corner of its display area.
WEST	17	Puts the component on the left side of its display area, centered vertically.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the color of the DirectionButton's arrow.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

Use the `setArrowColor` method to change the arrow color at run time.

#### Settings

Valid values	Description
black	(Default) Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the amount of blank space between the arrow and the [DirectionButton](#) border. This margin around the arrow is specified in pixels.

Sets the amount of blank space between the arrow and the button border in pixels.

**Settings**

Any non-negative integer. This integer specifies the margin, in pixels, around the arrow. The default value is zero (0), in which case the arrow takes up the entire button area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the StatusScroller's message will automatically start scrolling when the applet is loaded.

#### Settings

Valid values	Description
true	(Default) The scrolling will automatically start when the applet is loaded.
false	The scrolling will not automatically start when the applet is loaded.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the color displayed for the background of a component. This color is only used when the Inherit Background property is false. Choosing a background color automatically sets that property to false.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

All visual components and containers use the Background property.

#### Runtime modification

At run time you can change background or foreground colors using the component's `setBackground` or `setForeground` method.

#### Settings

Valid values	Description
black	Makes the component black.
blue	Makes the component blue.
cyan	Makes the component cyan.
darkGray	Makes the component dark gray.
gray	Makes the component gray.
green	Makes the component green.
lightGray	Makes the component light gray.
magenta	Makes the component magenta.
orange	Makes the component orange.
pink	Makes the component pink.
red	Makes the component red.
white	(Default) Makes the component white.
yellow	Makes the component yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the height of the shadow on any component that derives from Border Panel.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the border bevel style for the component.

#### Runtime modification

At run time, use the `setBevelStyle` method to modify this property.

#### Settings

Valid values	Description
BEVEL_LINE	(Default) Uses a single line border. This provides a "flat" appearance. Flat appearance should be used for a scrollable region or control, such as a drop-down list and drop-down combo box.
BEVEL_LOWERED	3D effect. Makes the component appear lower than the surrounding area. This field style is standard for text boxes, check boxes, spin boxes, and list boxes.
BEVEL_NONE	No border is used.
BEVEL_RAISED	3D effect. Makes the component appear higher than the surrounding area. This window style is used for primary and secondary windows.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the large increment of a scroll bar. This is the amount that is added to or subtracted from the component's Value property setting when the user clicks the area between the scroll thumb and scroll arrow.

#### Runtime modification

Use the `setBlockIncrement` method to modify this property.

#### Settings

Valid Values	Description
10	(Default)
0-32767	Valid values are within this range.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color for a border around the component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	(Default) Makes the border black.
blue	Makes the border blue.
cyan	Makes the border cyan.
darkGray	Makes the border dark gray.
gray	Makes the border gray.
green	Makes the border green.
lightGray	Makes the border light gray.
magenta	Makes the border magenta.
orange	Makes the border orange.
pink	Makes the border pink.
red	Makes the border red.
white	Makes the border white.
yellow	Makes the border yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Property indicating the drawing margin, in pixels, around the outside of a BEVEL\_LINE border.

#### Settings

Valid values	Description
INDENT_ZERO	(Default) Specifies a drawing margin of 0 pixels around the outside of a BEVEL_LINE border
INDENT_ONE	Specifies a drawing margin of 1 pixel around the outside of a BEVEL_LINE border.
INDENT_TWO	Specifies a drawing margin of 2 pixels around the outside of a BEVEL_LINE border.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the setting or suppressing of the component's border.

#### Runtime modification

At run time, use the `setBevelStyle` method to modify this property.

#### Settings

Valid values	Description
BEVEL_LINE	(Default) Uses a single line border. This provides a "flat" appearance. Flat appearance should be used for a scrollable region or control, such as a drop-down list and drop-down combo box.
BEVEL_LOWERED	3D effect. Makes the component appear lower than the surrounding area. This field style is standard for text boxes, check boxes, spin boxes, and list boxes.
BEVEL_NONE	No border is used.
BEVEL_RAISED	3D effect. Makes the component appear higher than the surrounding area. This window style is used for primary and secondary windows.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the setting or suppressing of the ImageListBox's component's border.

#### Settings

Valid values	Description
BORDER_REGULAR	(Default) The component displays with a lowered border, making the component appear lower than the surrounding area.
BORDER_NONE	The component displays with no border.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color for a component's border.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	(Default) Makes the border black.
blue	Makes the border blue.
cyan	Makes the border cyan.
darkGray	Makes the border dark gray.
gray	Makes the border gray.
green	Makes the border green.
lightGray	Makes the border light gray.
magenta	Makes the border magenta.
orange	Makes the border orange.
pink	Makes the border pink.
red	Makes the border red.
white	Makes the border white.
yellow	Makes the border yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the position and size of a component, in pixels. All visual components and containers support the Bounds properties.

#### Runtime modification

Change the dimensions of a component by calling the `setBounds` method.

---

#### X Property

Sets the distance between the left edge of a component and the left edge of its container.

##### Valid values

	Description
0	(Default) The left-most edge of the container.

#### Y Property

Sets the distance between the top of a component and the top of its container.

Valid values	Description
0	(Default) The top-most edge of the container.

#### Width Property

Sets the distance of a component's right edge from its left edge.

Valid values	Description
The default value varies.	

#### Height Property

Sets the distance of a component's bottom edge from its top edge.

Valid values	Description
The default value varies.	

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



If the Draw as Boxes property is true, Box Width specifies the width, in pixels, of the discrete boxes that make up the indicator of a ProgressBar component. If the Draw as Boxes property is false, the Box Width property is ignored.

#### Settings

Valid values	Description
8	(Default) A box width of 8 pixels.
1-32767	The desired box width in pixels.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color of the button part of the component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	Makes the border black.
blue	Makes the border blue.
cyan	Makes the border cyan.
darkGray	Makes the border dark gray.
gray	Makes the border gray.
green	Makes the border green.
lightGray	(Default) Makes the border light gray.
magenta	Makes the border magenta.
orange	Makes the border orange.
pink	Makes the border pink.
red	Makes the border red.
white	Makes the border white.
yellow	Makes the border yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies how the Wizard's button will be aligned.

#### Runtime modification

Use the `setButtonsAlignment` method to modify this property.

#### Settings

Valid values	Description
LEFT	Aligns the buttons towards the left.
CENTER	Centers the buttons.
RIGHT	(Default) Aligns the buttons towards the right.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the name of the card in the layout.

**Settings**

The default value is None.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether searches within a component are case sensitive.

#### Settings

Valid values	Description
false	(Default) Searches are not case sensitive.
true	Searches are case sensitive.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the color to use for the background of all cells in the MultiList component.


To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	(Default) Makes the cell background white.
yellow	Makes the cell background yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines the font used to display text within all cells in the MultiList component. In the property field, click on the  button (or double-click in the field) to display the font dialog box. Select the font combination that you want.

#### **Runtime modification**

At run time, use the `setCellFont` method to modify this property.

#### **Settings**

The default value is

```
"SansSerif",java.awt.Font.PLAIN,12
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the color to use for all cell text in the MultiList component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	(Default) Makes the cell foreground black.
blue	Makes the cell foreground blue.
cyan	Makes the cell foreground cyan.
darkGray	Makes the cell foreground dark gray.
gray	Makes the cell foreground gray.
green	Makes the cell foreground green.
lightGray	Makes the cell foreground light gray.
magenta	Makes the cell foreground magenta.
orange	Makes the cell foreground orange.
pink	Makes the cell foreground pink.
red	Makes the cell foreground red.
white	Makes the cell foreground white.
yellow	Makes the cell foreground yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines where within the component area to draw the image.

#### Settings

Valid values	Description
true	(Default) Center the image.
false	Image is left-justified and top justified within the component area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether a menu item behaves like a check box. A Checkbox toggles between true and false when selected.

#### Settings

Valid values	Description
true	(Default) The menu behaves like a check box. When selected, the state of the checkbox toggles from checked to unchecked, and from unchecked to checked.
false	The menu item does not behave like a check box.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The fully qualified name of the Java [package](#) and [class](#) defining the current [component](#). All visual components and [containers](#) use this property.

### **Settings**

Read-only.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether the previous image is cleared before displaying the current image. If this property is set to true, the end user may see a flicker as the image changes.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



A list of strings that specifies how the text in each column of a MultiList is justified. The columns are listed from left to right.

To create an list item, type the item value within the list box. To create a new list item, press CTRL-ENTER and type the next item value. (The Macintosh version provides a button.)

#### Runtime modification

At run time, use the `setColumnAlignment` or `setColumnAlignments` methods to modify this property.

#### Settings

Valid values	Description
[empty]	The component does not have a list of values for this property.
[list]	The component has a list of values for this property.

Valid list values	Description
Left	Align the text to the left of the cell.
Center	Align the text in the center of the cell.
Right	Align the text to the right of the cell.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A list of strings that specifies a text heading for each column. List the columns from left to right. The headings are clipped if necessary.

To create an list item, type the item value within the list box. To create a new list item, press CTRL-ENTER and type the next item value. (The Macintosh version provides a button.)

#### Runtime modification

At run time, use the `setHeading` or `setHeadings` methods to modify this property.

#### Settings

Valid values	Description
[empty]	The component does not have a list of values for this property.
[list]	The component has a list of values for this property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



A list of strings that specifies the width of each column in pixels. List the columns from left to right.

To create an list item, type the item value within the list box. To create a new list item, press CTRL-ENTER and type the next item value. (The Macintosh version provides a button.)

#### Runtime modification

At run time, use the `setColumnSizes` method to modify this property.

#### Settings

Valid values	Description
[empty]	The component does not have a list of values for this property.
[list]	The component has a list of values for this property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the number of columns in a text component. Each column holds a single character. When both the row and column values are greater than zero, the `getMinimumSize` method returns a size appropriate for the desired number of rows and columns.

#### Settings

Valid values	Description
0	(Default) Don't use when determining minimum component size.
1-32767	The desired number of columns to display.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of character columns to display in a list. This affects the dimensions returned by the `getPreferredSize` method and determines the number of columns displayed when this component is automatically laid out.

#### Runtime modification

Use the `setColumns` method to modify this property.

#### Settings

Valid values	Description
0	(Default) Don't use when determining minimum component size.
1-32767	The desired number of character columns to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether to combine the Next and the Finish buttons into a single button.

#### Runtime modification

Use the `setCombinedButton` method to modify this property.

#### Settings

Valid values	Description
true	The Next and Finish buttons are combined into one.
false	(Default) The Next and Finish buttons are separate.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This property determines the highlight action of the ImageListBox. When ComboBox Mode is true list selections will be highlighted anytime the mouse cursor is over them, even when the mouse button is not down. When false, the mouse needs to be pressed to select an item.

This property only applies when the ImageListBox is not allowing multiple selections.

#### Runtime modification

At run time, use the `setComboBoxMode` method to modify this property.

#### Settings

Valid values	Description
true	Items are selected and highlighted when the mouse cursor is over them.
false	(Default) Items are selected and highlighted only when the mouse button is pressed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The cursor to use while the mouse is over this component.

#### Runtime modification

Use the `setCursor` method to modify this property.

#### Settings

Valid values	Description
CROSSHAIR_CURSOR	A crosshair cursor.
DEFAULT_CURSOR	(Default) The default cursor.
E_RESIZE_CURSOR	An east-resize cursor.
HAND_CURSOR	A hand cursor.
MOVE_CURSOR	A move cursor.
N_RESIZE_CURSOR	A north-resize cursor.
NE_RESIZE_CURSOR	A north-east-resize cursor.
NW_RESIZE_CURSOR	A north-west-resize cursor.
S_RESIZE_CURSOR	A south-resize cursor.
SE_RESIZE_CURSOR	A south-east-resize cursor.
SW_RESIZE_CURSOR	A south-west-resize cursor.
TEXT_CURSOR	A text cursor.
W_RESIZE_CURSOR	A west-resize cursor.
WAIT_CURSOR	A wait cursor.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the starting date for the Calendar component. Most standard date syntaxes are recognized, including the IETF standard date syntax: "Mon, 17 Jul 1997 14:45:00 GMT", and also "7/17/97".

#### **Runtime modification**

At run time, use the `setDate` method to modify this property.

#### **Settings**

A string representing the desired date.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The properties used to control date mask formatting.

## Include Literals

Determines whether date and time separators should be used when inputting and displaying text. .

### Runtime modification

At runtime, use the `setIncludeLiterals` method to modify this property.

### Settings

Valid values	Description
true	(Default) The literal characters such as "/" and "-" are included.
false	The literal characters such as "/" and "-" are not included.

## Mask

Shows the mask the date or time uses in the field.

### Settings

This is a read only property.

## Date Pattern

Shows the pattern used to display the date or time.

### Settings

This is a read only property.

## Formatting Style

Determines how the date or time is displayed in the field. Formats are originally specified in the `java.Text.DateFormat` class.

### Runtime modification

At runtime, use the `setFormattingStyle` method to modify this property.

### Settings

Valid values	Description
SHORT	completely numeric, such as 12.13.52 or 3:30pm.
DEFAULT	(Default) longer, such as Jan 12, 1952.
MEDIUM	longer, such as Jan 12, 1952.
LONG	longer, such as January 12, 1952 or 3:30:32pm.
FULL	completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST.

## Type

Sets the datemask to an equivalent SQL date or time type.

### Runtime modification

At runtime, use the `setType` method to modify this property.

### Settings

Valid values	Description
DATE_TYPE	(Default) The equivalent to the SQL Date type.
TIME_TYPE	The equivalent to the SQL Time type.
TIMESTAMP_TYPE	The equivalent to the SQL Timestamp type.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the initial file directory that a file dialog box opens to.

**Runtime modification**

At run time, use the `setDirectory` method to modify this property.

**Settings**

Any valid operating system directory.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the default file name that a file dialog box opens with.

**Runtime modification**

At run time, use the `setFile` method to modify this property.

**Settings**

Any valid operating system file name.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The delay, in milliseconds, between animation frames (for `Animator` and `MovingAnimation`) or message scroll steps (for `StatusScroller`).

#### Runtime modification

At run time, use the `setDelay` method to modify this property.

#### Settings

Valid Values	Description
100	(Default for <code>StatusScroller</code> )
500	(Default for <code>Animator</code> and <code>MovingAnimation</code> )
integer	The delay in milliseconds.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Use this property to define the direction in which the [DirectionButton component](#) arrow points.

#### Runtime modification

At run time, use the `setDate` method to modify this property.

#### Settings

Valid values	Description
TOP	Arrow points up.
RIGHT	Arrow points to the right edge of the container.
BOTTOM	Arrow points down.
LEFT	Arrow points to the left edge of the container.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Toggles the display of cell borders within the ImageListBox component.

#### Runtime modification

At run time, use the `setCellBorders` method to modify this property.

#### Settings

Valid values	Description
true	Cell borders display.
false	(Default) Cell borders do not display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Used by the ProgressBar component to determine how to draw the progress indicator. The indicator may be drawn as a single solid box, or as a series of discrete boxes.

#### Settings

Valid values	Description
true	Progress indicator is drawn as a series of discrete boxes.
false	(Default) Progress indicator is drawn as a single box.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Allows resizing of the edit field, if needed, when new values are entered.

#### Settings

Valid values	Description
true	The edit field will resize as needed.
false	(Default) The edit field will not resize.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



When this property is not empty, the character specified appears in the text component as the user types text. This property is useful for password fields, for example, where you don't want the characters a user enters to display on the screen.

#### Runtime modification

At run time, use the `setEchoChar` method to modify this property.

#### Settings

You can use any single alphanumeric character as the echo character.

Valid values	Description
empty	(Default) Turn echoing off.
non-empty	Echo specified character.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies whether the user can edit selected text on a list in a text-based component.

#### **Runtime modification**

You can change this property at run time using the `setEditable` method.

#### **Settings**

<b>Valid values</b>	<b>Description</b>
true	The user can edit text in the component.
false	The text in the component is read only.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Enables or disables a component.

#### Runtime modification

At run time, use the `setEnabled` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The component can recognize events when it receives focus.
false	The component will not recognize events.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Prevents the dragging of a split between panels from making a panel smaller than the minimum size of its component.

#### Runtime modification

At run time, use the `setEnforceMinDim` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Dragging a split between panels may make the panel smaller than the minimum size of its component.
true	Dragging a split between panels may never make the panel smaller than the minimum size of its component.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines this file dialog's filename filter. A filename filter allows the user to specify which files appear in the file dialog window.

**Runtime modification**

At run time, use the `setFilenameFilter` method to modify this property.

**Settings**

The default setting is None.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color that fills the component when the Fill Mode property is true.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

At run time, use the `setFillColor` method to modify this property.

#### Settings

Valid values	Description
black	(Default) Makes the enabled text black.
blue	Makes the enabled text blue.
cyan	Makes the enabled text cyan.
darkGray	Makes the enabled text dark gray.
gray	Makes the enabled text gray.
green	Makes the enabled text green.
lightGray	Makes the enabled text light gray.
magenta	Makes the enabled text magenta.
orange	Makes the enabled text orange.
pink	Makes the enabled text pink.
red	Makes the enabled text red.
white	Makes the enabled text white.
yellow	Makes the enabled text yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether a shape component is automatically filled with the color specified by its Fill Color property.

#### Runtime modification

At run time, use the `setFillMode` method to modify this property.

#### Settings

Valid values	Description
true	The component is filled with the color defined in the Fill Color property.
false	(Default) The component is not filled with a color.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [Grid Bag Constraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Enabled when the component's display area is larger than the component's requested size. It determines whether (and how) to resize the component, and resizes the component according to the settings listed below.

This property is valid only for components placed within containers using the GridBagLayout layout manager.

#### Runtime modification

At run time, use the `setConstraints` method to modify this property.

#### Settings

Valid values	Numeric	Description
NONE	0(Default)	The component is not resized.
BOTH	1	Makes the component fill the display area, vertically and horizontally.
HORIZONTAL	2	Makes the component wide enough to fill its display area horizontally. The component's height is unchanged.
VERTICAL	3	Makes the component tall enough to fill its display area vertically. Does not affect the width of the component.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the font for a component's text. This property is broken into the subproperties that are accessed from the Font property dropdown list box.

This font is only used when the Inherit Font property is false. Choosing a font automatically sets that property to false.

All visual components and containers support this property.

Customized settings display as follows in the INIT\_CONTROL code block:

```
comboBox1.setFont(new Font("TimesRoman", Font.ITALIC, 14));
```

### Runtime modification

At run time, the `setFont` method is typically used to modify this property.

---

#### Name

Specifies the name of the font displaying text within a component. The pulldown menu lists available fonts. To set the current font to match the parent font, change the Inherit Font property to `true`.

#### Size property

Sets the font size in points.

#### Valid values

	Description
12	(Default) 12 point font.
1-32767	Valid points sizes are within this range.

#### Style:Bold

Sets component text to bold.

#### Valid values

	Description
true	Set component text to bold.
false	(Default) Sets component text to not bold.

#### Style:Italic

Sets component text to italicized.

Valid values	Description
true	Set component text to italic.
false	(Default) Sets component text to not italic.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the color displayed for the text in a component. This color is only used when the Inherit Foreground property is false. Choosing a foreground color automatically sets that property to false.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

All visual components and containers support this property.

#### Runtime modification

At run time you can change background or foreground colors using the component's `setBackground` or `setForeground` method.

#### Settings

Valid values	Description
black	(Default) Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the frame used to show a URL document in a browser or applet viewer.

#### Runtime modification

At run time, use the `setFrame` method to modify this property.

#### Settings

Valid values	Description
empty	(Default) The name is blank. Equivalent to “_self”.
"_self"	show document in the current frame
"_parent"	show document in the parent frame
"_top"	show document in the topmost frame
"_blank"	show document in a new unnamed toplevel window
all others	show document in a new toplevel window with the given name

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the display color for the [SplitterPanel component](#)'s gap between subpanels.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

At run time, use the `setGapColor` method to modify this property.

#### Settings

Valid values	Description
black	Makes the gap black.
blue	Makes the gap blue.
cyan	Makes the gap cyan.
darkGray	Makes the gap dark gray.
gray	Makes the gap gray.
green	Makes the gap green.
lightGray	(Default) Makes the gap light gray.
magenta	Makes the gap magenta.
orange	Makes the gap orange.
pink	Makes the gap pink.
red	Makes the gap red.
white	Makes the gap white.
yellow	Makes the gap yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The distance, in pixels, to space between boxes drawn within the ProgressBar component.

This property is only used for ProgressBars with a false Box Bar Style property value.

**Runtime modification**

At run time, use the `setGapWidth` method to modify this property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines how components are positioned inside a container that uses a GridBagLayout layout manager. These properties are not available unless the [Layout property](#) for the component's container is set to GridBagLayout.

The Grid Bag Constraints properties apply to visual components, but not their container. However, their container must have its Layout property set to GridBagLayout.

Grid Bag Constraints 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 separate Grid Bag Constraints properties values. 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.

The properties for components within a grid bag layout are:

[Anchor](#)

[Fill](#)

[Grid](#)

[Insets](#)

[Internal Pad X](#)

[Internal Pad Y](#)

[Weight X](#)

[Weight Y](#)

#### **Runtime modification**

At run time, use the `setConstraints` method to modify this property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [Grid Bag Constraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

These properties specify the characteristics of the component's display area within a grid bag layout.

#### Runtime modification

At run time, use the `setConstraints` method to modify this property.

---

#### Grid X property

Sets the cell where a component's display area begins. The leftmost cell has *gridx* = 0.

#### Grid Y property

Specifies the cell number at the top of the component's display area. The topmost cell has *gridy* = 0.

#### Grid Width property

Specifies the number of cells in a row to use for the component's display area.

#### Grid Height property

Specifies the number of cells in a column to use for the component's display area.

Use the value `GridBagConstraints.REMAINDER` to specify that the component be the last one in its column. Use the value `GridBagConstraints.RELATIVE` to specify that the component be the next to last one in its column.

#### Settings

The settings for Grid X are:

Valid values	Description
-1	(Default) Places the component immediately to the right of the component that was most recently added to the container.
x	An integer expression specifying the cell number where the left side of a component's display area begins.

The settings for Grid Y are:

Valid values	Description
-1	(Default) Places the component below the component that was added to the container was most recently added to the container.
y	An integer expression specifying the cell number where the top of a component's display area begins.

The settings for Grid Width are:

Valid values	Description
1	(Default) Specifies that the component use the space remaining in its column for its display area.
w	An integer expression that specifies the width of the display area, in columns.

The settings for Grid Height are:

Valid values	Description
1	(Default) Specifies that the component use the last cell in its column as its display area.
h	An integer expression that specifies the width of the display area, in rows.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the name for a set of Checkbox components to be treated as a group or set. If the check box group does not already exist, Visual Café creates a group with the specified name.

When the Group property is specified, the Checkbox behaves like a radio button. Only one of the components in the check box group can be "on" at a time.

#### **Runtime modification**

At run time, use the `setCheckboxGroup` method to modify this property.

#### **Settings**

The value of the Group property must be a valid variable name.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the color to use for the background of all cell headings in the component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

At run time, use the `setHeadingBg` or `setHeadingColors` method to modify this property.


#### Settings

Valid values	Description
black	Makes the heading background black.
blue	Makes the heading background blue.
cyan	Makes the heading background cyan.
darkGray	Makes the heading background dark gray.
gray	Makes the heading background gray.
green	Makes the heading background green.
lightGray	(Default) Makes the heading background light gray.
magenta	Makes the heading background magenta.
orange	Makes the heading background orange.
pink	Makes the heading background pink.
red	Makes the heading background red.
white	Makes the heading background white.
yellow	Makes the heading background yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines the font used to display all cell heading text within the component.

In the property field, click on the  button (or double-click in the field) to display the font dialog box. Select the font combination that you want.

#### **Runtime modification**

At run time, use the `setHeadingFont` method to modify this property.

#### **Settings**

The default value is

```
"SansSerif",java.awt.Font.PLAIN,12
```

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the color to use for all cell heading text in the component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At run time, use the `setHeadingFg` or `setHeadingColors` method to modify this property.

### Settings

Valid values	Description
black	(Default) Makes the enabled text black.
blue	Makes the enabled text blue.
cyan	Makes the enabled text cyan.
darkGray	Makes the enabled text dark gray.
gray	Makes the enabled text gray.
green	Makes the enabled text green.
lightGray	Makes the enabled text light gray.
magenta	Makes the enabled text magenta.
orange	Makes the enabled text orange.
pink	Makes the enabled text pink.
red	Makes the enabled text red.
white	Makes the enabled text white.
yellow	Makes the enabled text yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the Help button is shown in the Wizard.

#### Runtime modification

Use the `setHelpButtonVisible` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The Help button is visible.
false	The Help butto is not visible.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies if the Menu component is the Help menu.

#### Runtime modification

At run time, use the `setHelpMenu` method to modify this property.

#### Settings


Valid values	Description
true	The menu component is the Help menu.
false	(Default) The menu component is not the Help menu.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The URL of the document the component links to.

To define the URL:

1. Click in the HTML Link URL field in the Property List.
2. Click on the  button (or double-click in the field) to display the HTML Link URL dialog box.
3. Enter the name of the target HTML file.

#### **Runtime modification**

At run time, use the `setLinkURL` method to modify this property.

#### **Settings**

Valid values	Description
empty	(Default)
URL	A valid URL.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies how to display the image inside the component.

#### Runtime modification

Use the `setStyle` method (for `ImagePanels` and `ImageViewers`) or `setImageStyle` method (for `ImageButtons`) to modify this property.

#### Settings


Valid values	Description
IMAGE_TILED	The image is drawn repeatedly, filling the component.
IMAGE_CENTERED	(Default) The image is drawn once, centered in the component.
IMAGE_SCALED_TO_FIT	The image is drawn once, resized to fill up the entire component.
IMAGE_NORMAL	(for <code>ImageButtons</code> ) The image is drawn once in the upper-left corner of the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The URL of the image to display in the component. The image should be in GIF or JPEG format.

To specify the URL:

1. Click in the Image URL field in the Property List.
2. Click on the  button (or double-click in the field) to display the Image URL dialog box.
3. Enter the name of the target HTML file.

#### Runtime modification

Use the `setImageUrl` method to modify this property.

#### Settings

Valid values	Description
empty	(Default) No image is displayed.
URL	The URL of a GIF or JPEG image to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the amount that the component value is incremented, plus or minus, when the user presses the component's up or down arrow.

#### Runtime modification

Use the `setIncrement` method to modify this property.

#### Settings

Valid values	Description
1	(Default) The component value is incremented by one.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines where a component gets its background color setting from. When true the component uses the background color of its parent container. When false the component uses the color specified by the Background property.

All components and containers support this property.

#### Runtime modification

At run time, use the `setBackground` method to modify this property. Passing a null value will make the component get its background from its parent container.

#### Settings

Valid values	Description
true	(Default) The component uses the background color in its parent container's <a href="#">Background property</a> .
false	The component uses its own <a href="#">Background property</a> .

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines where a component gets its font settings from . When true the component uses the font of its parent component. When false the component uses the font specified by the Font property.

All visual components and [containers](#) support this property.

#### Runtime modification

At run time, use the `setFont` method to modify this property. Passing a null value will make the component get its font from its parent component.

#### Settings

Valid values	Description
true	(Default) The component uses the font set in its parent component's Font property.
false	The component uses its own Font property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines where a component gets its foreground color setting from. When true the component uses the foreground color of its parent container. When false the component uses the color specified by the Foreground property.

All components and containers support this property.

#### Runtime modification

At run time, use the `setForeground` method to modify this property. Passing a null value will make the component get its foreground from its parent container.

#### Settings

Valid values	Description
true	(Default) The component uses the foreground color in its parent container's <a href="#">Foreground property</a> .
false	The component uses its own <a href="#">Foreground property</a> .

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the inset padding, in pixels, for a component. This is the distance between the drawn border and the usable display area within the border. It is used to determine the size of the panel contained within this component's borders.

#### Runtime modification

At run time, use the `setIPad...` methods to modify this property.

#### Settings

Valid values	Description
0	Specifies that there is no space between the edge of a component and the edge of its display area.
n	An integer expression that sets the number of pixels.

---

#### Inset Padding Bottom

Sets the number of pixels between the bottom of the drawn border and the display area within that border. The default value is 7.

#### Inset Padding Left

Sets the number of pixels between the left side of the drawn border and the display area within that border. The default value is 4.

#### Inset Padding Right

Sets the number of pixels between the right side of the drawn border and the display area within that border. The default value is 4.

#### Inset Padding Top

Sets the number of pixels between the top of the drawn border and the display area within that border. The default value is 2.

#### Inset Padding Sides (BorderPanel component)

Sets the number of pixels between the sides of the drawn border and the display area within that border. The default value is 4.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [Grid Bag Constraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Specifies the amount of space in pixels, to add to the minimum size of the component to determine its overall width. This property is inaccessible unless a component is inside a container which uses a grid bag layout.

#### Runtime modification

At run time, use the `setConstraints` method to modify this property.

#### Settings

Valid values	Description
0 (zero)	(Default) Specifies that the component's display at its minimum width.
n	An integer expression that specifies the number of additional pixels to add to the minimum width of the component. Note, the width of the component is at least its minimum width in pixels plus twice its Internal Pad X property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [Grid Bag Constraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Specifies the amount of space to add to the minimum size of the component to determine its overall height. This property is inaccessible unless a components is inside a container which uses a gridbag layout.

#### Runtime modification

At run time, use the `setConstraints` method to modify this property.

#### Settings

Valid values	Description
0 (zero)	(Default) Specifies that the component's display at its minimum height.
n	An integer expression that specifies the number of additional pixels to add to the minimum height of the component. Note, the width of the component is at least its minimum height in pixels plus twice its Internal Pad Y property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A list of strings that populate the list portion of a component.

To create an list item, type the item value within the list box. To create a new list item, press CTRL-ENTER and type the next item value. (The Macintosh version provides a button.)

By default, the first item added to the choice menu typically becomes the selected item, until the user makes a different selection, or your program calls one of the select methods.

### Runtime modification

At run time, use the `addItem` methods (or `append` for the `TreeView` component) to modify this property.

### Settings

Valid values	Description
[empty]	No items are in the list.
[list]	The list contains some items.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the alignment of text in the label area.

#### Runtime modification

At run time, use the `setAlignStyle` method to modify this property.

#### Settings

Valid values	Description
ALIGN_CENTERED	(Default) Center text within the label area.
ALIGN_LEFT	Justify the text to the left edge of the label area.
ALIGN_RIGHT	Justify the text to the right edge of the label area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color for a label (Label property) appearing within the component.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

At run time, use the `setLabelColor` method to modify this property.

#### Settings

Valid values	Description
black	(Default) Makes the enabled text black.
blue	Makes the enabled text blue.
cyan	Makes the enabled text cyan.
darkGray	Makes the enabled text dark gray.
gray	Makes the enabled text gray.
green	Makes the enabled text green.
lightGray	Makes the enabled text light gray.
magenta	Makes the enabled text magenta.
orange	Makes the enabled text orange.
pink	Makes the enabled text pink.
red	Makes the enabled text red.
white	Makes the enabled text white.
yellow	Makes the enabled text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the text that appears in a component as its label.

**Runtime modification**

At run time, use the `setLabel` method to modify this property.

**Settings**

The valid value for Label is any string expression including all alphanumeric characters.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the `LayoutManager` used to arrange components within a container.

A container is a component that contains other components. In Java, the `LayoutManager` is an interface for an object that positions components within a container. Java provides five predefined `LayoutManagers`, which you use by calling the container's `setLayout` method. In addition, you can write your own `LayoutManager` by writing a class that implements the `LayoutManager` interface.

Visual Cafe simplifies the layout process by creating a "Layout property" on the Property List window for your container. This allows you to select or change the `LayoutManager` with the click of a mouse button at any time during your GUI design process. You set the Layout property, and then drag and drop components onto the Forms Designer window.

At design time, when you change a container's Layout property, the properties of the container's child components change to allow you to specify how the component is arranged inside the container.

### Runtime modification

To use a `LayoutManager` in code:

1. Instantiate a Container object,
2. Instantiate the desired type of `LayoutManager`,
3. Use the container's `setLayout` method to specify the `LayoutManager` to be used,
4. Instantiate, configure, and add each component to the container in the desired order.

### Settings

Valid values	Description
None	Positions and sizes components within a container precisely where you place and size them.
BoxLayout (Swing Only)	This layout manager is similar to the Flow layout manager, but it can lay out components from left to right or from top to bottom.
BorderLayout	Positions components within a container using individual component placement. When you specify <code>BorderLayout</code> , the <a href="#">Placement property</a> is added to each component in the container. You can control component spacing by using the layout's Horizontal and Vertical Gap property in pixels.
CardLayout	Contains several 'cards'. Only one card is visible at a time, allowing you to flip through the cards. Each card supports its own layout manager. You can control component spacing by using the layout's Horizontal and Vertical Gap property in pixels.
FlowLayout	Arranges components left to right until no more components fit on the same line, and then moves to the next line. By default, each line is centered. You need to modify the component size and placement within the flow layout line. You can control component spacing by using the layout's Horizontal and Vertical Gap property in pixels. If you do not want the lines centered, use the layout's <a href="#">Alignment property</a> to specify the justification of each row of components in the container.
GridBagLayout	Aligns components vertically and horizontally, without requiring that the components be the same size. Each row and column in a <code>GridBagLayout</code> can expand to hold its largest component, and pack components to conserve space relative to bordering cells—which is similar to how table rows and columns expand to hold their largest cells. <code>GridBagLayout</code> manages a rectangular grid of cells, with each component occupying one or more cells. Components within the <code>GridBagLayout</code> are positioned based on their <a href="#">Grid Bag</a>

Constraints properties . You also need to carefully test GridBagConstraints properties for each component displaying in a GridBagLayout container to determine that each component expands, shrinks and repositions itself appropriately, relative to other components in the container.

**GridLayout** A layout manager that divides a container into rows and columns of uniform rectangular cells. The GridLayout can have a fixed number of Rows , or Columns , or both—but all cells maintain the same size. After placing a component within a cell, you need to adjust the component size and position.

You can control component spacing by using the layout's Horizontal and Vertical Gap property in pixels.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The width of the line in pixels.

#### Runtime modification

Use the `setLineThickness` method to modify this property.

#### Settings

Valid values	Description
1	(Default)
1-32767	Valid values are within this range.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies whether the ComboBox's list is initially visible or not.

#### Runtime modification

At run time, use the `setListDown` methods to modify this property.

#### Settings

Valid values	Description
false	(Default) The list is not initially visible.
true	The list is initially visible.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the items that populate the list portion of a component.

To create an list item, type the item value within the list box. To create a new list item, press CTRL-ENTER and type the next item value. (The Macintosh version provides a button.)

#### **Runtime modification**

At run time, use the `setListItems` methods to modify this property.

#### **Settings**

Valid values	Description
[empty]	No items are in the list.
[list]	The list contains some items.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the number of times to show an animated sequence. This value is ignored if the Repeat Mode property is set to true.

#### Runtime modification

At run time, use the `setNumLoops` method to modify this property.

#### Settings

Valid values	Description
1	Displays the sequence one time. Holds a display on the last image.
integer	Any whole number, shows the animation this number of times.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Controls the user input to the component.

The `FormattedTextField`, and derived components, use a set of predefined characters that define the mask. When defining a mask, be sure to preface any character that is to be a part of the mask with the Escape character (`'/'`), including spaces. For instance, specify the mask for a long distance telephone number as: `/(999)/ 999-9999`.

The following are valid characters that can be used for a mask:

<u>To restrict the digit position to:</u>	<u>Use:</u>
An escape	/
Digit	9
Sign (+ or -)	+
Sign or digit	-
Any upper case alphabetic character	A
Any lower case alphabetic character	a
Any alphabetic character, forced uppercase	U
Any alphabetic character, forced lowercase	L
Any alphabetic or digit, upper case	X
Any alphabetic or digit, lower case	x
Any alphabetic or digit, forced upper case	N
Any alphabetic or digit, forced lower case	n
Any character (wildcard)	*

The following components have predefined masks that cannot be edited:

<u>Component</u>	<u>Default mask</u>
<code>IntlLongDistPhoneNumber</code>	<code>999/-999/-999/-9999</code>
<code>LocalPhoneNumber</code>	<code>999/-9999</code>
<code>LongZipCode</code>	<code>99999/-9999</code>
<code>PostalCode</code>	<code>A9A/-9A9</code>
<code>SocialInsuranceNumber</code>	<code>999/-999/-999</code>
<code>SocialSecurityNumber</code>	<code>999/-99/-9999</code>
<code>USLongDistPhoneNumber</code>	<code>999/-999/-9999</code>
<code>ZipCode</code>	<code>99999</code>

### Runtime modification

At run time, the `FormattedTextField` component can use the `setMask` method to modify this property.

### Settings

<u>Valid values</u>	<u>Description</u>
empty	(Default) The default mask is used.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The maximum value a Slider component may have.

#### Runtime modification

Use the `setMaxValue` method to modify this property.

#### Settings

Valid values	Description
10	(Default) Maximum value is ten.
integer	The maximum value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Use these properties to specify the maximum and minimum value range for a spinner.

**Runtime modification**

Use the `setMax` or `setMin` method to modify this property.

Use the `getMax` or `getMin` method for the current value of this property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the maximum value of the scroll thumb when it is in its bottom or rightmost position of a scrollbar.

The scrollbar [component](#) supports this property.

#### Runtime modification

At run time, use the `setMaximum` method to modify this property.

#### Settings

Valid values	Description
0	(Default) The maximum scrollbar value is 0.
integer	The maximum scrollbar value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the menu bar to display in a Frame. You can create more than one menu bar in a frame, but only one Menu Bar can be displayed at a time.

**Runtime modification**

At run time, you can display a different menu bar by calling the Frame's `setMenuBar` method.

**Settings**

The Default setting is None.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies a shortcut key that may be used to invoke the MenuItem. This property consists of the subproperties Key Code and Use Shift Key, described below.

#### Runtime modification

At run time, use the `setShortcut` method to modify this property.

---

#### Key Code

The raw keycode that invokes this menu item. Choose the desired value from the drop-down list. This is the same type of value as the `keyCode` in the `KeyEvent` class.

#### Use Shift Key

Determines whether this shortcut requires the shift key pressed to invoke.

#### Valid values

Description	
false	(Default) Shift key not required.
true	Shift key required.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The highlighted color for message text. A text message is highlighted in this color when the mouse cursor is over it and there is a corresponding non-null URL link for that message.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

#### Runtime modification

Use the `setHiliteColor` method to modify this property.

#### Settings

Valid values	Description
black	Makes the highlighted message text black.
blue	Makes the highlighted message text blue.
cyan	Makes the highlighted message text cyan.
darkGray	Makes the highlighted message text dark gray.
gray	Makes the highlighted message text gray.
green	Makes the highlighted message text green.
lightGray	Makes the highlighted message text light gray.
magenta	Makes the highlighted message text magenta.
orange	Makes the highlighted message text orange.
pink	Makes the highlighted message text pink.
red	(Default) Makes the highlighted message text red.
white	Makes the highlighted message text white.
yellow	Makes the highlighted message text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



A list of URL addresses for message links. Each link corresponds with a message in the `ScrollingText`'s message list. If a message has no link, the entry should be blank.

Selecting this property displays a pop-up dialog. URL addresses are shown in the order that they are placed on the list.

#### **Runtime modification**

Use the `setLinkToList` method to modify this property.

#### **Settings**

A list of valid URL strings.

\$ K A + {ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



A list of strings that specifies the messages to be displayed in the ScrollingText component.

To specify the list of strings to display, type the first item in the list box. Press CTRL+ENTER after each item to add the next item. (The Macintosh version provides a button.)

#### Runtime modification

Use the `setMessageList` method to modify this property.

#### Settings

Valid values	Description
[empty]	The component does not have a list of values for this property.
[list]	The component has a list of values for this property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The minimum value a Slider component may have.

#### Runtime modification

Use the `setMinValue` method to modify this property.

#### Settings

Valid values	Description
1	(Default) Minimum value is one.
integer	The minimum value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Use the Minimum Height and Minimum Width properties to define the smallest size of the container. This is the value returned by the `getMinimumSize` method.

**Runtime modification**

At run time, use the `setMinimumHeight` or `setMinimumWidth` method to modify this property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the minimum value of the scroll thumb is in its top or leftmost position of a scrollbar.

#### Runtime modification

At run time, use the `setMinimum` method to modify this property.

Move the scroll thumb to it's minimum position by calling the `setValue` method for the scrollbar and use the value returned by `getMinimum` as the parameter.

#### Settings

Valid values	Description
0	(Default) The minimum scrollbar value is 0.
integer	The minimum scrollbar value.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the standard file dialog box is used to save files or to locate and open files.

#### **Runtime modification**

This property cannot be changed after the FileDialog is constructed.

#### **Settings**

The settings for Mode are:


<b>Valid values</b>	<b>Description</b>
SAVE	Creates a Save File dialog box that can be used to save a file.
LOAD	Creates a Open File dialog box that the user can use to locate and open a file.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The URL of the image to display in the RollOverButton component while the mouse is pressed. The image should be in GIF or JPEG format.<sup>1</sup>

To specify the URL:

1. Click in the Mouse Down Image field in the Property List.
2. Click on the  button (or double-click in the field) to display the URL dialog box.
3. Enter the name of the target HTML file.

#### Runtime modification

Use the `setDownURL` method to modify this property.

#### Settings


Valid values	Description
empty	(Default)
URL	The URL of a GIF or JPEG image to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The URL of the image to display while the mouse is over the component. The image should be in GIF or JPEG format.

To specify the URL:

1. Click in the Mouse Over Image field in the Property List.
2. Click on the  button (or double-click in the field) to display the URL dialog box.
3. Enter the name of the target HTML file.

#### Runtime modification

Use the `setOverURL` method to modify this property.

#### Settings

Valid values	Description
empty	(Default)
URL	The URL of a GIF or JPEG image to display.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether a list allows the user to select multiple items from the list simultaneously.

#### Runtime modification

Use the `setMultipleMode` method to modify this property.

#### Settings

Valid values	Description
true	The list box allows multiple selections.
false	(Default) The list box allows only a single selection at time.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether a list allows the user to select multiple items from the list simultaneously.

#### Runtime modification

Use the `setMultipleSelections` method to modify this property.

#### Settings

Valid values	Description
true	The list box allows multiple selections.
false	(Default) The list box allows only a single selection at time.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the name of the component as used in the Java source code.

All components support this property.

#### **Runtime modification**

This property can not be changed at runtime.

#### **Settings**

Any valid Java identifier. A character string less than 256 characters. The default name for the component is the master component's name and a number.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The delay, in milliseconds, to wait between issuing new action events while the mouse button is continuously pressed in the component. This will only happen if the Notify While Pressed property is true.

#### Runtime modification

At run time, use the `setNotifyDelay` method to modify this property.

#### Settings

Valid Values	Description
1000	(Default)
integer	The delay in milliseconds.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether action events are repeatedly issued while the mouse is pressed in the component. If true, the delay between action events is specified with Notify Delay property.

#### Runtime modification

At run time, use the `setNotifyWhilePressed` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Action events are not sent.
true	Action events are sent.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The number of columns to initialize the Grid with.

#### Runtime modification

At run time, use the `createColumns` method to modify this property. Note that this method deletes all pre-existing columns in the Grid.

#### Settings

Valid Values	Description
0	(Default)
integer	The desired number of columns for the Grid to have.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The number of rows to initialize the Grid with. This also determines the value returned by `getPreferredSize`.

#### Runtime modification

At run time, use the `setPreferredSize` method to modify this property.

#### Settings

Valid Values	Description
0	(Default)
integer	The desired number of rows for the Grid to have.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the component to display either horizontally or vertically.

#### **Runtime modification**

At run time, use the `setOrientation` method to modify this property.

#### **Settings**

<b>Valid value</b>	<b>Description</b>
VERTICAL	(Default) Specifies vertical orientation.
HORIZONTAL	Specifies horizontal orientation.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the spinner arrows are displayed one above the other (vertically) or side-by-side (horizontally)

#### Runtime modification

At run time, use the `setOrientation` method to modify this property.

#### Settings

Valid value	Description
ORIENTATION_VERTICAL (Default)	Display the spinner buttons one above the other.
ORIENTATION_HORIZONTAL	Display the spinner buttons side-by-side.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies padding space in pixels, for a reserved margin around a component. This is the distance between the drawn border and the actual bounds of the component.

#### Runtime modification

At run time, use the `setPadding...` methods to modify this property.

#### Settings

Valid values	Description
0	Specifies that there is no space between the edge of a component and the edge of its display area.
n	A integer expression.

#### Padding Bottom

Sets the number of pixels between the component bounds and the bottom of the drawn border. The default value is 6.

#### Padding Left

Sets the number of pixels between the component bounds and the left side of the drawn border. The default value is 6.

#### Padding Right

Sets the number of pixels between the component bounds and the right side of the drawn border. The default value is 6.

#### Padding Top

Sets the number of pixels between the component bounds and the top of the drawn border. The default value is 10.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies whether the nervous text animation is suspended or running.

#### Runtime modification

Use the `setPaused` method to modify this property.

#### Settings

Valid values	Description
false	(Default) The nervous text animation is running.
true	The nervous text animation is still.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Positions a component within a container managed by the BorderLayout layout manager, see [Layout property](#) .

#### Runtime modification

The placement must be specified when the component is added to its container using the `add` method. To change placement after that, the component must be removed and then added again with a different placement value.

#### Settings

- n If necessary, a component whose placement is North or South is horizontally resized to the width of the container. A component whose placement is East or West is vertically resized to the height of the container.
- n Invalid string values are ignored.

Valid values	Description
North	Places the component at the top of the container.
South	Places the component at the bottom of the container.
East	Places the component along the right side of the container.
West	Places the component along the left side of the container.
Center	Places the component in the center of the container, in the remaining space.
(Empty)	(Default) Same as Center.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the line will be drawn from the lower-left to the upper-right corner or from the upper-left to the lower-right corner. The corners are determined by the component bounds.

#### Runtime modification

Use the `setPositiveSlope` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Line is drawn from the lower-left to the upper-right.
false	Line is drawn from the upper-left to the lower-right.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies if an animation component is “alive” at design time. Form editing is easier when preview is turned off.

#### Runtime modification

At run time, use the `setPreviewMode` method to modify this property.

#### Settings

Valid values	Description
true	Preview mode is on. The animation component runs during design time.
false	Preview mode is off.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color for the progress bar.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Settings

Valid values	Description
black	Makes the progress bar black.
blue	(Default) Makes the progress bar blue.
cyan	Makes the progress bar cyan.
darkGray	Makes the progress bar dark gray.
gray	Makes the progress bar gray.
green	Makes the progress bar green.
lightGray	Makes the progress bar light gray.
magenta	Makes the progress bar magenta.
orange	Makes the progress bar orange.
pink	Makes the progress bar pink.
red	Makes the progress bar red.
white	Makes the progress bar white.
yellow	Makes the progress bar yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether resizing a splitter panel also resizes the component within that panel. If true, that component is scaled to fit within the new panel size, otherwise it is not resized and ends up being clipped as needed when drawn.

#### Runtime modification

At run time, use the `setPropResize` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Resize propagation mode is on.
false	Resize propagation mode is off.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the status message is scrolled repeatedly, or just once.

#### Runtime modification

At run time, use the `setRepeat` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The status message is scrolled repeatedly.
false	The status message is not scrolled repeatedly.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies if the Timer component resets and starts continuously, generating a series of events.

#### Runtime modification

At run time, use the `setRepeat` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The timer resets and starts continuously.
false	The Timer does not repeat. The timer only runs once.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of times to play the list of sounds.

#### Runtime modification

At run time, use the `setRepeat` method to modify this property.

#### Settings

Valid values	Description
-1	Repeats forever.
1	(Default) Plays the list of sounds once.
integer	The number of times to play the list of sounds.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines if the component should repeat when finished showing the animation.

#### Runtime modification

At run time, use the `setRepeatMode` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The animator automatically starts over again when finished.
false	The animator doesn't start over again when finished.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether a Frame or Dialog can be resized at run time.

#### Runtime modification

At run time, use the `setResizable` method to modify this property.

#### Settings

Valid values	Description
true	The user can resize the component at runtime.
false	The user can't resize the component at runtime.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the status message moves from right-to-left or left-to-right.

#### Runtime modification

At run time, use the `setRightToLeft` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The status message moves right-to-left.
false	The status message moves left-to-right.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the number of rows in the component. When both the row and column values are greater than zero, the `getMinimumSize` method returns a size appropriate for the desired number of rows and columns.

#### Runtime modification

At run time, use the `setRows` method to modify this property.

#### Settings

Valid values	Description
0	(Default) Don't use when determining minimum component size.
1-32767	The desired number of rows to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the number of rows to display in a list. If there are more items in the list than the number of specified rows, a scroll bar is automatically added to the component. This affects the dimensions returned by the `getPreferredSize` method and determines the number of rows displayed when this component is automatically laid out.

#### Runtime modification

At run time, use the `setRowsToShow` method to modify this property.

#### Settings

Valid values	Description
0	(Default) Don't use when determining minimum component size.
1-32767	The desired number of rows to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the message will scroll completely off before scrolling on again.

#### Runtime modification

Use the `setScrollClean` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The message will scroll completely off before scrolling on again.
false	The message will scroll back on before scrolling completely off.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This property specifies the direction the `ScrollingText`'s message will scroll.

#### Runtime modification

Use the `setScrollDirection` method to modify this property.

#### Settings

Valid values	Description
SCROLL_LEFT	(Default) Text move from right to left.
SCROLL_RIGHT	Text moves from left to right.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the increment, in pixels, of a `ScrollingPanel`. The increment is the amount that is the component scrolls when the user clicks on the scroll arrow.

#### Runtime modification

At run time, use the `setScrollLineIncrement` method to modify this property.

#### Settings

Valid values	Description
1	(Default) Scroll one pixel per arrow press.
0-32767	The desired number of pixels to scroll per arrow press.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Set delay, in milliseconds, between scroll steps. This function controls the speed of scrolling. A lower number indicates a faster speed.

The minimum value of 30.

#### Runtime modification

Use the `setScrollInterval` method to modify this property.

#### Settings

Valid Values	Description
150 (Default)	
integer $\geq$ 30	The delay in milliseconds.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The number of pixels to move the text on each scroll step.

#### Runtime modification

Use the `setScrollUnit` method to modify this property.

#### Settings

Valid values	Description
10	(Default) Scroll one pixel per step.
0-32767	The desired number of pixels to scroll per step.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Scrollbar display options.

#### **Runtime modification**

This property cannot be changed after the ScrollPane is constructed.

#### **Settings**

<b>Valid values</b>	<b>Description</b>
SCROLLBARS_AS_NEEDED	(Default) Displays scrollbars if the component doesn't fit entirely in the panel.
SCROLLBAR_ALWAYS	Always displays the scrollbars.
SCROLLBAR_NEVER	Never displays the scrollbars.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines which scrollbars are created for the text area.

#### Settings

Valid values	Description
SCROLLBARS_BOTH	(Default) Create and display both vertical and horizontal scrollbars.
SCROLLBARS_VERTICAL_ONLY	Create and display vertical scrollbar only.
SCROLLBARS_HORIZONTAL_ONLY	Create and display horizontal scrollbar only.
SCROLLBARS_NONE	Do not create or display any scrollbars for the text area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determine whether the ComboBox is searchable. A searchable ComboBox allows the user to enter text into the text field as long as it matches one of the existing list items. This allows the user to select from a non-editable list by typing instead of dropping-down the list and selecting with the mouse.

#### Runtime modification

At run time, use the `setSearchable` method to modify this property.

#### Settings

Valid values	Description
true	Allows the user to enter text to search the ComboBox.
false	(Default) The user is not allowed to enter text to search the ComboBox.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the item in a Choice component that is currently selected. The value is zero-relative.

#### Runtime modification

At run time, use the `select` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) Indicates that none of the list items is currently selected.
0-32767	The index of the selected item. The first item in the list has an index of zero (0).

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the index of the selected RadioButton. The value is zero-relative. A -1 indicates no RadioButton is selected.

#### Runtime modification

At run time, use the `setSelectedRadioButtonIndex` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) Indicates that none of the RadioButtons is currently selected.
0-32767	The index of the selected item. The first RadioButton has an index of zero (0).

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the foreground color for the selected element in the component. For example, the selected item on the Calendar component is the day of the month.

#### Runtime modification

At run time, use the `setSelectedColor` method to modify this property. For example:

```
calendar1.setSelectedColor(new Color(16711935));
```

#### Settings

Valid values	Description
black	(Default) Makes the selection foreground black.
blue	Makes the selection foreground blue.
cyan	Makes the selection foreground cyan.
darkGray	Makes the selection foreground dark gray.
gray	Makes the selection foreground gray.
green	Makes the selection foreground green.
lightGray	Makes the selection foreground light gray.
magenta	Makes the selection foreground magenta.
orange	Makes the selection foreground orange.
pink	Makes the selection foreground pink.
red	Makes the selection foreground red.
white	Makes the selection foreground white.
yellow	Makes the selection foreground yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether a menu item is a separator. A separator displays as a solid line in the menu and cannot be selected at run time.

#### Runtime modification

At run time, use the `addSeparator` method of the `MenuItem`'s parent menu to modify this property, or call the `MenuItem`'s `setLabel` method with the label "-". The label "-" is reserved to display a separator between menu items. If you add a menu item which uses "-" as a Label, it behaves like a menu separator.

#### Setting

Valid values	Description
true	The item is a separator
false	(Default) The item is a standard menu item.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The distance in pixels to shift the animated image after each loop. Each image is shifted horizontally from the previous image by the specified offset. The offset may be positive or negative. If the offset is positive, the first image is drawn on the left side of the component and subsequent images are drawn to the right of the first image by offset pixels. If the offset is negative, the first image is drawn at the right side of the component area. This causes the animated image to appear to move across the container.

#### Runtime modification

At run time, use the `setShiftOffset` method to modify this property.

#### Settings

Valid values	Description
< 0	Image is drawn on the right side of the component and moves left.
> 0	Image is drawn on the left side of the component and moves right.
0	Image doesn't move.
10	(Default) Image is drawn on the left side of the component and moves right.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies whether the border of the component is drawn.

#### Runtime modification

At run time, use the `setShowBorder` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The component's border displays.
false	The component's border does not display.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether this component will visually indicate when it has the focus. This is done by drawing a focus rectangle within the component as needed.

#### Runtime modification

At run time, use the `setShowFocus` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Visually indicates whether this component has the focus.
false	Never indicates whether this component has the focus, even when it does.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Toggles the display of the component's horizontal scrollbar.

#### Runtime modification

At run time, use the `setShowHorizontalScroll` method to modify this property.

#### Settings

Valid values	Description
true	The horizontal scrollbar displays.
false	(Default) The horizontal scrollbar does not display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the link URL will be displayed in the status area when the mouse is over the button. This flag also controls erasing of the status area after the URL has been displayed.

#### Runtime modification

At run time, use the `setShowURLStatus` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Show the link URL in the status area when the mouse is over the button.
false	Do not show the link URL in the status area.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Shows the percentage complete as a number, i.e. 75%.

#### Runtime modification

At run time, use the `setShowProgress` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The progress percentage string is visible.
false	The progress percentage string is not visible.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Toggles the display of the component's vertical scrollbar.

#### Runtime modification

At run time, use the `setShowHorizontalScroll` method to modify this property.

#### Settings

Valid values	Description
true	The vertical scrollbar displays.
false	(Default) The vertical scrollbar does not display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the width of the `ToolBarSpacer`, in pixels.

#### Runtime modification

At run time, use the `setSpace` method to modify this property.

#### Settings


Valid values	Description
10	(Default)
integer	The size of the spacer, in pixels

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The URL of the image to display when the mouse is not over the component. If no image is specified the component will be invisible when the mouse is not over it. The image should be in GIF or JPEG format.

To specify the URL:

1. Click in the Standard Image field in the Property List.
2. Click on the  button (or double-click in the field) to display the URL dialog box.
3. Enter the name of the target HTML file.

#### Runtime modification

Use the `setStandardURL` method to modify this property.

#### Settings

Valid values	Description
empty	(Default)
URL	The URL of a GIF or JPEG image to display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the initial state of the component.

#### Runtime modification

At run time, use the `setState` method to modify this property.

#### Settings

For most components:

Valid values	Description
true	The component has the value: true.
false	(Default) The component has the value: false.

For the `StateCheckBox` component:

The [Style property](#) must be set to `THREE_STATE` for the `State` property to support the third, 'default' setting.

The third state is often used to indicate that an attribute of the current selection varies across the selected items. For example, a checkbox used to indicate "bold text" might use the third state if the currently selected text included both bold and plain characters. It is also used to indicate a default action or state.

Valid values	Description
<code>STATE_UNCHECKED</code>	(Default) The checkbox is unchecked.
<code>STATE_DEFAULT</code>	Third state of three state check box. Box is grayed and checked.
<code>STATE_CHECKED</code>	The checkbox is checked.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the text that appears in the status bar.

**Runtime modification**

At run time, use the `setStatusText` method to modify this property.

**Settings**

Any string is valid.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color of the text in the status bar.

#### Runtime modification

At run time, use the `setStatusTextColor` method to modify this property.

#### Settings

Valid values	Description
black	(Default) Makes the status text black.
blue	Makes the status text blue.
cyan	Makes the status text cyan.
darkGray	Makes the status text dark gray.
gray	Makes the status text gray.
green	Makes the status text green.
lightGray	Makes the status text light gray.
magenta	Makes the status text magenta.
orange	Makes the status text orange.
pink	Makes the status text pink.
red	Makes the status text red.
white	Makes the status text white.
yellow	Makes the status text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the text that appears in the StatusScroller message.

**Runtime modification**

At run time, use the `setString` method to modify this property.

**Settings**

Any string is valid.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the border bevel style for the component.

#### Runtime modification

At run time, use the `setBevelStyle` method to modify this property.

#### Settings

Valid values	Description
BEVEL_LINE	(Default) Uses a single line border. This provides a "flat" appearance. Flat appearance should be used for a scrollable region or control, such as a drop-down list and drop-down combo box.
BEVEL_LOWERED	3D effect. Makes the component appear lower than the surrounding area. This field style is standard for text boxes, check boxes, spin boxes, and list boxes.
BEVEL_NONE	No border is used.
BEVEL_RAISED	3D effect. Makes the component appear higher than the surrounding area. This window style is used for primary and secondary windows.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the style of the checkbox.

The Style property must be set to THREE\_STATE for the [State property](#) to support the 'default' setting.

#### Runtime modification

At run time, use the `setStyle` method to modify this property.

#### Settings

Valid values	Description
TWO_STATE	(Default) Allows two states, checked and unchecked.
THREE_STATE	Allows three states, checked, unchecked, and default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether sound clips are played one after the other, or all simultaneously.

#### Runtime modification

At run time, use the `setSyncMode` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The sound clips play sequentially.
false	All sound clips play at once.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines the text strings to use for each tab label in a `TabPanel` component.

#### Runtime modification

At run time, use the `setTabPanelLabels` method to modify this property.

#### Settings

Valid values	Description
[empty]	There is no data currently in the component.
[list]	The component currently has a list of values.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies if the tabs in a `TabPanel` component display on the bottom or the top of the component.

#### Runtime modification

At run time, use the `setTabsOnBottom` method to modify this property.

#### Settings

Valid values	Description
true	Tabs display on the bottom.
false	(Default) Tabs display on the top.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Controls whether the menu can be “torn off” the menu bar.

#### **Runtime modification**

The Tear Off property cannot be changed at runtime. This property is determined at design time.

#### **Settings**

<b>Valid values</b>	<b>Description</b>
true	The user can tear off the menu
false	(Default) The menu cannot be torn off.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the text that appears in the component.

**Runtime modification**

At run time, use the `setText` method to modify this property.

**Settings**

Any string is valid.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the color of the associated text.

#### Runtime modification

At run time, use the `setTextColor` method to modify this property. For the `ProgressBar`, use the `setProgressBarTextColor` method.

#### Settings

Valid values	Description
black	(Default) Makes the text black.
blue	Makes the text blue.
cyan	Makes the text cyan.
darkGray	Makes the text dark gray.
gray	Makes the text gray.
green	Makes the text green.
lightGray	Makes the text light gray.
magenta	Makes the text magenta.
orange	Makes the text orange.
pink	Makes the text pink.
red	Makes the text red.
white	Makes the text white.
yellow	Makes the text yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Used by the Slider component to specify the number of tick marks that display on the component bar. This is the range in value between each tick mark.

#### Runtime modification

At run time, use the `setTickFreq` method to modify this property.

Valid values	Description
1	(Default) Each tick mark corresponds to a change in value of 1.
n	Each tick mark corresponds to a change in value of n.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the style of the tick marks that display on the Slider components.

#### Runtime modification

At run time, use the `setTickStyle` method to modify this property.

#### Settings

Valid values	Description
TICK_BOTH	(Default) Tick marks on both the top and bottom..
TICK_BOTTOM	Tick marks only along the bottom edge.
TICK_NONE	No tick marks.
TICK_TOP	Tick marks only along the top edge.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the time to wait before posting an event.

**Runtime modification**

At run time, use the `setDelay` method to modify this property.

**Settings**

The valid range is 0 to 32767 milliseconds. The default is 1000.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the text to display in the title bar of a container component.

**Runtime modification**

At run time, use the `setTitle` method to modify this property.

**Settings**

Any string is valid.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the increment unit for a scrollbar component. This is the amount that is added or subtracted from this scroll bar's Value property when the user presses the up or down scroll arrow.

#### Runtime modification

Use the `setUnitIncrement` method to modify this property.

#### Settings

Valid values	Description
1	(Default)
0-32767	Valid values are within this range.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**General**

Specifies a pathname in URL format.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



For animators and slide shows, this is a list of URL addresses that refer to GIF or JPEG files comprising an animated sequence. Selecting this property displays a pop-up dialog. Images display in the order that they are placed on the list. The `MovingAnimation` component displays the URL at a transition speed determined by the `Delay` property, to build an animated image.

For the `SoundPlayer`, this is a list of URL addresses that refer to sound data files. The sound data gets played either serially or simultaneously, depending on the value of the `Synchronized Mode` property.

#### **Runtime modification**

For animators, use the `setImageList` method to modify this property.

For the `SlideShow` component, use the `addImageAndDescription` method.

For the `SoundPlayer` component, use the `addStringURL`, `addURL`, and the `setURLList` methods.

#### **Settings**

A list of valid URL strings.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the border around panels is drawn in a three-dimensional way.

#### Runtime modification

Use the `setUse3DBorder` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The borders around panels are drawn in a three-dimensional way.
false	The borders around panels are not drawn in a three-dimensional way.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether the contents of the button will be drawn offset down and to the right by the bevel height amount when the button is drawn in its down state.

#### Runtime modification

Use the `setUseOffset` method to modify this property.

#### Settings

Valid values	Description
true	Offset objects in button when drawn in its down state.
false	(Default) Don't offset objects in button.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the position of the thumb in slider, spinner, progress bar, scrollbar and JFC Swing versions of the components.

#### **Runtime modification**

At run time, use the `setValue` method to modify this property.

For spinners, use the `setCurrent` method to modify this property.

#### **Settings**

The valid range is typically 0 to 32767.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the vertical alignment of the text in the component.

#### Settings

Valid values	Description
ALIGN_CENTERED	(Default) The text is drawn centered vertically within the component.
ALIGN_TOP	The text is drawn at the top of the component.
ALIGN_BOTTOM	The text is drawn at the bottom of the component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



The range of values that are displayed at one time. The scrollbar thumb grows to fill the slide as this value approaches the difference between the maximum and minimum values. This makes the scrollbar thumb indicate the size of a page, while the scrollbar slide indicates the size of the entire document.

The scroll bar uses this value when paging up or down by a page.

#### Runtime modification

At run time, use the `setVisibleAmount` method to modify this property.

#### Settings

Valid Values	Description
0	(Default)
0-32767	Valid values are within this range.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Controls whether a component is visible at run time or not. When this property is false, the component is invisible. An invisible component cannot be seen by the user nor does it take up space in its container, but it does continue to exist.

All visual components and containers support this property.

#### Runtime modification

At run time, you can make components visible or invisible using the `show` or `hide` method.

#### Settings

Valid value	Description
true	(Default) The component is visible.
false	The component is not visible.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the number of rows in the component. That value is used to calculate preferred size for the component when it is laid out. A scrollbar is used when the actual number of rows exceeds this value.

**Settings**

Any valid integer expression. The default value is 0 (zero).

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



This property is a subproperty of [GridBagConstraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Specifies how to distribute extra horizontal space for a component within a grid bag layout. The weight of a row is calculated as the maximum *weightx* of all the components in a row. If the resulting layout is smaller vertically than the area it needs to fill, the extra space is distributed to each row in proportion to its weight. A row that has weight 0 receives no extra space. The component does not take any extra space if the [Fill property](#) is set to NONE.

If all the weights are zero, extra space appears between the grids of the cell and the top and bottom edges.

#### **Runtime modification**

At run time, use the `setConstraints` method to modify this property.

#### **Settings**

The default value is 0 (zero). This setting can be any valid `double` expression.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



This property is a subproperty of [GridBagConstraints](#) . To access this property, set the component's parent container's [Layout property](#) to GridBagLayout.

Specifies how to distribute extra vertical space for components within a grid bag layout. The weight of a column is calculated as the maximum *weighty* of all the components in a row. If the resulting layout is smaller horizontally than the area it needs to fill, the extra space is distributed to each column in proportion to its weight. A column that has weight 0 receives no extra space. The component does not take any extra space if the [Fill property](#) is set to NONE.

If all the weights are zero, extra space appears between the grids of the cell and the right and left edges.

#### **Runtime modification**

At run time, use the `setConstraints` method to modify this property.

#### **Settings**

The default value is 0 (zero). This setting can be any valid `double` expression.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether the spinner value can wrap between its maximum and minimum values.

#### Runtime modification

At run time, use the `setWrappable` method to modify this property.

#### Settings

Valid Values	Description
false	(Default) The value cannot wrap.
true	The value can wrap.

About...

**Obselete**

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the menu bar to display in a Frame. You can create more than one menu bar in a frame, but only one Menu Bar can be displayed at a time.

#### **Runtime modification**

At run time, you can display a different menu bar by calling the Frame's `setMenuBar` method.

#### **Settings**

Valid values	Description
true	The menu bar displays in the frame.
false	The menu bar is not displayed.

## Change history

6/1/98 - created swing components file

7/16/98 - all 3.0 swing components added to file in alpha/beta draft form

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also


 See Also


 See Also

 See Also

 See Also

 See Also

 See Also


 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

## Swing Components List

## Swing Components

### Swing

[JButton](#)  
[JCheckBox](#)  
[JComboBox](#)  
[JEditorPane](#)  
[JLabel](#)  
[JList](#)  
[JOptionPane](#)  
[JPasswordField](#)  
[JProgressBar](#)

### Forms

[JApplet](#)  
[JDialog](#)  
[JFrame](#)  
[JInternalFrame](#)  
[JWindow](#)

### Swing (continued)

[JRadioButton](#)  
[JScrollBar](#)  
[JSlider](#)  
[JTable](#)  
[JTextArea](#)  
[JTextField](#)  
[JTextPane](#)  
[JToggleButton](#)  
[JTree](#)  
[JViewport](#)

### Actions

[Action](#)

### Swing Containers

[JPanel](#)  
[JRootPane](#)  
[JScrollPane](#)  
[JSplitPane](#)  
[JTabbedPane](#)  
[JToolBar](#)

### Icons

[ImageIcon](#)

### Swing Additons

[JActionButton](#)  
[JBorderlessButton](#)  
[JButtonGroupPanel](#)  
[JChart](#)  
[JCurrencyTextField](#)  
[JDateMaskedField](#)  
[JListChooser](#)  
[JLookAndFeelComboBox](#)  
[JMaskedTextField](#)  
[JToolBarSeparator](#)  
[JWrappingLabel](#)

### Models

[FileTableModel](#)  
[StringComboBoxModel](#)  
[StringListModel](#)  
[StringTableModel](#)  
[StringTreeModel](#)  
[UnsignedNumberDocume](#)  
[nt](#)

### Menus and Menu Items

[JActionMenuItem](#)  
[JCheckBoxMenuItem](#)  
[JMenu](#)  
[JMenuBar](#)  
[JMenuItem](#)  
[JPopupMenu](#)  
[JRadioButtonMenuItem](#)  
[JSeparator](#)

### Borders

[BevelBorder](#)  
[CompoundBorder](#)  
[EmptyBorder](#)  
[EtchedBorder](#)  
[LineBorder](#)  
[MatteBorder](#)  
[SoftBevelBorder](#)  
[TitledBorder](#)

## Swing components

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a simple button that initiates an action. A button usually displays a text label.

Use an action button to:

- accept or yield focus. Buttons accept and yield focus automatically by default.
- respond to a user event. Buttons accept clicked events automatically by default.
- send an action event to another visual component. To send an action event to another component, use the Interaction Wizard. Optionally, you can register an event listener with the button in project source code.

A button appears outlined when it possesses input focus. This indicates that it can receive a user event. When the user presses the button it get redrawn so that it appears depressed. When the user then releases the mouse button while still over the button component, a click event is generated.

### Properties

Use the [Action property](#) to set the action to perform when the user clicks the button.

Use the [Action Command property](#) to set the command name of the action event fired.

### Coding the component

In project source code, use this syntax to create a new JButton:

```
JActionButton1 = new com.symantec.itools.swing.actions.JActionButton();  
getContentPane().add(JActionButton1);
```

Part	Type	Description
JActionButton1	JActionButton	The variable name used to reference the component in code.

## **JActionButton Properties**

<a href="#"><u>Action</u></a>	<a href="#"><u>FocusTraversable</u></a>
<a href="#"><u>Action Command</u></a>	<a href="#"><u>Font</u></a>
<a href="#"><u>Alignment/Position</u></a>	<a href="#"><u>Foreground</u></a>
<a href="#"><u>AlignmentX</u></a>	<a href="#"><u>Icons</u></a>
<a href="#"><u>AlignmentY</u></a>	<a href="#"><u>Insets</u></a>
<a href="#"><u>Autoscrolls</u></a>	<a href="#"><u>ManagingFocus</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>Mnemonic</u></a>
<a href="#"><u>Border</u></a>	<a href="#"><u>Model</u></a>
<a href="#"><u>BorderPainted</u></a>	<a href="#"><u>Name</u></a>
<a href="#"><u>Bounds</u></a>	<a href="#"><u>NextFocusableComponent</u></a>
<a href="#"><u>Class</u></a>	<a href="#"><u>Opaque</u></a>
<a href="#"><u>Cursor</u></a>	<a href="#"><u>OptimizedDrawingEnabled</u></a>
<a href="#"><u>DebugGraphicsOptions</u></a>	<a href="#"><u>PaintingTile</u></a>
<a href="#"><u>Default Background</u></a>	<a href="#"><u>RequestFocusEnabled</u></a>
<a href="#"><u>Default Font</u></a>	<a href="#"><u>RolloverEnabled</u></a>
<a href="#"><u>Default Foreground</u></a>	<a href="#"><u>Selected</u></a>
<a href="#"><u>DefaultButton</u></a>	<a href="#"><u>ToolTipText</u></a>
<a href="#"><u>Display</u></a>	<a href="#"><u>UIClassID</u></a>
<a href="#"><u>DoubleBuffered</u></a>	<a href="#"><u>ValidateRoot</u></a>
<a href="#"><u>FocusCycleRoot</u></a>	<a href="#"><u>Visible</u></a>
<a href="#"><u>FocusPainted</u></a>	<a href="#"><u>VisibleRect</u></a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a menu Item that uses actions. An example would be a menu item that allows you to cut, copy, or paste items.

### Properties

Use the [Action property](#) to set the action to perform when the user selects the menu item.

Use the [Action Command property](#) to set the command name of the action event fired.

### Coding the component

In project source code, use this syntax to create a new JMenuItem:

```
JActionMenuItem1 = new com.symantec.itools.swing.actions.JActionMenuItem();  
JMenu1.add(JActionMenuItem1);
```

Part	Type	Description
JActionMenuItem1	JActionMenuItem	The variable name used to identify the Action Menu Item in code.
JMenu1	JMenu	The variable name used to identify the menu that the menu item is added to.

## **JActionMenuItem Properties**

[Accelerator](#)

[Action](#)

[Action Command](#)

[Alignment/Position](#)

[AlignmentX](#)

[AlignmentY](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[FocusCycleRoot](#)

[FocusPainted](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[Icons](#)

[Insets](#)

[ManagingFocus](#)

[Mnemonic](#)

[Model](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[RolloverEnabled](#)

[Selected](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[Visible](#)

[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

## Description

An applet is the parent container that holds all components comprising an applet's GUI.

Use a JApplet container to create an applet program.

Build an applet in Visual Cafe by choosing the JFC Applet icon displayed in the New Project window. Then, to create the applet's GUI interface, drag the Swing components you want from the Component Palette or Component Library onto the Form Designer window attached to that applet project.

JApplets support the following functionality:

- Applet programs are run by another program.
- Applet components include some built-in support for [threads](#). (For simple applets, you do not need to consider how your applet will handle threads.)
- Applets contain [methods](#) which manage sounds, pictures, and URL addresses.
- Applet programs can write to files only if the applets are located on the host system of the managing Web page. An applet cannot write a file over a network.

Your applet can send and receive [events](#). Use the Interaction Wizard to post predefined events between your JApplet container and the components it contains.

An applet program is a self-contained block of running Java code that a Web browser launches and runs from a Web page. Applets are called using the HTML `</Applet>` tag. To display an applet on a Web page, use an HTML file containing a reference to your applet. In HTML, a simple applet tag looks like this:

```
<APPLET CODE="JApplet1.class" WIDTH=100 HEIGHT=100></APPLET>
```

## Properties

You can change the layout manager using the [Layout property](#).

You can assign a specific MenuBar to JApplet by using the [JMenuBar property](#).

## Coding the component

In project source code, use this syntax to create a new JApplet:

```
public class JApplet1 extends JApplet
```

Part	Type	Description
------	------	-------------

JApplet1	JApplet	The variable name used to identify the applet in code.
----------	---------	--

## **JApplet Properties**

[Background](#)

[Border](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[InheritBackground](#)

[InheritFont](#)

[InheritForeground](#)

[JMenuBar](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a simple button that initiates an action. A button usually displays a text label. This component does not display a border around the button.

Use a `JBorderlessButton` to:

- accept or yield focus. Buttons accept and yield focus automatically by default.
- respond to a user event. Buttons accept clicked events automatically by default.
- send an action event to another visual component. To send an action event to another component, use the Interaction Wizard. Optionally, you can register an event listener with the button in project source code.

A button appears outlined when it possesses input focus. This indicates that it can receive a user event. When the user presses the button it get redrawn so that it appears depressed. When the user then releases the mouse button while still over the button component, a click event is generated.

### Properties

To specify text for the label, use the [Text property](#).

When a button is disabled, it is 'grayed out.' This indicates that it cannot receive user input. Set the enable state using the [Enabled property](#).

### Coding the component

In project source code, use this syntax to create a new `JBorderlessButton`:

```
JBorderlessButton1 = new com.symantec.itools.swing.JBorderlessButton();  
getContentPane().add(JBorderlessButton1);
```

Part	Type	Description
JBorderlessButton1	JBorderlessButton	The variable name used to identify the button in code.

## **JBorderlessButton Properties**

<a href="#"><u>Action Command</u></a>	<a href="#"><u>Font</u></a>
<a href="#"><u>Alignment/Position</u></a>	<a href="#"><u>Foreground</u></a>
<a href="#"><u>AlignmentX</u></a>	<a href="#"><u>Icons</u></a>
<a href="#"><u>AlignmentY</u></a>	<a href="#"><u>Insets</u></a>
<a href="#"><u>Autoscrolls</u></a>	<a href="#"><u>ManagingFocus</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>Mnemonic</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>Mnemonic</u></a>
<a href="#"><u>Border</u></a>	<a href="#"><u>Model</u></a>
<a href="#"><u>Bounds</u></a>	<a href="#"><u>Name</u></a>
<a href="#"><u>Class</u></a>	<a href="#"><u>NextFocusableComponent</u></a>
<a href="#"><u>Cursor</u></a>	<a href="#"><u>Opaque</u></a>
<a href="#"><u>DebugGraphicsOptions</u></a>	<a href="#"><u>OptimizedDrawingEnabled</u></a>
<a href="#"><u>Default Background</u></a>	<a href="#"><u>PaintingTile</u></a>
<a href="#"><u>Default Font</u></a>	<a href="#"><u>RequestFocusEnabled</u></a>
<a href="#"><u>Default Foreground</u></a>	<a href="#"><u>RolloverEnabled</u></a>
<a href="#"><u>DefaultButton</u></a>	<a href="#"><u>Selected</u></a>
<a href="#"><u>DoubleBuffered</u></a>	<a href="#"><u>Text</u></a>
<a href="#"><u>Enabled</u></a>	<a href="#"><u>ToolTipText</u></a>
<a href="#"><u>FocusCycleRoot</u></a>	<a href="#"><u>UIClassID</u></a>
<a href="#"><u>FocusPainted</u></a>	<a href="#"><u>ValidateRoot</u></a>
<a href="#"><u>FocusTraversable</u></a>	<a href="#"><u>Visible</u></a>
	<a href="#"><u>VisibleRect</u></a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a simple button that initiates an action. A button usually displays a text label. The JButton component is a JFC Swing component.

Use a JButton to:

- accept or yield focus. Buttons accept and yield focus automatically by default.
- respond to a user event. Buttons accept clicked events automatically by default.
- send an action event to another visual component. To send an action event to another component, use the Interaction Wizard. Optionally, you can register an event listener with the button in project source code.

A button appears outlined when it possesses input focus. This indicates that it can receive a user event. When the user presses the button it get redrawn so that it appears depressed. When the user then releases the mouse button while still over the button component, a click event is generated.

### Properties

To specify text for the label, use the [Text property](#).

When a button is disabled, it is 'grayed out.' This indicates that it cannot receive user input. Set the enable state using the [Enabled property](#).

To set the border type, use the [Border property](#).

### Coding the component

In project source code, use this syntax to create a new JButton:

```
JButton1 = new com.sun.java.swing.JButton();  
getContentPane().add(JButton1);
```

Part	Type	Description
JButton1	JButton	The variable name used to identify the button in code.

## **JButton Properties**

[Action Command](#)  
[Alignment/Position](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[Default Background](#)  
[DefaultButton](#)  
[Default Font](#)  
[Default Foreground](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusPainted](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Insets](#)  
[ManagingFocus](#)  
[Mnemonic](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[RolloverEnabled](#)  
[Selected](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

Extends JPanel.

Creates a rectangular panel area that automatically groups the buttons that it contains.

Add buttons to the panel by dragging them from the Component Palette onto the panel.

All the buttons within the JButtonGroupPanel act together. Only one of the buttons can be "on" at a time. This is accomplished by making each button added to the panel a member of the panel's button group.

**Properties**

Use the [Layout property](#) to control how the buttons in the group will display. The default layout is the Grid layout for this component.

**Coding the component**

In project source code, use this syntax to create a new JButtonGroupPanel:

```
JButtonGroupPanel1 = new com.symantec.itools.swing.JButtonGroupPanel();
getContentPane().add(JButtonGroupPanel1);
```

Part	Type	Description
JButtonGroupPanel1	JButtonGroupPanel	The variable name used to identify the button group panel in code.

## **JButtonGroupPanel Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Use this component to create a chart.

This component supports bar, line, scatter, area, and pie chart types. Use a TableModel for entering data. The TableModel can contain objects of type Integer, Long, Double, String, or DataElem. DataElem contains support for individual data color and label information.

 See Also

**Note:** When using JChart in Visual Cafe Database Edition, be aware that a databound JChart does not display data in the chart at design-time, only at run-time can you see how a chart will display.

### Properties

Use the [Model property](#) to select an appropriate model for your data. StringTableModel works well. Put the series names in the column headings. Each row represents another data point. For example if your column headings are "Dogs, Cats" then a single data point would be represented as "25, 12" which would mean 25 dogs and 12 cats. Each row is a new data point.

To determine what type of graph you want, use the [Style property](#).

Use the [Title property](#) to enter the title to be displayed above the chart.

To set the number of major tick increments on the Y axis, use the [NumYTicks property](#).

Set the [ShowLegend property](#) to true to display a legend for the data. The legend uses the column headings created in the model as the text source.

### Coding the component

In project source code, use this syntax to create a new JChart:

```
JChart1 = new com.symantec.itools.swing.JChart();
stringTableModel1 = new com.symantec.itools.swing.models.StringTableModel();
JChart1.setModel(stringTableModel1);
add(JChart1);
```

Part	Type	Description
JChart1	JChart	The variable name used to identify the chart in code.
stringTableModel1	Model	The variable name used to identify the model in code.
Model	String	The name of the model assigned to the chart.

## JChart Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[InheritBackground](#)  
[InheritFont](#)  
[InheritForeground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[NumYTicks](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ShowGrid](#)  
[ShowLegend](#)  
[Style](#)  
[Title](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)  
[YAxisMax](#)  
[YAxisMin](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Extends JToggleButton.

A labeled square box that the user toggles to turn an option on or off. To toggle, either click on the JCheckBox's box or label, or press the spacebar when it has the focus.

Use a JCheckBox to:

- display a true/false value to the user and optionally allow it to be changed.
- display a possible range of options to the user and optionally allow one or more to be chosen.

See also: [JToggleButton](#)

See also: [JRadioButton](#)

### Properties

To set the default state, either checked (true) or cleared (false), use the [Selected property](#).

Use the Interaction Wizard to have another component perform some action when the JCheckBox value changes and generates an ItemEvent type event. Actions can be triggered when the JCheckBox generates other standard events, too.

To group JCheckBox components, use the [JButtonGroupPanel component](#). Only one of the components in the JButtonGroupPanel can be "on" at a time. This allows the user to select one of a possible range of options.

### Coding the component

In project source code, use this syntax to create a new JCheckBox:

```
JCheckBox1.setText("1");  
JCheckBox1.setSelected(true);  
getContentPane().add(JCheckBox1);
```

Part	Type	Description
JCheckBox1	JCheckBox	The variable name used to identify the checkbox in code.
Text	String	The text to display next to the checkbox.
Selected	boolean	The initial state of the checkbox: true is "checked"; false is "unchecked".

## JCheckBox Properties

[Alignment/Position](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusPainted](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[HorizontalAlignment](#)

[HorizontalTextPosition](#)

[Icons](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Insets](#)

[Layout](#)

[ManagingFocus](#)

[Mnemonic](#)

[Model](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[RolloverEnabled](#)

[Selected](#)

[Text](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[VerticalAlignment](#)

[VerticalTextPosition](#)

[Visible](#)

[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

Creates a checkbox that can be included in a menu. Clicking the associated menu item toggles its state between checked (marked with a check mark) and unchecked.

Use `JCheckBoxMenuItem` whenever you include a menu item that you want to toggle between on and off.

Use the Interaction Wizard to trigger an action when the `CheckBoxMenuItem` is checked or unchecked.

### Properties

Use the [Checkbox property](#) to set the initial menu state, either checked or unchecked.

The [Label property](#) sets the menu item text.

### Coding the component

The Java language does not require you to create a variable to reference a menu item. Instead, you can refer to the menu item by its label.

In project source code, use this syntax to create a new `JCheckBoxMenuItem`:

```
JCheckBoxMenuItem1.setText("jcheckBoxMenuItem");
```

To add a checkbox menu item to an existing menu, call the menu component's `add` method:

```
JMenu1.add(JCheckBoxMenuItem1);
```

Part	Type	Description
<code>JCheckBoxMenuItem1</code>	<code>JCheckBoxMenuItem</code>	The variable name used to refer to the component in code.
<code>JMenu1</code>	<code>Menu</code>	The variable name of the menu component to which you are adding the menu item.

## JCheckBoxMenuItem Properties

[Accelerator](#)  
[Action Command](#)  
[Alignment/Position](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusPainted](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Insets](#)  
[ManagingFocus](#)  
[Mnemonic](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[RolloverEnabled](#)  
[Selected](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A text box with an attached list box. Users can either type the desired value into the text box or select their choice using the list box.

Use a JComboBox to:

- compactly display a list of items.
- manage a list of items longer than the display window size.
- change display fonts and colors and add edit and search functions.

When you include a JComboBox in a container, make sure that the attached list box doesn't cover up other components at run time.

### Properties

To add an item to a JComboBox list box, use [Model](#) in the Property List window to set the model to StringComboBoxModel. Use the [Items property](#) in the [StringComboBoxModel component](#) to add the items. To include more than one item in the list, press CTRL+ENTER after typing each item.

Vertical scrollbars are automatically generated if you exceed the value set for the [MaximumRowCount property](#).

To use a different font in the JComboBox, set the [Font property](#).

To make the JComboBox list editable, set the [Editable property](#) to true.

A searchable JComboBox allows the user to enter text into the text field as long as it matches one of the existing list items. This allows the user to select from a non-editable list by typing instead of dropping down the list and selecting with the mouse. Setting the Editable property to true also allows the user to treat the JComboBox as searchable.

### Coding the component

In project source code, use this syntax to create a new JComboBox:

```
JComboBox1 = new com.sun.java.swing.JComboBox();  
getContentPane().add (JComboBox1)
```

Part	Type	Description
JComboBox1	JComboBox	The variable name used to identify the combo box in code.

## JComboBox Properties

[Action Command](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Editable](#)  
[Editor](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ItemCount](#)  
[KeySelectionManager](#)  
[LightWeightPopupEnabled](#)  
[ManagingFocus](#)  
[MaximumRowCount](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[Renderer](#)  
[RequestFocusEnabled](#)  
[SelectedIndex](#)  
[SelectedItem](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A text field which has currency formatting logic applied to the user input.

Use a JCurrencyTextField to:

- set the character used as a currency symbol.
- place a currency symbol before or after a value.
- limit the number of characters displayed after the decimal point.
- set the character used as a decimal point.
- treat the value the same way ATM machines do (use a stationary decimal point and move digits left as the user types).

### Properties

To specify the required format for the text entry, use the [Leading Symbol property](#), [Symbol property](#), [DecimalPoint property](#), [DigitsAfterDecimal property](#), [Separator property](#), [SeparatorEnabled property](#), and the [ATMMode property](#).

To specify a text string to display in the field by default, use [Text property](#).

To make the default text editable, use the [Editable property](#). You do not need to write code or use the Interaction Wizard to connect the JCurrencyTextField component with any other component to display text.

### Coding the component

To create a JCurrencyTextField in project source code, use this syntax:

```
JCurrencyTextField1 = new symantec.itools.swing.JCurrencyTextField();  
getContentPane().add(JCurrencyTextField1);
```

Part	Type	Description
JCurrencyTextField1	JCurrencyTextField	The variable name used to identify the currency text field in code.

## JCurrencyTextField Properties

<a href="#">AlignmentX</a>	<a href="#">Default Background</a>
<a href="#">AlignmentY</a>	<a href="#">Default Font</a>
<a href="#">Autoscrolls</a>	<a href="#">Default Foreground</a>
<a href="#">ATMMode</a>	<a href="#">Insets</a>
<a href="#">Background</a>	<a href="#">Keymap</a>
<a href="#">Border</a>	<a href="#">Leading Symbol</a>
<a href="#">Bounds</a>	<a href="#">ManagingFocus</a>
<a href="#">Caret</a>	<a href="#">Margin</a>
<a href="#">CaretColor</a>	<a href="#">Name</a>
<a href="#">Class</a>	<a href="#">NextFocusableComponent</a>
<a href="#">Columns</a>	<a href="#">Opaque</a>
<a href="#">Cursor</a>	<a href="#">OptimizedDrawingEnabled</a>
<a href="#">DebugGraphicsOptions</a>	<a href="#">PaintingTile</a>
<a href="#">DecimalPoint</a>	<a href="#">PreferredScrollableViewportSize</a>
<a href="#">DigitsAfterDecimal</a>	<a href="#">RequestFocusEnabled</a>
<a href="#">DisabledTextColor</a>	<a href="#">ScrollOffset</a>
<a href="#">Document</a>	<a href="#">ScrollableTracksViewportHeight</a>
<a href="#">DoubleBuffered</a>	<a href="#">ScrollableTracksViewportWidth</a>
<a href="#">Editable</a>	<a href="#">Selection</a>
<a href="#">Enabled</a>	<a href="#">Separator</a>
<a href="#">FocusAccelerator</a>	<a href="#">SeparatorEnabled</a>
<a href="#">FocusCycleRoot</a>	<a href="#">Symbol</a>
<a href="#">FocusTraversable</a>	<a href="#">Text</a>
<a href="#">Font</a>	<a href="#">ToolTipText</a>
<a href="#">Foreground</a>	<a href="#">UIClassID</a>
<a href="#">Highlighter</a>	<a href="#">ValidateRoot</a>
<a href="#">HorizontalAlignment</a>	<a href="#">Visible</a>
	<a href="#">VisibleRect</a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Extends JMaskedTextField.

A text field which has edit masking logic applied to the user input for dates and times. The masking logic ensures that a match between the input character and the relevant filter for the cursor position is made. Based on the DateMask properties settings and the current locale setting, the component gets a date pattern ( like "d/MM/yy" ) and translates this pattern to a mask.

Use a JDateMaskedField to:

- limit the type of text that can be entered in a text field.
- display text captured from the keyboard.
- edit a line of text.
- post an [event](#) based on text input from the keyboard.

### Properties

Use the [DateMask properties](#) to set the date or time type, the formatting style of the date or time, and whether the field should recognize literals (date and time separators) passed to it.



**Note:** Changes made to DateMask properties are not reflected at design-time.

### Coding the component

In project source code, use this syntax to create a new `JDateMaskedField`:

```
JDateMaskedField1 = new com.symantec.itools.swing.JDateMaskedField();  
getContentPane().add(JDateMaskedField1);
```

Part	Type	Description
<code>JDateMaskedField1</code>	<code>JDateMaskedField</code>	The variable name used to identify the component in code.

## JDateMaskedField Properties

<a href="#">AlignmentX</a>	<a href="#">FocusTraversable</a>
<a href="#">AlignmentY</a>	<a href="#">Font</a>
<a href="#">Autoscrolls</a>	<a href="#">Foreground</a>
<a href="#">Background</a>	<a href="#">Highlighter</a>
<a href="#">Border</a>	<a href="#">HorizontalAlignment</a>
<a href="#">Bounds</a>	<a href="#">Insets</a>
<a href="#">Caret</a>	<a href="#">Keymap</a>
<a href="#">CaretColor</a>	<a href="#">ManagingFocus</a>
<a href="#">Class</a>	<a href="#">Name</a>
<a href="#">Columns</a>	<a href="#">NextFocusableComponent</a>
<a href="#">Cursor</a>	<a href="#">Opaque</a>
<a href="#">DateMask</a>	<a href="#">OptimizedDrawingEnabled</a>
<a href="#">DebugGraphicsOptions</a>	<a href="#">PaintingTile</a>
<a href="#">Default Background</a>	<a href="#">RequestFocusEnabled</a>
<a href="#">Default Font</a>	<a href="#">ScrollOffset</a>
<a href="#">Default Foreground</a>	<a href="#">Selection</a>
<a href="#">DisabledTextColor</a>	<a href="#">Text</a>
<a href="#">DoubleBuffered</a>	<a href="#">ToolTipText</a>
<a href="#">Editable</a>	<a href="#">UIClassID</a>
<a href="#">Enabled</a>	<a href="#">ValidateRoot</a>
<a href="#">FocusAccelerator</a>	<a href="#">Visible</a>
<a href="#">FocusCycleRoot</a>	<a href="#">VisibleRect</a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A dialog box that can be shown and dismissed apart from its parent window. When you add a JDialog component to your project, a new Form Designer window opens. Add JFC components to the JDialog by dragging components and containers to the Form Designer as necessary.

Dialog boxes are intended to be temporary window objects. They present specific timely information to your user, or they allow your user to set options for the current operation. A JDialog can be modal or non-modal. A modal JDialog receives all mouse, keyboard, and focus events that occur while it is open. The user must interact with a modal JDialog window and dismiss the window before returning to its parent window.

Use the Interaction Wizard to open a JDialog in response to an action such as a button press or a menu selection. The Interaction Wizard allows you to open the dialog as modal or non-modal.

You can set whether the JDialog will be modal or non-modal using the [Modal property](#).

Use JDialog to:

- prompt the user for information or to make a decision.
- interrupt program processing until your user has provided information or read the message.
- display status on an activity taking place.

Dialog boxes commonly contain the following GUI elements: JCheckBox, JComboBox, JButton, JLabel, JList, JPasswordField, JTextArea, and more.

Dialogs are empty when created. You must add components, set properties, and add interactions as necessary to create functionality.

### Coding the component

To create a JDialog window in project source code, use this syntax:

*JDialog1* = public JDialog(Frame *owner*,String *title*,boolean *modal*)

Part	Type	Description
JDialog1	Dialog	The variable name used to refer to the component in code.
owner	Frame	The main application window from which the Dialog can be shown.
title	String	The text to display in the dialog box's title bar.
modal	boolean	Whether the dialog is modal or not.

The settings for *modal* are:

Value	Description
true	The dialog box blocks input to other windows. While it is visible, the dialog box retrieves all user input.
false	The dialog box is modeless. Other windows can be activated and used while the dialog box is open.

## **JDialog Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DefaultCloseOperation](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Layout](#)

[Modal](#)

[Name](#)

[Resizable](#)

[Title](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A text pane to edit various kinds of content, such as HTML and RTF. This uses implementations of the [EditorKit](#) to accomplish its behavior. This component effectively morphs into the proper kind of text editor for the kind of content it is given.

### Properties

Use the [ContentType property](#) to set which text format you are using. If the content is set to a new URL, its type is used to determine the EditorKit that should be used to load the content. Plain text is the default content type.

### Coding the Component

In project source code, use this syntax to create a new JEditorPane:

```
JEditorPanel = new com.sun.java.swing.JEditorPane();  
getContentPane().add(JEditorPanel);
```

Part	Type	Description
JEditorPanel	JEditorPane	The variable name used to identify the editor pane in code.

## JEditorPane Properties

<a href="#"><u>AlignmentX</u></a>	<a href="#"><u>Foreground</u></a>
<a href="#"><u>AlignmentY</u></a>	<a href="#"><u>Highlighter</u></a>
<a href="#"><u>Autoscrolls</u></a>	<a href="#"><u>Default Background</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>Default Font</u></a>
<a href="#"><u>Border</u></a>	<a href="#"><u>Default Foreground</u></a>
<a href="#"><u>Bounds</u></a>	<a href="#"><u>Insets</u></a>
<a href="#"><u>Caret</u></a>	<a href="#"><u>Keymap</u></a>
<a href="#"><u>CaretColor</u></a>	<a href="#"><u>Layout</u></a>
<a href="#"><u>Class</u></a>	<a href="#"><u>ManagingFocus</u></a>
<a href="#"><u>ContentType</u></a>	<a href="#"><u>Name</u></a>
<a href="#"><u>Cursor</u></a>	<a href="#"><u>NextFocusableComponent</u></a>
<a href="#"><u>DebugGraphicsOptions</u></a>	<a href="#"><u>Opaque</u></a>
<a href="#"><u>DisabledTextColor</u></a>	<a href="#"><u>OptimizedDrawingEnabled</u></a>
<a href="#"><u>Document</u></a>	<a href="#"><u>PaintingTile</u></a>
<a href="#"><u>DoubleBuffered</u></a>	<a href="#"><u>RequestFocusEnabled</u></a>
<a href="#"><u>Enabled</u></a>	<a href="#"><u>Selection</u></a>
<a href="#"><u>Editable</u></a>	<a href="#"><u>Text</u></a>
<a href="#"><u>EditorKit</u></a>	<a href="#"><u>ToolTipText</u></a>
<a href="#"><u>FocusAccelerator</u></a>	<a href="#"><u>UIClassID</u></a>
<a href="#"><u>FocusCycleRoot</u></a>	<a href="#"><u>ValidateRoot</u></a>
<a href="#"><u>FocusTraversable</u></a>	<a href="#"><u>Visible</u></a>
<a href="#"><u>Font</u></a>	<a href="#"><u>VisibleRect</u></a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also


 See Also

### Description

A frame is a top-level window that supports a title bar and menu bar. Visual Cafe uses the frame component to build applications. Frames cannot be placed inside another component.

Use a frame whenever you need to:

- build a GUI for an application. JFrame is the parent container for application GUIs.
- build a container that supports a menu bar.

To create a JFrame, select File  New Project and choose JFC Application. Visual Cafe automatically provides a JFrame and a main method, and updates both Frame methods and property and event defaults as necessary.

You can also drag a JFrame from the Component Palette to the Form Designer window, or from the Component Library to the Project window. Adding a JFrame from the Palette or the Component Library may require you to provide a `main` method and override other JFrame methods manually.

When a JFrame is active, it receives all mouse, keyboard, and focus events that occur over it.

### Properties

To add a [JMenuBar component](#) to a JFrame, drag the JMenuBar icon to the Form Designer window. Use the [JMenuBar property](#) to select any JMenuBar components you want to make active in this JFrame.

### Coding the component

In project source code, use this syntax to create a new JFrame:

```
public class JFrame1 extends com.sun.java.swing.JFrame
```

Part	Type	Description
JFrame1	JFrame	The variable name used to identify the frame in code.

## **JFrame Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DefaultCloseOperation](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[JMenuBar](#)

[Layout](#)

[Name](#)

[Resizable](#)

[Title](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A lightweight object that provides many of the features of a native frame, including dragging, closing, becoming an icon, resizing, title display, and support for a menu bar.

The `JInternalFrame` has a 'contentPane' that is used to describe the area of the component where normal child components should be placed.

### Coding the component

In project source code, use this syntax to create a new `JInternalFrame`:

```
public class JInternalFrame1 extends com.sun.java.swing.JInternalFrame
```

Part	Type	Description
JInternalFrame1	JInternalFrame	The variable name used to identify the internal frame in code.

## **InternalFrame Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Closable](#)  
[Closed](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[DefaultCloseOperation](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Iconifiable](#)  
[Insets](#)  
[JMenuBarJ](#)  
[Layer](#)  
[Layout](#)  
[ManagingFocus](#)  
[Maximizable](#)  
[Maximum](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Resizable](#)  
[Selected](#)  
[Title](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A text string that is usually attached to another visual component (like a JTextField or JButton).

Use this component to:

- label a JTextArea, or any other Swing component that does not have a viewable label.
- display static, explanatory text.
- display a simple status area.

An application or applet can change the label text string, but a user cannot edit it.

### Properties

To specify the label text, type the text in the [Text property](#).

### Coding the component

In code, use this syntax to create a new JLabel:

```
JLabel1 = new com.sun.java.swing.JLabel (label, alignment);
```

Part	Type	Description
JLabel1	Label	The variable name you can use to identify the component in code.
label	String	The text to display in the label.
alignment	int	Positions text within the label bounding box. If no alignment is specified, the label is automatically left-aligned. Legal values are LEFT, RIGHT, and CENTER.

## **JLabel Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DisplayedMnemonic](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[HorizontalAlignment](#)  
[HorizontalTextPosition](#)  
[IconTextGap](#)  
[Icons](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[LabelFor](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[VerticalAlignment](#)  
[VerticalTextPosition](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A box containing a scrollable and selectable list of items.

When a JList item is selected or deselected, an ItemEvent is generated (see [item state changed](#)). When a user double-clicks on an item, the DbClicked action performed event is generated (see [action performed](#)). Use the Interaction Wizard to perform some action triggered by these (and other) events.

Drag a JList into a JScrollPane component to make a display with scroll bars when the display area is not large enough to display all the list items.

### Properties

To specify the initial items to use to populate the JList, use the [Items property](#) in the StringListModel. To include more than one item in the list, press CTRL+ENTER after typing each item.

To allow selection of multiple items in a list use the [SelectionMode property](#) in JList. If you want to select items from the list that are not adjacent, set the SelectionMode to MULTIPLE\_INTERVAL\_SELECTION.

Use the [Border property](#) to change the border type.

### Coding the component

To create a new JList component, use this syntax:

```
JList1 = new com.sun.java.swing.JList();  
JList1.setModel(stringListModel1);
```

To add items to a JList component, use this syntax:

```
String[] tempString = new String[3];  
tempString[0] = "1";  
tempString[1] = "2";  
tempString[2] = "3";  
stringListModel1.setItems(tempString)
```

To create a scrolling list, add the JList component to a JScrollPane component using this syntax:

```
getContentPane().add(JScrollPane1);  
JScrollPane1.getViewport().add(JList1);
```

Part	Type	Description
JList1	JList	The variable name used to refer to the component in code.
stringListModel1	StringListModel	The model assigned to JList.
JScrollPane1	JScrollPane	The variable name used to refer to the JScrollPane in code.

## JList Properties

<a href="#"><u>AlignmentX</u></a>	<a href="#"><u>LeadSelectionIndex</u></a>
<a href="#"><u>AlignmentY</u></a>	<a href="#"><u>ManagingFocus</u></a>
<a href="#"><u>Autoscrolls</u></a>	<a href="#"><u>MaxSelectionIndex</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>MinSelectionIndex</u></a>
<a href="#"><u>Border</u></a>	<a href="#"><u>Model</u></a>
<a href="#"><u>Bounds</u></a>	<a href="#"><u>Name</u></a>
<a href="#"><u>CellRenderer</u></a>	<a href="#"><u>NextFocusableComponent</u></a>
<a href="#"><u>Class</u></a>	<a href="#"><u>Opaque</u></a>
<a href="#"><u>Cursor</u></a>	<a href="#"><u>OptimizedDrawingEnabled</u></a>
<a href="#"><u>DebugGraphicsOptions</u></a>	<a href="#"><u>PaintingTile</u></a>
<a href="#"><u>DoubleBuffered</u></a>	<a href="#"><u>PrototypeCellValue</u></a>
<a href="#"><u>Enabled</u></a>	<a href="#"><u>RequestFocusEnabled</u></a>
<a href="#"><u>FirstVisibleIndex</u></a>	<a href="#"><u>SelectedIndex</u></a>
<a href="#"><u>FixedCellHeight</u></a>	<a href="#"><u>SelectionBackground</u></a>
<a href="#"><u>FixedCellWidth</u></a>	<a href="#"><u>SelectionEmpty</u></a>
<a href="#"><u>FocusCycleRoot</u></a>	<a href="#"><u>SelectionForeground</u></a>
<a href="#"><u>FocusTraversable</u></a>	<a href="#"><u>SelectionMode</u></a>
<a href="#"><u>Font</u></a>	<a href="#"><u>SelectionModel</u></a>
<a href="#"><u>Foreground</u></a>	<a href="#"><u>ToolTipText</u></a>
<a href="#"><u>Default Background</u></a>	<a href="#"><u>UIClassID</u></a>
<a href="#"><u>Default Font</u></a>	<a href="#"><u>ValidateRoot</u></a>
<a href="#"><u>Default Foreground</u></a>	<a href="#"><u>ValuesAdjusting</u></a>
<a href="#"><u>Insets</u></a>	<a href="#"><u>Visible</u></a>
<a href="#"><u>LastVisibleIndex</u></a>	<a href="#"><u>VisibleRect</u></a>
	<a href="#"><u>VisibleRowCount</u></a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A component that allows a user to create a list of items, select items from that list and store them in another list, and use the selected items elsewhere in an application. Use this component when you want a user to select a subset of items from a master list, such as picking which toolbar buttons a user wants to display.

The component has three major parts: Available List, Selected List, and buttons to control movement between the two lists. Available List and Selected List are similar to a JList component. Both items use a border and a model component.

 See Also

**Note:** When you select the same model type for the Available and Selected Lists, use a different instance of the model for each one.

The buttons control whether an item is added to or removed from the Selected List and have the following functionality: Add, Add All, Remove, and Clear.

- n Add adds a highlighted item in the Available List to the Selected List.
- n Add All adds all the items from the Available List to the Selected List.
- n Remove deletes the highlighted item from the Selected List and places it back into the Available List.
- n Clear deletes all the items in the Selected List and places them back into the Available List.

 See Also

**Note:** Double-Clicking a single item in the Available List is the same as using the Add button. Double-Clicking a single item in the Selected List is the same as using the Remove button.

### Properties

Use the [Button Labels properties](#) to change the labels on the buttons that control movement between the Available and Selected Lists.

Use the [Models properties](#) to set a model for the Available and Selected Lists.

Use the [Borders properties](#) to set the borders around the Available and Selected Lists.

### Coding the component

To create a new JListChooser component, use this syntax:

```
JListChooser1 = new com.symantec.itools.swing.JListChooser();
```

Part	Type	Description
JListChooser1	JListChooser	The variable name used to refer to the component in code.

## **JListChooser Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Borders](#)  
[Bounds](#)  
[Button Labels](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[Models](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

A JComboBox that allows a user to switch to and from the three default Look and Feel types available in JFC.

When you include a JLookAndFeelComboBox in a container, make sure that the attached list box doesn't cover up other components at run time.

**Properties**

To use a different font in the JLookAndFeelComboBox, set the [Font property](#).

**Coding the component**

In project source code, use this syntax to create a new JLookAndFeelComboBox:

```
JLookAndFeelComboBox1 = new com.symantec.itools.swing.JLookAndFeelComboBox();
getContentPane().add(JLookAndFeelComboBox1);
```

Part	Type	Description
JLookAndFeelComboBox1	JLookAndFeelComboBox	The variable name used to identify the Look and Feel Combo Box in code.

## **JLookAndFeelComboBox Properties**

[Action Command](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Editable](#)  
[Editor](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ItemCount](#)  
[KeySelectionManager](#)  
[LightWeightPopupEnabled](#)  
[ManagingFocus](#)  
[MaximumRowCount](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[Renderer](#)  
[RequestFocusEnabled](#)  
[SelectedIndex](#)  
[SelectedItem](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

See Also

See Also

See Also

Description

A text field which has edit masking logic applied to the user input. The masking logic ensures that a match between the input character and the relevant filter for the cursor position is made.

Use a JMaskedTextField to:

- limit the type of text that can be entered in a text field.
- display text captured from the keyboard.
- edit a line of text.
- post an [event](#) based on text input from the keyboard.

Properties

To specify the required format for the text entry, use the [Mask property](#).

To specify a text string to display in the field by default, use [Text property](#).

To make the default text editable, use the [Editable property](#). You do not need to write code or use the Interaction Wizard to connect the JMaskedTextField component with any other component to display text.

Coding the component

In project source code, use this syntax to create a new JMaskedTextField:

```
JMaskedTextField1 = new symantec.itools.swing.JMaskedTextField();  
add(JMaskedTextField1);
```

Part	Type	Description
JMaskedTextField1	JMaskedTextField	The variable name used to refer to the component in code.

## JMaskedTextField Properties

<a href="#"><u>AlignmentX</u></a>	<a href="#"><u>Default Background</u></a>
<a href="#"><u>AlignmentY</u></a>	<a href="#"><u>Default Font</u></a>
<a href="#"><u>Autoscrolls</u></a>	<a href="#"><u>Default Foreground</u></a>
<a href="#"><u>Background</u></a>	<a href="#"><u>Insets</u></a>
<a href="#"><u>Border</u></a>	<a href="#"><u>Keymap</u></a>
<a href="#"><u>Bounds</u></a>	<a href="#"><u>ManagingFocus</u></a>
<a href="#"><u>Caret</u></a>	<a href="#"><u>Margin</u></a>
<a href="#"><u>CaretColor</u></a>	<a href="#"><u>Mask</u></a>
<a href="#"><u>Class</u></a>	<a href="#"><u>Name</u></a>
<a href="#"><u>Columns</u></a>	<a href="#"><u>NextFocusableComponent</u></a>
<a href="#"><u>Cursor</u></a>	<a href="#"><u>Opaque</u></a>
<a href="#"><u>DebugGraphicsOptions</u></a>	<a href="#"><u>OptimizedDrawingEnabled</u></a>
<a href="#"><u>DisabledTextColor</u></a>	<a href="#"><u>PaintingTile</u></a>
<a href="#"><u>Document</u></a>	<a href="#"><u>PreferredScrollableViewportSize</u></a>
<a href="#"><u>DoubleBuffered</u></a>	<a href="#"><u>RequestFocusEnabled</u></a>
<a href="#"><u>Editable</u></a>	<a href="#"><u>ScrollOffset</u></a>
<a href="#"><u>Enabled</u></a>	<a href="#"><u>ScrollableTracksViewportHeight</u></a>
<a href="#"><u>FocusAccelerator</u></a>	<a href="#"><u>ScrollableTracksViewportWidth</u></a>
<a href="#"><u>FocusCycleRoot</u></a>	<a href="#"><u>Selection</u></a>
<a href="#"><u>FocusTraversable</u></a>	<a href="#"><u>Text</u></a>
<a href="#"><u>Font</u></a>	<a href="#"><u>ToolTipText</u></a>
<a href="#"><u>Foreground</u></a>	<a href="#"><u>UIClassID</u></a>
<a href="#"><u>Highlighter</u></a>	<a href="#"><u>ValidateRoot</u></a>
<a href="#"><u>HorizontalAlignment</u></a>	<a href="#"><u>Visible</u></a>
	<a href="#"><u>VisibleRect</u></a>

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Use this component to create a drop-down menu containing a list of commands.

Menus must be placed within a [JMenuBar component](#).

Visual Cafe provides a Menu Designer to help you create menus.

See: [Adding Menus to a Form](#)

### Properties

Use the [Mnemonic property](#) to set modifier keys that are associated with the menu.

Use the [Icons properties](#) to display images in the menu.

### Coding the component

Each item in a menu must belong to the [JMenuItem](#) class. This can be an instance of JMenuItem, a submenu (an instance of JMenu), a chec box (an instance of [JCheckboxMenuItem](#)) or a radio button (an instance of [JRadioButtonMenuItem](#) ).

Create a new menu using the syntax below. You can create a menu with a specified label:

```
JMenuBar JMenuBar1=new JMenuBar();  
JMenu JMenu1=new JMenu("Label");  
JMenuItem JMenuItem1=new JMenuItem ("menuitem");
```

Part	Type	Description
JMenu1	JMenu	The variable name used to reference the component in code.
Label	String	The text that displays as the menu name on the menu bar.
MenuItem	String	The text that displays as the menu item name in the menu.

See Also: [JMenuBar component](#).

See Also: [JFrame component](#)

## JMenu Properties

[Action Command](#)  
[Alignment/Position](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[Delay](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusPainted](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Insets](#)  
[ManagingFocus](#)  
[MenuComponentCount](#)  
[Mnemonic](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[RolloverEnabled](#)  
[Selected](#)  
[Text](#)  
[ToolTipText](#)  
[TopLevelMenu](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Creates a platform-specific horizontal bar containing a list of menus. Use JMenuBar to group your menus within a horizontal bar.

JApplet and JFrame containers can accept JMenuBar components. To display a menu bar in an application, add the JMenuBar component to a JFrame window.

See: [Adding Menus to a Form](#)

### Runtime modification

At run time, you can display a different menu bar by calling the [setJMenuBar](#) method.

### Coding the component

To create a new menu bar in source code, use this syntax:

```
JMenuBar1 = new com.sun.java.swing.JMenuBar();  
setJMenuBar(JMenuBar1);
```

 See Also

**Note:** You may substitute the following line of code for setJMenuBar:

```
getContentPane().setLayout(new BorderLayout(0,0));  
getContentPane().add(JMenuBar1, BorderLayout.Region)
```

To add a JMenu to the menu bar in source code, use this syntax:

```
JMenuBar1.add(new JMenu("Label"));
```

Part	Type	Description
JMenuBar1	JMenuBar	The variable name used to reference the component in code.
JMenu	JMenu	Instantiates this component in code.
Label	String	The text that displays as the menu name on the menu bar.
Region	Region	One of five regions in a container; center, north, south, east, or west

## JMenuBar Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[MenuCount](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Selected](#)  
[SelectionModel](#) property  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

An item in a menu. Menu items are the commands that appear on a pull-down menu.

Visual Cafe creates menu items for you automatically in the Menu Editor as you edit a [JMenuBar component](#).

See: [Adding Menus to a Form](#)

 See Also

**Tip:** When using the Interaction Wizard to connect a JMenuItem component to another component, it may be easiest to start the interaction line from the JMenuItem in the Menu Designer and drag the line to the appropriate component in the Project window.

To create a menu separator, insert a [JSeparator component](#) between the items you want separated.

### Properties

Use the [Mnemonic property](#) to set modifier keys that are associated with the menu.

Use the [Icons properties](#) to display images in the menu.

Use the [Accelerator property](#) to create a shortcut key for the component.

### Coding the component

The Java language does not require you to create a variable to reference a menu item. Instead, you can refer to the menu item by its label.

To create a menu item with the specified label in project source code, use this syntax:

```
JMenuItem1 = new com.sun.java.swing.JMenuItem();
JMenuItem1.setText("label");
JMenu1.add(JMenuItem1);
```

Part	Type	Description
JMenuItem1	JMenuItem	The variable name used to reference the component in code.
label	String	The label for the menu item, or null for an unlabeled menu item.
JMenu1	JMenu	The variable name of the menu component to which you are adding the menu item.

To create a menu item labeled “Undo” and add it to a JMenu with the variable name editMenu, use this syntax:

```
JMenuItem1.setText("Undo");
editMenu.add(JMenuItem1);
```

Alternatively you can make the editMenu automatically create the JMenuItem for you:

```
editMenu.add("Undo");
```

## **JMenuItem Properties**

[Accelerator](#)  
[Action Command](#)  
[Alignment/Position](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusPainted](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Insets](#)  
[ManagingFocus](#)  
[Mnemonic](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[RolloverEnabled](#)  
[Selected](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A standard dialog box that prompts users for a value or informs them of something.

Most uses of JOptionPane are one-line calls to one of the static showXXXDialog methods shown below:

showConfirmDialog	Asks a confirming question, like yes/no/cancel.
ShowInputDialog	Prompt for some input.
ShowMessageDialog	Tell the user about something that has happened.
ShowOptionDialog	Allows any combination of the above three.

Each of these methods also comes in a showInternalXXX flavor, which uses an internal frame to hold the dialog box (see [JInternalFrame](#) ).

All dialogs are modal. Each showXXXDialog method blocks the current thread until the user's interaction is complete.

### Properties

The [MessageType property](#) defines the style of the message. The look&feel manager may lay out the dialog differently depending on this value, and will often provide a default icon.

The [OptionType property](#) defines the set of option buttons that appear at the bottom of the dialog box.

### Coding the component

Show an error dialog that displays the message, 'alert':

```
JOptionPane.showMessageDialog(null, "alert", "alert", ERROR_MESSAGE);
```

Show an information panel with the options yes/no and message 'choose one':

```
JOptionPane.showConfirmDialog(null, "choose one", "choose one", YES_NO);
```

Show a dialog asking the user to type in a String:

```
String inputValue = JOptionPane.showInputDialog("Please input a value");
```

To create and use a JOptionPane directly, the standard pattern is roughly as follows:

```
JOptionPane pane = new JOptionPane(arguments);
pane.set.Xxxx(...); // Configure
JDialog dialog = pane.createDialog(parentComponent, title);
dialog.show();
Object selectedValue = pane.getValue();
if(selectedValue == null)
    return CLOSED_OPTION;
//If there is not an array of option buttons:
if(options == null) {
    if(selectedValue instanceof Integer)
        return ((Integer)selectedValue).intValue();
    return CLOSED_OPTION;
}
//If there is an array of option buttons:
for(int counter = 0, maxCounter = options.length;
```

```
        counter < maxCounter; counter++) {  
            if(options[counter].equals(selectedValue))  
                return counter;  
        }  
        return CLOSED_OPTION;
```

## **JOptionPane Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[JMenuBar](#)

[Layout](#)

[Name](#)

[Visible](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A generic container. A panel can contain other visual components and panels. Panels do not support menus.

Use JPanel to:

- create a subcontainer that organizes container space within an JApplet, JFrame, JTabPanel, JDialog, etc. This simplifies your component layout task.
- form a basis for a layout style.
- hold other containers.

### Properties

Use the [Layout property](#) to set the desired layout manager.

### Coding the component

To create a new JPanel, use the following syntax:

```
JPanel1= new com.sun.java.swing.JPanel();
```

Part	Type	Description
------	------	-------------

JPanel1	JPanel	The variable name used to reference the component in code.
---------	--------	--

## **JPanel Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Extends JTextField.

Allows the editing of a single line of text where the view indicates something was typed, but does not show the original characters. It is provided separately to make it easier to safely change the UI for the JTextField without affecting password entries.

### Properties

Use the [Text property](#) to enter a password.

### Coding the component

To create a new JPasswordField, use the following syntax:

```
JPasswordField1 = new com.sun.java.swing.JPasswordField();  
JPasswordField1.setEchoChar('*');
```

---

Part	Type	Description
JPasswordField1	JPasswordField	The variable name used to reference the component in code.
*	String	The echo character to display.

## **JPasswordField Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Caret](#)  
[CaretColor](#)  
[Class](#)  
[Columns](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DisabledTextColor](#)  
[Document](#)  
[DoubleBuffered](#)  
[EchoChar](#)  
[Editable](#)  
[Enabled](#)  
[FocusAccelerator](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Highlighter](#)  
[HorizontalAlignment](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Keymap](#)  
[ManagingFocus](#)  
[Margin](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[PreferredScrollableViewportSize](#)  
[RequestFocusEnabled](#)  
[ScrollOffset](#)  
[ScrollableTracksViewportHeight](#)  
[ScrollableTracksViewportWidth](#)  
[Selection](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A small window which pops up and displays a series of choices. A JPopupMenu is used for the menu that appears when the user selects an item on the menu bar. It is also used for a submenu that appears when the user selects a menu item that activates it.

A JPopupMenu can also be used anywhere else you want a menu to appear – for example, when the user right-clicks in a specified area.

A JPopupMenu holds [JMenuItems](#) like [JMenu](#).

### Properties

Use the [Invoker property](#) to set what component invokes the popup menu.

Use [Label](#) to display a title for the popup menu in the UI.

Set [LightWeightPopupEnabled](#) to true use a lightweight component to display the popup menu. Set the property to false if any part of the popup menu is not displayed completely on top of an existing JFC window.

### Coding the component

To create a new JPopupMenu, use the following syntax:

```
JPopupMenu l = new com.sun.java.swing.JPopupMenu();
```

---

Part	Type	Description
JPopupMenu1	JPopupMenu	The variable name used to reference the component in code.

---

## **JPopupMenu Properties**

[AlignmentX](#)

[AlignmentY](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[Insets](#)

[Invoker](#)

[Label](#)

[LightWeightPopupEnabled](#)

[ManagingFocus](#)

[Margin](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[SelectionMode](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[Visible](#)

[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A bar that displays the percentage completed of a particular process. It is commonly used to indicate the percentage completed of a printing task or program installation. The bar has no user interaction.

### Properties

To set the percent complete for a process, use the [Value property](#). For example, a value of 33 would indicate 33% complete.

### Coding the component

Use the following syntax to create a JProgressBar:

```
JProgressBar1=new com.sun.java.swing.JProgressBar();
```

While a process is underway, periodically note the percentage completed by using a formula like  $i*100/\text{max}$  with the setValue method.

Part	Type	Description
JProgressBar1	JProgressBar	The variable name used to reference the component in code.

## JProgressBar Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[Maximum](#)  
[Minimum](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[Orientation](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Value](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Extends JToggleButton.

A round button used to select one choice from several options.

Use a JRadioButton to:

- manage a selection-deselection user task.
- display a selection choice.
- post an event to another component.
- determine that only one choice in a set of choices is selected.

Typically, a JRadioButton will be a member of a group. In a group of radio buttons, only one can be selected at a time. When the user selects one radio button, the other radio buttons in the group are automatically deselected.

Use the Interaction Wizard to trigger an action when a JRadioButton is clicked.

### Properties

To set the default state, checked or cleared, of the JRadioButton, use the [Selected](#).

### Coding the component

Use this syntax to create a radio button:

```
JRadioButton1 = new com.sun.java.swing.JRadioButton("label");
```

Part	Type	Description
JRadioButton1	RadioButton	The variable name used to reference the component in code.
label	String	The text to display next to the component.

## JRadioButton Properties

[Alignment/Position](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusPainted](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[HorizontalAlignment](#)

[HorizontalTextPosition](#)

[Icons](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Insets](#)

[Layout](#)

[ManagingFocus](#)

[Mnemonic](#)

[Model](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[RolloverEnabled](#)

[Selected](#)

[Text](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[VerticalAlignment](#)

[VerticalTextPosition](#)

[Visible](#)

[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Creates a radio button that can be included in a menu. Clicking the associated menu item toggles its state between checked or cleared.

Use `JRadioButtonMenuItem` whenever you include a menu item that you want to toggle between on and off.

Use the Interaction Wizard to trigger an action when the `JRadioButtonMenuItem` is checked or unchecked.

### Properties

Use the [Selected](#) to set the initial menu state, either checked or cleared.

The [Text](#) sets the menu item text.

### Coding the component

The Java language does not require you to create a variable to reference a menu item. Instead, you can refer to the menu item by its label.

In project source code, use this syntax to create a new `JRadioButtonMenuItem`:

```
JRadioButtonMenuItem1 = new com.sun.java.swing.JRadioButtonMenuItem();  
JRadioButtonMenuItem1. setText("jradioButtonMenuItem");
```

To add a check box menu item to an existing menu, call the menu component's `add` method:

```
JMenu1.add(JRadioButtonMenuItem1);
```

Part	Type	Description
JRadioButtonMenuItem1	JRadioButtonMenuItem	The variable name used to reference the component in code.
Text	String	The label for the checkbox menu item, or null for an unlabeled menu item.
JjMenu1	Menu	The variable name of the menu component to which you are adding the menu item.

## JRadioButtonMenuItem Properties

[Accelerator](#)  
[Action Command](#)  
[Alignment/Position](#)  
[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[BorderPainted](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusPainted](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Icons](#)  
[Insets](#)  
[ManagingFocus](#)  
[Mnemonic](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[RolloverEnabled](#)  
[Selected](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

## Description

Extends JComponent.

The JRootPane component is made up of several components. In order they are:

```
JRootPane
    glassPane
    layeredPane
        [menuBar]
    contentPane
```

The menuBar component is optional and may or may not exist at any time. The layeredPane, contentPane, and glassPane will always be available.

The glassPane is always added as the first child of the JRootPane. This allows the glass component to get in the way of all mouse events, as well as providing a convenient place to draw above all other components. The glassPane is always non-null. Attempting to set it to null generates an exception. By default the glassPane is not visible.

The layeredPane contains the menuBar and the contentPane. Any methods used are normally sent to the contentPane instead of to the JRootPane. The contentPane is always non-null. Attempting to set it to null generates an exception. The default contentPane has a BorderLayout manager set on it.

If a JMenuBar component is set on the JRootPane, it is positioned along the upper edge of the frame. The contentPane is adjusted in location and size to fill the remaining area.

The layeredPane is an instance of the JLayeredPane class. Its purpose is to be the parent of all children of the JRootPane. It provides the ability to add components at several layers. This is very useful when working with menu popups, dialog boxes, or during dragging. The layeredPane is always non-null. Attempting to set it to null generates an exception.

The JRootPane uses a custom LayoutManager. If you replace the LayoutManager of the JRootPane, you are responsible for managing all of these views.

## Coding the component

In project source code, use this syntax to create a newJRootPane:

```
JRootPane1 = new com.sun.java.swing.JRootPane();
JRootPane1.getContentPane().setLayout(new BorderLayout(0,0));
getContentPane().add(JRootPane1);
```

Part	Type	Description
JRootPane1	JRootPane1	The variable name used to identify the component in code.

## JRootPane Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DefaultButton](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[JMenuBarJ](#)  
[Layout](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

Scrollbar components preset to display left-to-right and up-down, respectively.

A scrollbar is a component used when the contents of a window or box take up more space than is allocated to that window or box. This commonly occurs when a large amount of text is displayed in a window. The scrollbar indicates three things: the total length of the contents (corresponding to the length of the scrollbar), the location of the contents shown in the window (corresponding to the position of the scrollbar thumb), and the amount of the contents shown in the window (corresponding to the length of the scrollbar thumb). A different part of the contents may be shown in the window by moving the scrollbar's thumb. The thumb can be moved by dragging, clicking an arrow button, or by clicking between the thumb and an arrow.

Use a JScrollbar to:

- scroll through a data buffer that is too large to display in an associated view window.
- respond to a user event.
- send an action event to another visual component.

Use the Interaction Wizard to trigger an action when the scrollbar's value has changed. A typical event might be painting the appropriate part of the data a window is displaying into that window, thus making the scrollbar "scroll" the data in the window.

### Properties

To change the orientation of a JScrollbar, set the [Orientation property](#) to VERTICAL or HORIZONTAL.

To set the location of the scroll thumb in the scroll bar, set the scroll bar's [Value property](#). The [Minimum](#) and [Maximum](#) properties set the range of values the scrollbar may have.

In code, you can move the scroll thumb to its minimum position by calling the [setValue](#) method for the scrollbar and use the [Minimum property](#) setting as the parameter.

### Coding the component

The example below instantiates a JScrollbar in project source code:

```
JScrollbar1 = new com.sun.java.swing.JScrollbar();  
getContentPane().add(JScrollbar1);
```

Part	Type	Description
JScrollbar1	JScrollbar	The variable name used to reference the component in code.

## **JScrollBar Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[BlockIncrement](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Maximum](#)  
[Minimum](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[Orientation](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Value](#)  
[ValuesAdjusting](#)  
[Visible](#)  
[VisibleAmount](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

A specialized container that manages a viewport, optional vertical and horizontal scrollbars, and optional row and column heading viewports.

The most common use of JScrollPane is to hold a JTable component. Placing a JTable component in a JScrollPane allows you to see the column headers in the JTable at design time.

**Properties**

To turn on horizontal or vertical scrolling, use the [ScrollBar Policies properties](#) .

**Coding the component**

In project source code, use this syntax to create a new JScrollPane:

```
JScrollPane1 = new com.sun.java.swing.JScrollPane();
```

Part	Type	Description
JScrollPane1	JScrollPane	The variable name used to identify the component in code.

## **JScrollPane Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[ScrollBarPolicies](#)  
[ViewportBorder](#)  
[Visible](#)  
[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Determines whether a menu item is a separator. A separator displays as a solid line in the menu and cannot be selected at run time.

### Coding the component

Use the following syntax to create a separator:

```
JSeparator1 = new com.sun.java.swing.JSeparator();  
JMenu1.add(jSeparator1);
```

Part	Type	Description
<i>JSeparator1</i>	<i>JSeparator</i>	The variable name used to reference the component in code.
<i>JMenu1</i>	<i>Menu</i>	The variable name of the menu component to which you are adding the menu item.

## **JSeparator Properties**

[AlignmentX](#)

[AlignmentY](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[Insets](#)

[ManagingFocus](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[Visible](#)

[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

A horizontal or vertical slider component.

A slider is used to select one value from a continuous range of values. It has a movable thumb in front of a gauge with ticks marks on it.

### Properties

To specify the range of values that the slider indicates, change the [Minimum](#) and the [Maximum](#).

To change the number of tick marks on the slider, set the [MajorTickSpacing property](#) and the [MinorTickSpacing property](#).

Use the [Value property](#) to set the initial value of the slider.

Use the [PaintTicks property](#) to display the tick marks under the slider.

### Coding the component

In project source code, use this syntax to create a new JSlider:

```
JSlider1 = new com.sun.java.swing.JSlider();
```

Part	Type	Description
------	------	-------------

JSlider1	JSlider	The variable name used to identify the component in code.
----------	---------	---

## JSlider Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[Extent](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Inverted](#)  
[Layout](#)  
[MajorTickSpacing](#)  
[ManagingFocus](#)  
[Maximum](#)  
[Minimum](#)  
[MinorTickSpacing](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[Orientation property \(JProgressBar, JScrollPane, JSlider\)](#)  
[PaintLabels](#)  
[PaintTicks](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[SnapToTicks](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Value](#)  
[ValuesAdjusting](#)  
[Visible](#)  
[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

JSplitPane is used to divide two (and only two) components. The two components are graphically divided based on the look and feel implementation. By user intervention the two components can be interactively resized.

The two components can be aligned left to right, or top to bottom. JSplitPane will try to reposition the other component if one component changes.

To resize the components to their preferred sizes, invoke `resetPreferredSizes`.

When the user is resizing the components the minimum size of the components is used to determine the maximum/minimum position the components can be set to. Therefore, if the minimum size of the two components is greater than the size of the splitpane, the divider will not allow you to resize it.

### Properties

Change the size of the components by using the [DividerSize property](#).

### Coding the component

In project source code, use this syntax to create a new JSplitPane:

```
JSplitPanel = new com.sun.java.swing.JSplitPane();
```

Part	Type	Description
JSplitPane1	JSplitPane	The variable name used to identify the component in code.

## JSplitPane Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[ContinuousLayout](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DividerLocation](#)  
[DividerSize](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[InheritBackground](#)  
[InheritFont](#)  
[InheritForeground](#)  
[Insets](#)  
[LastDividerLocation](#)  
[ManagingFocus](#)  
[MaximumDividerLocation](#)  
[MinimumDividerLocation](#)  
[Name](#)  
[NextFocusableComponent](#)  
[OneTouchExpandable](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[Orientation](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A component which lets the user switch between a group of components by clicking on a tab with a given title and/or icon.

Tabs/components are added to a `JTabbedPane` object by using the `addTab` and `insertTab` methods. A tab is represented by an index corresponding to the position it was added in, where the first tab has an index equal to 0 and the last tab has an index equal to the tab count minus 1.

The `JTabbedPane` uses a `SingleSelectionModel` to represent the set of tab indices and the currently selected index. If the tab count is greater than 0, then there will always be a selected index, which by default will be initialized to the first tab. If the tab count is 0, then the selected index will be -1.

### Properties

Use the [Titles property](#) to create the names for each tab in the tabbed pane.

### Coding the component

In project source code, use this syntax to create a new `JTabbedPane`:

```
JTabbedPane1 = new com.sun.java.swing.JTabbedPane();
```

Part	Type	Description
<code>JTabbedPane1</code>	<code>JTabbedPane1</code>	The variable name used to identify the component in code.

## **JTabbedPane Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[SelectedComponent](#)  
[SelectedIndex](#)  
[TabCount](#)  
[TabPlacement](#)  
[TabRunCount](#)  
[Titles](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

JTable is a component that presents data in a two-dimensional table format. JTable has many features that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

JTable uses integers exclusively to refer to both the rows and the columns.

 See Also

**Note:** You must place the JTable inside a [JScrollPane](#) to see the column headers and to scroll through the table.

It is still possible to build TableModels which use identifiers, either for the columns or the rows; this can be a good way to give meaning to data values which is independent of position. The JTable simply takes a tabular range of cells and uses `getValueAt(int, int)` to retrieve and display them.

Use the [FileTableModel component](#) or [StringTableModel component](#) to enter data into the JTable.

### Properties

To set the color of the grid in a table, use the [GridColor property](#).

Use the [AutoCreateColumnsFromModel property](#) to create columns based on information from the model.

To allow the columns to size themselves based on the longest item in a column, use the [AutoResizeMode property](#).

### Coding the component

In project source code, use this syntax to create a new JTable:

```
JTable1 = new com.sun.java.swing.JTable();  
getContentPane().add(JTable1);
```

Part	Type	Description
JTable1	JTable	The variable name used to identify the component in code.

## **JTable Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[AutoCreateColumnsFromModel](#)  
[AutoResizeMode](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[CellEditor](#)  
[Class](#)  
[ColumnCount](#)  
[ColumnModel](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Editing](#)  
[EditingColumn](#)  
[EditingRow](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[GridColor](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[ManagingFocus](#)  
[Model](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[PreferredScrollableViewportSize](#)  
[RequestFocusEnabled](#)  
[RowCount](#)  
[RowHeight](#)  
[ScrollableTracksViewportHeight](#)  
[ScrollableTracksViewportWidth](#)  
[Selection](#)  
[ShowHorizontalLines](#)  
[ShowVerticalLines](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

## Description

Extends JTextComponent.

An editable multi-line display window with two scroll bars. The display window can scroll through an amount of text that is too large to display at one time.

You cannot remove the scrollbars.

Use JTextArea to:

- display large sections of text.
- edit a block of text.
- scroll through a large block of text.
- post an event based on text input from the keyboard.

Use the Interaction Wizard to trigger one or more actions when the JTextArea generates events.

To provide scrollbars in the text area, place the JTextArea component inside a [JScrollPane component](#). Use the [ScrollBarPolicies properties](#) in JScrollPane to turn on both vertical and horizontal scrolling.

 See Also

**Note:** For many purposes, you can use the Interaction Wizard to bind the KeyTyped event in order to receive notification whenever a user presses any key in the text area. This is often good enough to replace a true change notification.

## Properties

To make the JTextArea text read-only, use the [Editable property](#).

Use the [Columns property](#) to specify the number of text columns.

Use the Rows property to

## Coding the component

By default, JTextArea behavior accepts keyboard input and displays that input in its display pane. You can use the Interaction Wizard to change this default behavior.

To create a new JTextArea component in project source code, use the following syntax:

```
JTextArea1 = new com.sun.java.swing.JTextArea("text", cols, rows);  
getContentPane().add(JTextArea1);
```

Part	Type	Description
JTextArea1	JTextArea	The variable name used to reference the component in code.
text	String	The text to display in the text field.
cols	int	The width (in characters) that the text field can display and accept as input.
rows	int	The number of rows of text that the text area can display.

## **JTextArea Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Columns](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DisabledTextColor](#)  
[Document](#)  
[DoubleBuffered](#)  
[Editable](#)  
[Enabled](#)  
[FocusAccelerator](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Highlighter](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Keymap](#)  
[LineCount](#)  
[LineWrap](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Rows](#)  
[Selection](#)  
[TabSize](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)  
[WrapStyleWord](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

## Description

Extends JTextComponent.

A box in which your user can type text.

Use JTextField to:

- display text captured from the keyboard.
- edit a line of text.
- post an event based on text input from the keyboard.

Use the Interaction Wizard to trigger one or more actions when the JTextField generates events.

If the text box already contains text, the user can select the default text and delete or edit it.

The horizontal alignment of JTextField can be set to be left justified, centered, or right justified if the required size of the field text is smaller than the size allocated to it. This is determined by the `setHorizontalAlignment` and `getHorizontalAlignment` methods. The default is to be left justified.

## Properties

Use the [Editable property](#) to make the default text editable. You do not need to write code or use the Interaction Wizard to connect the JTextField component with any other component to display text.

The [Horizontal Alignment property](#) can be set to be left justified, centered, or right justified if the required size of the field text is smaller than the size allocated to it.

## Coding the component

To create a new TextField component in project source code, use the following syntax:

```
JTextField1 = new com.sun.java.swing.JTextField();  
JTextField1.setText("text");  
JTextField1.setColumns(cols);
```

Part	Type	Description
JTextField1	JTextField	The variable name used to reference the component in code.
text	String	The text to display in the text field.
cols	int	The width (in characters) that the text field can display and accept as input.

## **JTextField Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Caret](#)  
[CaretColor](#)  
[Class](#)  
[Columns](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DisabledTextColor](#)  
[Document](#)  
[DoubleBuffered](#)  
[Editable](#)  
[Enabled](#)  
[FocusAccelerator](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[Highlighter](#)  
[HorizontalAlignment](#)  
[Default Background](#)  
[Default Font](#)  
[Default Foreground](#)  
[Insets](#)  
[Keymap](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[ScrollOffset](#)  
[Selection](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Extends JEditorPane.

A text component that functions like a word processor. JTextPane allows you to have different text colors, multiple fonts and text styles in the paragraphs of text. JTextPane also supports image embedding. JTextPane can also contain other components, like a JButton.

### Properties

Use the [Highlighter property](#) to set the background highlight color.

The [CaretColor property](#) is used to set the text color.

Use the [StyledDocument property](#) to select a document model containing styles to use in the JTextPane.

### Coding the component

In project source code, use this syntax to create a new JTextPane:

```
JTextPanel = new com.sun.java.swing.JTextPane();
```

Part	Type	Description
JTextPanel1	JTextPane	The variable name used to identify the component in code.

## **JTextPane Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Caret](#)  
[CaretColor](#)  
[Class](#)  
[ContentType](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DisabledTextColor](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[Editable](#)  
[EditorKit](#)  
[FocusAccelerator](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Highlighter](#)  
[Default Background](#)  
[Insets](#)  
[Keymap](#)  
[LogicalStyle](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Selection](#)  
[StyledDocument](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

A button used to select one choice from several options.

Use a JToggleButton to:

- manage a selection-deselection user task.
- display a selection choice.
- post an event to another component.
- determine that only one choice in a set of choices is selected.

Typically, a JToggleButton will be a member of a group. In a group of toggle buttons, only one can be selected at a time. When the user selects one toggle button, the other toggle buttons in the group are automatically deselected.

Use the Interaction Wizard to trigger an action when a JToggleButton is clicked.

### Properties

To set the default state, checked or cleared, of the JToggleButton, use the [Selected](#).

### Coding the component

Use this syntax to create a toggle button:

```
JToggleButton1 = new com.sun.java.swing.JToggleButton("label");
```

Part	Type	Description
JToggleButton1	JToggleButton	The variable name used to reference the component in code.
label	String	The text to display next to the component.

## JToggleButton Properties

[Alignment/Position](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusPainted](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[HorizontalAlignment](#)

[HorizontalTextPosition](#)

[Icons](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Insets](#)

[Layout](#)

[ManagingFocus](#)

[Mnemonic](#)

[Model](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[RolloverEnabled](#)

[Selected](#)

[Text](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[VerticalAlignment](#)

[VerticalTextPosition](#)

[Visible](#)

[VisibleRect](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

JToolBar provides a component which is useful for displaying commonly used actions or controls. You can even add buttons, combo boxes or additional menus to a toolbar. It can be dragged out into a separate window by the user (unless the `floatable` property is set to `false`). In order for drag-out to work correctly, it is recommended that you add JToolBar instances to a container whose layout manager is a `BorderLayout`. Place the toolbar in one of the four sides of the container and do not add children to the remaining sides.

### Properties

Use the [Floatable property](#) to set whether the toolbar is permanently situated or can be moved within the viewing area.

### Coding the component

In project source code, use this syntax to create a new JToolBar:

```
JToolBar1 = new com.sun.java.swing.JToolBar();
```

Part	Type	Description
JToolBar1	JToolBar	The variable name used to identify the component in code.

## **JToolBar Properties**

[AlignmentX](#)

[AlignmentY](#)

[Autoscrolls](#)

[Background](#)

[Border](#)

[BorderPainted](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[DebugGraphicsOptions](#)

[DoubleBuffered](#)

[Enabled](#)

[FocusCycleRoot](#)

[FocusTraversable](#)

[Font](#)

[Foreground](#)

[Floatable](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Insets](#)

[Layout](#)

[ManagingFocus](#)

[Margin](#)

[Name](#)

[NextFocusableComponent](#)

[Opaque](#)

[OptimizedDrawingEnabled](#)

[PaintingTile](#)

[RequestFocusEnabled](#)

[ToolTipText](#)

[UIClassID](#)

[ValidateRoot](#)

[Visible](#)

[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

Creates spaces or separation between items in a JToolBar.

**Coding the component**

In project source code, use this syntax to create aJToolBarSeparator:

```
JToolBarSeparator1 = new com.symantec.itools.swing.JToolBarSeparator();
getContentPane().add(JToolBarSeparator1);
```

Part	Type	Description
JToolBarSeparator1	ToolBarSeparator	The variable name used to reference the component in code.

## JToolBarSeparator Properties

[Bounds](#)

[Class](#)

[Name](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

A control that displays a set of hierarchical data as an outline. A specific node can be identified either by a `TreePath` (an object that encapsulates a node and all of its ancestors) or by its display row, where each row in the display area displays one node. Use the [StringTreeModel](#) component to create a list of items for the tree.

An expanded node is one which displays its children. A collapsed node is one which hides them. A visible node is one which is currently viewable in the display area.

If you are interested in knowing when the selection changes, implement the `TreeSelectionListener` interface and add the instance using the method `addTreeSelectionListener`. `valueChanged` will be invoked when the selection changes; if the user clicks twice on the same node `valueChanged` will only be invoked once.

 See Also

**Note:** You need to place a `JTree` inside a `JScrollPane` component if you want the tree to be scrollable.

### Properties

Use the [RootVisible property](#) to control whether the root of the tree is visible.

### Coding the component

In project source code, use this syntax to create a new `JTree`:

```
JTree1 = new com.sun.java.swing.JTree();  
getContentPane().add(JTree1);
```

Part	Type	Description
------	------	-------------

JTree1	JTree	The variable name used to identify the component in code.
--------	-------	---

## JTree Properties

<a href="#">AlignmentX</a>	<a href="#">ManagingFocus</a>
<a href="#">AlignmentY</a>	<a href="#">MaxSelectionRow</a>
<a href="#">Autoscrolls</a>	<a href="#">MinSelectionRow</a>
<a href="#">Background</a>	<a href="#">Model</a>
<a href="#">Border</a>	<a href="#">Name</a>
<a href="#">Bounds</a>	<a href="#">NextFocusableComponent</a>
<a href="#">Class</a>	<a href="#">Opaque</a>
<a href="#">CellEditor</a>	<a href="#">OptimizedDrawingEnabled</a>
<a href="#">CellRenderer</a>	<a href="#">PaintingTile</a>
<a href="#">Cursor</a>	<a href="#">PreferredScrollableViewportSize</a>
<a href="#">DebugGraphicsOptions</a>	<a href="#">RequestFocusEnabled</a>
<a href="#">DoubleBuffered</a>	<a href="#">RootVisible</a>
<a href="#">Editable</a>	<a href="#">RowCount</a>
<a href="#">Editing</a>	<a href="#">RowHeight</a>
<a href="#">Enabled</a>	<a href="#">ScrollableTracksViewportHeight</a>
<a href="#">FixedRowHeight</a>	<a href="#">ScrollableTracksViewportWidth</a>
<a href="#">FocusCycleRoot</a>	<a href="#">SelectionCount</a>
<a href="#">FocusTraversable</a>	<a href="#">SelectionEmpty</a>
<a href="#">Font</a>	<a href="#">SelectionMode</a>
<a href="#">Foreground</a>	<a href="#">SelectionPath</a>
<a href="#">Default Background</a>	<a href="#">ShowsRootHandles</a>
<a href="#">Default Font</a>	<a href="#">ToolTipText</a>
<a href="#">Default Foreground</a>	<a href="#">UIClassID</a>
<a href="#">Insets</a>	<a href="#">ValidateRoot</a>
<a href="#">InvokesStopCellEditing</a>	<a href="#">Visible</a>
<a href="#">LargeModel</a>	<a href="#">VisibleRect</a>
<a href="#">Layout</a>	<a href="#">VisibleRowCount</a>
<a href="#">LeadSelectionRow</a>	

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



### Description

JViewport can be used to create a partial view of a component, particularly very large components. Usually, you use a viewport because the component is potentially too large to fit into a small place. An example of a viewport can be found in JScrollPane, which uses a viewport to show part of the component and uses scrollbars to move the view area to another part of the component.

### Properties

Use the [ViewSize property](#) to set the view coordinates that appear in the upper left hand corner of the viewport, and the size of the view.

### Coding the component

In project source code, use this syntax to create a new JViewport:

```
JViewport1 = new com.sun.java.swing.JViewport();  
getContentPane().add(JViewport1);
```

Part	Type	Description
JViewport1	JViewport	The variable name used to identify the component in code.

## JViewport Properties

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[BackingStoreEnabled](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[ExtentSize](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[InheritBackground](#)  
[InheritFont](#)  
[InheritForeground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[View](#)  
[ViewPosition](#)  
[ViewRect](#)  
[ViewSize](#)  
[Visible](#)  
[VisibleRect](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A simple top-level window that has no border, title, or menubar. It behaves modally, blocking input to other windows when visible.

When you add a JWindow component to a project, a new Form Designer window opens. You can add other components to the JWindow by dragging components and containers to the Form Designer as necessary.

Visual Cafe provides frames, dialogs and other window subclasses for you. You must use a JFrame component to create a Swing-based application. You must use the JApplet component to create a Swing-based applet program.

### Coding the component

To create a JWindow in project source code, use the following syntax:

```
public class JWindow1 extends com.sun.java.swing.Jwindow
(new JWindow1()).setVisible(true);
```

Part	Type	Description
JWindow1	JWindow	The variable name used to reference the component in code.

## **JWindow Properties**

[Background](#)

[Bounds](#)

[Class](#)

[Cursor](#)

[Enabled](#)

[Font](#)

[Foreground](#)

[Default Background](#)

[Default Font](#)

[Default Foreground](#)

[Layout](#)

[Name](#)

[Visible](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

A multi-line text label. Text is displayed on one or more lines, wrapping as needed to fit in the available horizontal space.

Use a WrappingLabel when you want to display a paragraph of static (not user-editable) text.

An application or applet can change the label text string using the [setText](#) method, but a user cannot edit it.

### Properties

To specify the displayed text, type the text in the [Text property](#).

To specify how the text is justified within the bounds of the wrapping label, use the [Alignment Style property](#).

### Coding the component

In project source code, use this syntax to create a new JWrappingLabel:

```
JWrappingLabel1 = new com.symantec.itools.swing.JWrappingLabel();  
getContentPane().add(JWrappingLabel1);
```

Part	Type	Description
JWrappingLabel1	JWrappingLabel	The variable name used to identify the component in code.

## **JWrappingLabel Properties**

[AlignmentX](#)  
[AlignmentY](#)  
[Autoscrolls](#)  
[Background](#)  
[Border](#)  
[Bounds](#)  
[Class](#)  
[Cursor](#)  
[DebugGraphicsOptions](#)  
[DoubleBuffered](#)  
[Enabled](#)  
[FocusCycleRoot](#)  
[FocusTraversable](#)  
[Font](#)  
[Foreground](#)  
[InheritBackground](#)  
[InheritFont](#)  
[InheritForeground](#)  
[Insets](#)  
[Layout](#)  
[ManagingFocus](#)  
[Name](#)  
[NextFocusableComponent](#)  
[Opaque](#)  
[OptimizedDrawingEnabled](#)  
[PaintingTile](#)  
[RequestFocusEnabled](#)  
[Text](#)  
[TextAlignment](#)  
[ToolTipText](#)  
[UIClassID](#)  
[ValidateRoot](#)  
[Visible](#)  
[VisibleRect](#)

## Model components

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Use a `FileTableModel` when you want to populate a table with data from a previously created file.

### Properties

To specify column names, use the [ColumnHeaders property](#).

 See Also

**Note:** You must place the table inside a [JScrollPane](#) to see the column headers.

To specify the data used in the model, use the [Items property](#).

### Coding the component

In project source code, use this syntax to create a new `FileTableModel`:

```
fileTableModel1 = new com.symantec.itools.swing.models.FileTableModel();
```

Part	Type	Description
<code>fileTableModel1</code>	<code>FileTableModel</code>	The variable name used to identify the component in code.

## FileTableModel Properties

[Class](#)

[ColumnHeaders](#)

[Items](#)

[Name](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Displays a list of data for a JComboBox component. This type of model uses a tab-delimited file.

### Properties

To specify the data displayed in a JComboBox component, use the [Items property](#).

### Coding the component

In project source code, use this syntax to create a new StringComboBoxModel:

```
stringComboBoxModel1 = new com.symantec.itools.swing.models.StringComboBoxModel();
```

Part	Type	Description
stringComboBoxModel1	StringComboBoxModel	The variable name used to identify the component in code.

## StringComboBoxModel Properties

Class

Items

Name

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Displays a list of data for a JList component.

### Properties

To specify the data displayed in a JList component, use the [Items property](#).

### Coding the component

In project source code, use this syntax to create a new StringListModel:

```
stringListModel1 = new com.syntec.itools.swing.models.StringListModel();
```

Part	Type	Description
stringListModel1	StringListModel	The variable name used to identify the component in code.

## StringListModel Properties

[Class](#)

[Items](#)

[Name](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Use a `StringTableModel` when you want to create an editable comma-delimited data file to use in a `JTable`.

### Properties

To specify column names, use the [ColumnHeaders property](#). Use a comma to separate the column names; for example:  
Name, Dept., Phone Number.

 See Also

**Note:** You must place the table inside a [JScrollPane](#) to see the column headers. Use the [Editable property](#) to allow your users to add data.

To specify the data used in the model, use the [Items property](#). Use a comma to separate the data into the correct column. Treat each line as one row, as shown below:

```
bob jones, sales, 555-1212
stella moon, payroll, 555-2121
```

### Coding the component

In project source code, use this syntax to create a new `StringTableModel`:

```
stringTableModel1 = new com.symantec.itools.swing.models.StringTableModel();
```

Part	Type	Description
stringTableModel1	StringTableModel	The variable name used to identify the component in code.

## StringTableModel Properties

[Class](#)

[ColumnHeaders](#)

[Editable](#)

[Items](#)

[Name](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

**Description**

Displays a list of data for a JTree component.

**Properties**

To specify the data displayed in a JTree component, use the [Items property](#). Items need to be formatted like the following example to create parents and children:

TreeItem1	Folder
Child1 of TreeItem1	Folder item
Child2 of TreeItem1	Folder (has item under it)
grandchild1	Folder item
TreeItem2	Folder
Child1 of TreeItem2	Folder item
Child2 of TreeItem2	Folder item

Use the [RootName property](#) to set the name of the uppermost (or root) folder in a tree.

**Coding the component**

In project source code, use this syntax to create a new StringTreeModel:

```
stringTreeModel1 = new com.symantec.itools.swing.models.StringTreeModel();
```

Part	Type	Description
stringTreeModel1	StringTreeModel	The variable name used to identify the component in code.

## StringTreeModel Properties

[Class](#)

[Items](#)

[Name](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

**Description**

Forces a text component to display only numbers.

**Properties**

Use the [\\_DefaultValue property](#) to set the initial value displayed in the component.  
Set the [\\_DefaultValueEnabled property](#) to determine whether a default value will be displayed.

**Coding the component**

In project source code, use this syntax to create a new UnsignedNumberDocument:

```
unsignedNumberDocument1 = new com.symantec.itools.swing.models.UnsignedNumberDocument();
```

Part	Type	Description
unsignedNumberDocument1	UnsignedNumberDocument	The variable name used to identify the component in code.

## UnsignedNumberDocument Properties

Class

DefaultValue

DefaultValueEnabled

Name

## Border components

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

Display component borders with a beveled edge.

**Properties**

To specify the colors displayed when the button is selected, use the [HighlightInnerColor property](#) and the [HighlightOuterColor property](#).

Use [ShadowInnerColor property](#) and [ShadowOuterColor property](#) to specify the colors displayed when the button is not selected.

**Coding the component**

In project source code, use this syntax to create a new BevelBorder:

```
bevelBorder1 = new com.symantec.itools.swing.borders.BevelBorder();
```

Part	Type	Description
bevelBorder1	BevelBorder	The variable name used to identify the component in code.

## BevelBorder Properties

BevelType

Class

HighLightInnerColor

HighLightOuterColor

Name

ShadowInnerColor

ShadowOuterColor

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Use this border to combine other border components such as EtchedBorder and BevelBorder to create an etched beveled border.

### Properties

To specify the inner border use the [InsideBorder property](#).

Use [OutsideBorder property](#) to specify the outer border.

### Coding the component

In project source code, use this syntax to create a new CompoundBorder:

```
compoundBorder1 = new com.symantec.itools.swing.borders.CompoundBorder();
```

Part	Type	Description
compoundBorder1	CompoundBorder	The variable name used to identify the component in code.

## CompoundBorder Properties

Class

InsideBorder

Name

OutsideBorder

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

#### Description

Does not display any border around a component.

#### Coding the component

In project source code, use this syntax to create a new EmptyBorder:

```
emptyBorder1 = new com.syntec.itools.swing.borders.EmptyBorder();
```

Part	Type	Description
emptyBorder1	EmptyBorder	The variable name used to identify the component in code.

## EmptyBorder Properties

[Class](#)

[Insets](#)

[Name](#)

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Displays component borders with an etched edge.

### Properties

Use the [HighlightColor property](#) and the [ShadowColor property](#) to set the highlight and shadow colors of the border edge.

To make the edge look raised or lowered use the [EtchType property](#).

### Coding the component

In project source code, use this syntax to create a new EtchedBorder:

```
etchedBorder1 = new com.syntec.itools.swing.borders.EtchedBorder();
```

Part	Type	Description
etchedBorder1	EtchedBorder	The variable name used to identify the component in code.

## EtchedBorder Properties

[Class](#)

[EtchType](#)

[HighlightColor](#)

[Name](#)

[ShadowColor](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

### Description

Displays a flat line border.

### Properties

To specify the border color, use the [LineColor property](#).

To specify whether the border corners are square or round, use the [RoundedCorners property](#).

To change the thickness of the border, use the [Thickness property](#).

### Coding the component

In project source code, use this syntax to create a new LineBorder:

```
lineBorder1 = new com.symantec.itools.swing.borders.LineBorder();
```

Part	Type	Description
lineBorder1	LineBorder	The variable name used to identify the component in code.

## LineBorder Properties

Class

LineColor

Name

RoundedCorners

Thickness

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}

 See Also

 See Also

 See Also

**Description**

Displays a border with a matte edge.

**Properties**

To specify the edge color, use the [Color property](#).

**Coding the component**

In project source code, use this syntax to create a new MatteBorder:

```
matteBorder1 = new com.symantec.itools.swing.borders.MatteBorder();
```

Part	Type	Description
matteBorder1	MatteBorder	The variable name used to identify the component in code.

## MatteBorder Properties

[Class](#)

[Color](#)

[Insets](#)

[Name](#)

[TileIcon](#)

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



**Description**

Displays the border edge with a soft bevel.

**Properties**

To specify the border bevel colors displayed, use the [HighLightInnerColor property](#) and the [HighLightOuterColor property](#). Use [ShadowInnerColor property](#) and [ShadowOuterColor property](#) to specify the shadow colors.

**Coding the component**

In project source code, use this syntax to create a new SoftBevelBorder:

```
softBevelBorder1 = new com.symantec.itools.swing.borders.SoftBevelBorder();
```

Part	Type	Description
softBevelBorder1	SoftBevelBorder	The variable name used to identify the component in code.

## SoftBevelBorder Properties

BevelType

Class

HighLightInnerColor

HighLightOuterColor

Name

ShadowInnerColor

ShadowOuterColor

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



### Description

Displays a border with text inside.

This border is used with other borders that handle formatting the border edge.

### Properties

To specify the border controlling the edge format, use the [Border property](#).

Use the [Title property](#) to enter the text you want displayed inside the border edges.

To set the font for the text, use the [TitleFont property](#).

Use the [TitleColor property](#) to select the text color.

To justify the text, use the [TitleJustification property](#).

To specify the text location inside the border, use the [TitlePosition](#).

### Coding the component

In project source code, use this syntax to create a new TitledBorder:

```
titledBorder1 = new com.symantec.itools.swing.borders.TitledBorder();
```

Part	Type	Description
titledBorder1	TitledBorder	The variable name used to identify the component in code.

## **TitledBorder Properties**

[Border](#)

[Class](#)

[Name](#)

[Title](#)

[TitleColor](#)

[TitleFont](#)

[TitleJustification](#)

[TitlePosition](#)

## Action components

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Create actions used by JactionButton or JActionMenuItem.

### Properties

Use the [ActionName property](#) to set the name of the action.

Use the [SmallIcon property](#) to select an image to display.

### Coding the component

In project source code, use this syntax to create a new Action:

```
public class Action1 extends com.symantec.itools.swing.actions.Action
```

Part	Type	Description
------	------	-------------

Action 1	Action	The variable name used to identify the action in code.
----------	--------	--

## Action Properties

ActionName

Class

Enabled

Name

SmallIcon

## Imagelcon components

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

 See Also

 See Also

 See Also

### Description

Display images as icons inside other components. Use an ImageIcon when you want to display graphics on a button.

### Properties

To specify the displayed image, type the full path and file name or use the browse button in the [ImageLocation property](#).

### Coding the component

In project source code, use this syntax to create a new ImageIcon:

```
imageIcon1 = new com.symantec.itools.swing.icons.ImageIcon();
```

Part	Type	Description
ImageIcon 1	ImageIcon	The variable name used to identify the ImageIcon in code.

## ImageIcon Properties

Class

ImageLocation

ImageObserver

Name

## Change history

6/11/98 - created swing properties file

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also


 See Also

 See Also

 See Also


 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

 See Also

## Swing properties

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Creates a shortcut key for the component.

#### Runtime modification

At runtime, use the `setAccelerator` method to modify this property.

#### Settings

Valid values	Description
SHIFT_MASK	Shift key is used.
CTRL_MASK	Ctrl key is used.
ALT_MASK	Alt key is used.
META_MASK	Right mouse button is used.

After choosing one of the three values, type a character in the text field above the check boxes. The resulting choice might look like CTRL B.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the action that is invoked when the button is pressed.

**Runtime modification**

At runtime, use the `setAction` method to modify this property.

**Settings**

The default is Null.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the position of text and justification of text or icons in a component. Components and containers that support the Text property and/or the Icons properties support this property.

---

## HorizontalTextPosition Property

Positioning property. Controls the location of text drawn in a component. Used when an Icon is present to ensure correct text placement in relation to the icon's location.

### Runtime modification

At runtime, use the `setHorizontalTextPosition` method to modify this property.

### Settings

Valid values	Description
CENTER	The text should appear in the center of the Icon.
LEFT	The text should be placed to the left of the Icon.
RIGHT	(Default) The text should be placed to the right of the Icon.

## VerticalTextPosition Property

Positioning property. Controls the location of text drawn in a component. Used when an Icon is present to ensure correct text placement in relation to the icon's location.

### Runtime modification

At runtime, use the `setVerticalTextPosition` method to modify this property.

### Settings

Valid values	Description
CENTER	(Default) The text should appear in the center of the Icon.
TOP	The text should be placed above the Icon.
BOTTOM	The text should be placed below the Icon.

## HorizontalAlignment Property

Specifies the justification of text drawn in a component.

### Runtime modification

At runtime, use the `setHorizontalAlignment` method to modify this property.

### Settings

Valid values	Description
CENTER	(Default for JButton) The text should be centered.
LEFT	(Default for JLabel) The text should be placed towards the left.
RIGHT	The text should be placed towards the right.

## VerticalAlignment Property

Specifies the justification of text drawn in a component.

### Runtime modification

At runtime, use the `setVerticalAlignment` method to modify this property.

## Settings

Valid values	Description
CENTER	(Default) The text should be in the center.
TOP	The text should be placed at the top.
BOTTOM	The text should be placed at the bottom.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The preferred horizontal alignment of the component.

#### **Runtime modification**

At runtime, use the `setAlignmentX` method to modify this property.

#### **Settings**

The default is 0.5.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The preferred vertical alignment of the component.

#### **Runtime modification**

At runtime, use the `setAlignmentY` method to modify this property.

#### **Settings**

The default is 0.5.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the first index argument from the most recent `addSelectionInterval` or `setSelectionInterval` call.

#### **Runtime modification**

At runtime, use the `getAnchorSelectionIndex` method to modify this property.

#### **Settings**

This property is read only.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets data entry to behave as it does on an ATM machine when you enter an amount of cash to withdraw. The cursor remains fixed at the right end of the value. The decimal point is also fixed and always appears.

#### Runtime modification

At runtime, use `setATMMode` method to modify this property.

#### Settings

Valid values	Description
false	(Default) ATMMode is not active.
true	ATMMode is active.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Uses column information obtained from the model the component is using.

#### Runtime modification

At runtime, use the `setAutoCreateColumnsFromModel` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Uses column information from model.
false	Ignores column information from model.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the automatic resizing capability of the columns in the component.

#### Runtime modification

At runtime, use the `setAutoResizeMode` method to modify this property.

#### Settings

Valid values	Description
AUTO_RESIZE_ALL_COLUMNS	(Default) Automatically resizes all columns.
AUTO_RESIZE_LAST_COLUMN	Automatically resizes only the last column (farthest right column).
AUTO_RESIZE_OFF	Does not automatically resize any columns.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if automatic scrolling in a list, table, or tree will occur when dragging the mouse over the component.

#### Runtime modification

At runtime, use the `setAutoscrolls` method to modify this property

#### Settings

Valid values	Description
true	(Default) Automatic scrolling is enabled.
false	Automatic scrolling is disabled.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if Jviewport will maintain an offscreen image of its contents.

#### Runtime modification

At runtime, use the `setBackingStoreEnabled` method to modify this property.

#### Settings

Valid values	Description
true	Offscreen image is maintained.
false	(Default) No offscreen image is maintained.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Adds selected border component information to the component. You may select one of eight border types. Once selected you need to set the properties of the border instance to change the look of the component's borders.

### Runtime modification

At runtime, use the `setBorder` method to modify this property.

### Settings

The default setting is (null).

Valid values	Description
BevelBorder	A plain two line bevel border that can be lowered or raised
CompoundBorder	A border made up of two other border types that nest inside each other.
EmptyBorder	An invisible border.
EtchedBorder	A border that looks like an etched line.
LineBorder	Uses a single line border.
MatteBorder	A border that allows a tiled icon or a solid color
SoftBevelBorder	A softer edged version of BevelBorder
TitledBorder	A border that allows you to display a title string with one of the other border types in several positions and justifications.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Adds selected border component information to a JListChooser component.

## Settings

---

### Border Property

Adds selected border component information to the component. You may select one of eight border types. Once selected you need to set the properties of the border instance to change the look of the component's borders.

#### Runtime modification

At runtime, use the `setBorder` method to modify this property.

#### Settings

The default setting is (Default).

Valid values	Description
BevelBorder	A plain two line bevel border that can be lowered or raised
CompoundBorder	A border made up of two other border types that nest inside each other.
EmptyBorder	An invisible border.
EtchedBorder	A border that looks like an etched line.
LineBorder	Uses a single line border.
MatteBorder	A border that allows a tiled icon or a solid color
SoftBevelBorder	A softer edged version of BevelBorder
TitledBorder	A border that allows you to display a title string with one of the other border types in several positions and justifications.

### Available List Border Property

Adds chosen border component information to the available list. You may select one of eight border types. Once selected you need to set the properties of the border instance to change the look of the component's borders.

#### Runtime modification

At runtime, use the `setAvailableBorder` method to modify this property.

#### Settings

The default setting is (Default).

### Selected List Border Property

Adds chosen border component information to the selected list. You may select one of eight border types. Once selected you need to set the properties of the border instance to change the look of the component's borders.

#### Runtime modification

At runtime, use the `setSelectedBorder` method to modify this property.

#### Settings

The default setting is (Default).

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the border will be displayed around a component.

#### Runtime modification

At runtime, use the `setBorderPainted` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Display a border.
false	No border is displayed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Creates button labels for the buttons that control list item movement between the Available List and the Selected List items in a JListChooser component.

## Settings

---

### Clear Label Property

Removes all items from the Selected List and places the items in the Available List.

#### Runtime modification

At runtime, use the `setClearLabel` method to modify this property.

#### Settings

The default setting is <<Clear.

### Add Label Property

Adds the chosen item(s) to the Selected List.

#### Runtime modification

At runtime, use the `setAddLabel` method to modify this property.

#### Settings

The default setting is Add>.

### Remove Label Property

Removes chosen items from the Selected List and places the items in the Available List.

#### Runtime modification

At runtime, use the `setremoveLabel` method to modify this property.

#### Settings

The default setting is <Remove.

### Add All Label Property

Adds all item(s) from the Available List to the Selected List.

#### Runtime modification

At runtime, use the `setAddAllLabel` method to modify this property.

#### Settings

The default setting is AddAll>>.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Creates the "|" symbol that indicates the current cursor position .

#### **Runtime modification**

At runtime, use the `setCaret` method to modify this property.

#### **Settings**

The default is default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the current color used to render the caret.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setCaretColor` method to modify this property.

### Settings

Valid values	Description
black	(Default) Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the cell editor. A null value implies the component cannot be edited.

**Runtime modification**

At runtime, use the `setCellEditor` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Paints the visible cells in the list of a component.

**Runtime modification**

At runtime, use the `setCellRenderer` method to modify this property.

**Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines cell selection behavior in the component.

#### Runtime modification

At runtime, use the `setCellSelectionEnabled` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Selecting a single cell also selects entire row and/or column.
true	Only the chosen cell is selected.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Indicates whether this frame can be closed by a user action.

#### Runtime modification

At runtime, use the `setClosable` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Frame can not be closed.
true	Frame can be closed by user action.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Indicates that the frame has been closed.

#### Runtime modification

At runtime, use the `setClosed` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Frame is Open.
true	Frame is closed.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies the number of columns in a text component. Each column holds a single character. If set to zero, the preferred width will be whatever naturally results from the component implementation.

#### Runtime modification

At runtime, use the `setColumns` method to modify this property.

#### Settings

Valid values	Description
0	(Default) Don't use when determining minimum component size.
1-32767	The desired number of columns to display.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the number of columns in the column model, note this may be different to the number of columns in the table model.

**Settings**

The default is 0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the object governing the way columns appear in the view.

**Runtime modification**

At runtime, use the `setColumnModel` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether cell selection also selects a column in the component.

#### Runtime modification

At runtime, use the `setColumnSelectionAllowed` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Selecting a cell in a column does not select the entire column.
true	Selecting a cell in a column selects the entire column.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the type of content that this editor handles.

#### Runtime modification

At runtime, use the `setContentTypes` method to modify this property.

#### Settings

Valid values	Description
text/plain	(Default) Regular text, no attributes.
text/rtf	Rich Text Format text.
text/html	HTML defined text.
application/rtf	Rich Text Format application

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets whether or not the child components are continuously redisplayed and layed out during user intervention.

#### Runtime modification

At runtime, use the `setContinuousLayout` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Layout is not continuous for child components.
true	Layout is continuous for child components.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Enables or disables diagnostic information about every graphics operation performed within the component or one of its children.

#### Runtime modification

At runtime, use the `setDebugGraphicsOptions` method to modify this property.

#### Settings

Valid values	Description
LOG_OPTION	Causes a text message to be printed.
FLASH_OPTION	Causes the drawing to flash several times.
BUFFERED_OPTION	Creates an ExternalWindow that displays the operations performed on the View's offscreen buffer.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the character used to represent a decimal point.

#### **Runtime modification**

At runtime, use `setDecimalPoint` method to modify this property.

#### **Settings**

The default is a period. Use a comma for most international formats.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the operation which will happen by default when the user initiates a "close" on this window.

#### Runtime modification

At runtime, use the `setDefaultCloseOperation` method to modify this property.

#### Settings

Valid values	Description
DO_NOTHING_ON_CLOSE	- Do not do anything - require the program to handle the operation.
HIDE_ON_CLOSE	(Default) Automatically hide the window after invoking any registered <code>InternalFrameListener</code> objects.
DISPOSE_ON_CLOSE	Automatically hide and dispose the window after invoking any registered <code>InternalFrameListener</code> objects.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines where a component gets its background color setting from. When true the component uses the background color from the swing look and feel. When false the component uses the color specified by the Background property.

All swing components and containers support this property.

#### Runtime modification

At run time, use the `setBackground` method to modify this property. Passing a null value will make the component get its background from its parent container.

#### Settings

Valid values	Description
true	(Default) The component uses the background color from the swing look and feel.
false	The component uses its own <a href="#">Background property</a> .

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the current default button for this component.

#### **Runtime modification**

At runtime, use the `setDefaultButton` method to modify this property.

#### **Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines where a component gets its font settings from . When true the component uses the font from the swing look and feel. When false the component uses the font specified by the Font property.

All swing components and [containers](#) support this property.

#### Runtime modification

At run time, use the `setFont` method to modify this property. Passing a null value will make the component get its font from its parent component.

#### Settings

Valid values	Description
true	(Default) The component uses the font set in the swing look and feel.
false	The component uses its own Font property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines where a component gets its foreground color setting from. When true the component uses the foreground color from the swing look and feel. When false the component uses the color specified by the Foreground property.

All swing components and containers support this property.

#### Runtime modification

At run time, use the `setForeground` method to modify this property. Passing a null value will make the component get its foreground from the swing look and feel.

#### Settings

Valid values	Description
true	(Default) The component uses the foreground color in the Swing look and feel.
false	The component uses its own <a href="#">Foreground property</a> .

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the suggested delay before the menu's PopupMenu is popped up or down.

#### Runtime modification

At runtime, use `setDelay` method to modify this property.

#### Settings

Valid values	Description
200	(Default) The current delay setting in milliseconds.
1-32767	Use a positive integer to set a delay in milliseconds.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of digits displayed after the decimal point.

**Runtime modification**

At runtime, use `setDigitsAfterDecimal` method to modify this property.

**Settings**

The default is 2. Any integer value can be used.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```

Determines whether an icon, an action name or both an icon and an action name will be displayed by a JButton.

#### Runtime modification

At runtime, use `setDisplay` method to modify this property.

#### Settings

Valid values	Description
DISPLAY_ACTION_ICON_AND_NAME	(Default) Displays both the icon for the action and the name of the action.
DISPLAY_ACTION_ICON	Displays the icon for the action.
DISPLAY_ACTION_NAME	Displays the name of the action.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies a keycode that indicates a mnemonic key. This property is used when the label is part of a larger component. If the `labelFor` property of the label is not set to `Default`, the label will call the `requestFocus` method of the component specified by the `labelFor` property when the mnemonic is activated.

#### **Runtime modification**

At runtime, use `setDisplayMnemonic` method to modify this property.

#### **Settings**

The default is 0. Use a valid keycode.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the current color used to render the disabled text.

### Runtime modification

At runtime, use the `setDisabledTextColor` method to modify this property.

### Settings

Valid values	Description
black	Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

The default color is 153,153,153 in JPasswordField.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the location of the divider from the look and feel implementation.

**Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the size of the divider.

#### Runtime modification

At runtime, use the `setDividerSize` method to modify this property.

#### Settings

Valid values	Description
8	(Default) The current diver size.
1-32767	The valid range of divider sizes.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Associates the editor with a text document.

**Runtime modification**

At runtime, use the `setDocument` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets whether the drawing from this component will be done in an offscreen painting buffer.

#### Runtime modification

At runtime, use the `setDoubleBuffered` method to modify this property

#### Settings

Valid values	Description
false	(Default) Does not use a buffer to paint.
true	Uses a buffer to paint.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the echo character for this JPasswordField.

#### **Runtime modification**

At runtime, use the `setEchoChar` method to modify this property.

#### **Settings**

The default is `'*'`. Any character can be used of type `char`.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if the tree is being edited.

#### Settings

Valid values	Description
false	(Default) The tree is not being edited.
true	The tree is being edited.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the index of the editing column.

#### Runtime modification

At runtime, use the `setEditingColumn` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) No columns are edited.
1-32767	The desired column to edit.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the index of the editing row.

#### Runtime modification

At runtime, use the `setEditingRow` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) No rows are edited.
1-32767	The desired row to edit.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the editor that JComboBox uses to edit the current value.

**Runtime modification**

At runtime, use the `setEditor` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Installs the kit for handling content. Old kits are deinstalled before the new kit is installed.

**Runtime modification**

At runtime, use the `setEditorKit` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the size of the range "covered" by the knob in a JSlider.

**Runtime modification**

At runtime, use the `setExtent` method to modify this property.

**Settings**

The default is 0. Use any positive integer.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the size of a dimension corresponding to the viewport.

**Runtime modification**

At runtime, use the `setExtentSize` method to modify this property.

**Settings**

The default is 0,0. Use integers in an x, y pattern.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the index of the first visible cell in the upper left corner.

#### Runtime modification

At runtime, use the `getFirstVisibleIndex` method to modify this property.

#### Settings

Valid values	Description
-1	(Default)The list is empty or the first cell is not visible
1-32767	The index number of the first cell of the list.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines a fixed cell height in pixels when greater than zero.

#### Runtime modification

At runtime, use the `setFixedCellHeight` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) Cell heights will be calculated.
1-32767	Cell heights are fixed at the specified height.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines a fixed cell width in pixels when greater than zero.

#### Runtime modification

At runtime, use the `setFixedCellWidth` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) Cell widths will be calculated.
1-32767	Cell widths are fixed at the specified width.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the height of each display row is a fixed size.

**Settings**

The default is true. This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether the toolbar can be made to float.

#### Runtime modification

At runtime, use the `setFloatable` method to modify this property.

#### Settings

Valid values	Description
false	The toolbar can not be dragged by the user.
true	(Default) The toolbar can be dragged by the user.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the key accelerator that will cause the receiving text component to get the focus.

**Runtime modification**

At runtime, use the `setFocusAccelerator` method to modify this property.

**Settings**

The default is Null.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if your component is the root of a component tree with its own focus cycle.

#### Runtime modification

At runtime, use the `isFocusCycleRoot` method to modify this property.

#### Settings

Valid values	Description
true	Has its own focus cycle.
false	(Default) Does not have its own focus cycle.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether focus should be painted.

#### Runtime modification

At runtime, use the `setFocusPainted` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Focus is painted.
false	Focus is not painted.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns whether the component can accept focus. The component may return false when it is disabled.

#### Runtime modification

At runtime, use the `isFocusTraversable` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Can accept focus.
false	Cannot accept focus.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the color displayed for a grid.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setGridColor` method to modify this property.

### Settings

Valid values	Description
black	Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	(Default) Makes the component text white.
yellow	Makes the component text yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the highlighter to be used for background highlights.

**Runtime modification**

At runtime, use the `setHighlighter` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the justification of text drawn in a component.

#### Runtime modification

At runtime, use the `setHorizontalAlignment` method to modify this property.

#### Settings

Valid values	Description
CENTER	(Default for JButton) The text should be centered.
LEFT	(Default for JLabel) The text should be placed towards the left.
RIGHT	The text should be placed towards the right.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines when the horizontal scrollbar appears in the scrollpane.

#### Runtime modification

At runtime, use the `setHorizontalScrollBarPolicy` method to modify this property.

#### Settings

Valid values	Description
HORIZONTAL_SCROLLBAR_AS_NEEDED	(Default) The horizontal scrollbar appears when the text extends beyond the scrollpane.
HORIZONTAL_SCROLLBAR_NEVER	The horizontal scrollbar never appears.
HORIZONTAL_SCROLLBAR_ALWAYS	The horizontal scrollbar always appears.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Positioning property. Controls the location of text drawn in a component. Used when an Icon is present to ensure correct text placement in relation to the icon's location.

#### Runtime modification

At runtime, use the `setHorizontalTextPosition` method to modify this property.

#### Settings

Valid values	Description
CENTER	The text should appear in the middle of the Icon.
LEFT	The text should be placed to the left of the Icon.
RIGHT	(Default) The text should be placed to the right of the Icon.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Iconizes and deiconizes the frame.

#### Runtime modification

At runtime, use the `setIcon` method to modify this property.

#### Settings

Valid values	Description
true	Iconify the frame.
false	(Default) Deiconify the frame.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Names the instances of ImageIcon components used for various actions a component has.

### Settings

---

## Icon Property

Sets the name of the instance of ImageIcon you want to use in a component.

### Runtime modification

At runtime, use the `setIcon` method to modify this property.

### Settings

The default is None.

## DisabledIcon Property

Sets the name of the instance of ImageIcon you want to use when the component is disabled.

### Runtime modification

At runtime, use the `setDisabledIcon` method to modify this property.

### Settings

The default is None.

## DisabledSelectedIcon Property

Sets the name of the instance of ImageIcon you want to use when the component is disabled, but selected.

### Runtime modification

At runtime, use the `setDisabledSelectedIcon` method to modify this property.

### Settings

The default is None.

## PressedIcon Property

Sets the name of the instance of ImageIcon you want to use when in a component is pressed.

### Runtime modification

At runtime, use the `setPressedIcon` method to modify this property.

### Settings

The default is None.

## RolloverIcon Property

Sets the name of the instance of ImageIcon you want to use when the mouse rolls over the component.

### Runtime modification

At runtime, use the `setRolloverIcon` method to modify this property.

#### **Settings**

The default is None.

### **RolloverSelectedIcon Property**

Sets the name of the instance of `ImageIcon` you want to use when the mouse rolls over the component and the component is selected.

#### **Runtime modification**

At runtime, use the `SetRolloverSelectedIcon` method to modify this property.

#### **Settings**

The default is None.

### **SelectedIcon Property**

Sets the name of the instance of `ImageIcon` you want to use when the component is selected.

#### **Runtime modification**

At runtime, use the `setSelectedIcon` method to modify this property.

#### **Settings**

The default is None.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Controls the settings for iconizing an internal frame.

### Settings

---

## Icon property

Iconizes and deiconizes the frame.

### Runtime modification

At runtime, use the `setIcon` method to modify this property.

### Settings

Valid values	Description
true	Iconify the frame.
false	(Default) Deiconify the frame.

## DesktopIcon property

Sets the icon shown when this frame is minimized.

### Runtime modification

At runtime, use the `setDesktopIcon` method to modify this property.

### Settings

The default is Default.

## FrameIcon property

Sets an image to be displayed in the titlebar of the frame (usually in the top-left corner).

### Runtime modification

At runtime, use the `setFrameIcon` method to modify this property.

### Settings

The default is Default. Use the [ImageIcon](#) component to set an image.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the amount of space between the icon and text properties when both properties are set.

#### Runtime modification

At runtime, use the `setIconTextGap` method to modify this property.

#### Settings

Valid values	Description
4	(Default)
1-32767	Text gap is fixed at the specified width.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether this frame can be iconified.

#### Runtime modification

At runtime, use the `setIconifiable` method to modify this property.

#### Settings

Valid values	Description
true	The frame can be iconified.
false	(Default) The frame can not be iconified.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the option pane's initial selection value object (component).

**Runtime modification**

At runtime, use the `setInitialSelectionValue` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the initial component that is to be enabled when the pane is initially displayed.

**Runtime modification**

At runtime, use the `setInitialValue` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the option pane's input value object (component).

**Runtime modification**

At runtime, use the `setInputValue` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the space between a border and the component it contains.

**Settings**

The default is 0,0,0,0.

This is a read only property except for some of the Border components where any positive integer value may be used.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets whether the slider values are reversed from their normal order. The reverse order is highest value on the left and lowest value on the right side of the slider.

#### Runtime modification

At runtime, use the `setInverted` method to modify this property.

#### Settings

Valid values	Description
true	Values are reversed from normal order.
false	(Default) Values are not reversed.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the invoker of this popupmenu -- the component in which the popupmenu menu is to be displayed.

**Runtime modification**

At runtime, use the `setInvoker` method to modify this property.

**Settings**

The default is Default. Select any available component.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines what happens when editing is interrupted.

#### Runtime modification

At runtime, use the `setInvokesStopCellEditing` method to modify this property.

#### Settings

Valid values	Description
true	stopCellEditing is invoked when editing is interrupted, and data is saved.
false	(Default) cancelCellEditing is invoked, and changes are lost.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the number of items in the list or in a menu.

### **Settings**

The default is 0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Selects a JMenuBar when multiple menu bars are present.

#### **Runtime modification**

At runtime, use the `setJMenuBar` method to modify this property.

#### **Settings**

The default is None.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the object that translates a keyboard character into a list selection.

**Runtime modification**

At runtime, use the `setKeySelectionManager` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the keymap to use for binding events to actions.

#### **Runtime modification**

At runtime, use the `setKeymap` method to modify this property.

#### **Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Assigns a label component to another component.

**Runtime modification**

At runtime, use the `setLabelFor` method to modify this property.

**Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies whether the UI should use a large model with this component.

#### Runtime modification

At runtime, use the `setLargeModel` method to modify this property.

#### Settings

Valid values	Description
true	Use a large model.
false	(Default) Do not use a large model.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the last location the divider was at. Specify the last divider location in pixels, from the left (or upper) edge of the pane to the left (or upper) edge of the divider.

#### Runtime modification

At runtime, use the `setLastDividerLocation` method to modify this property.

#### Settings

Valid values	Description
0	(Default) The default location of the divider.
1-32767	The last location of the divider.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the index of the cell in the lower right corner of the component.

#### Settings

Valid values	Description
-1	(Default)The list is empty or the last cell is not visible
1-32767	The index number of the last cell of the list.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Specifies what desktop layer is used.

### **Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the second index argument from the most recent `addSelectionInterval` or `setSelectionInterval` call. This is a convenience method that just delegates to the `selectionModel`.

#### Runtime modification

At runtime, use the `getLeadSelectionIndex` method to modify this property.

#### Settings

Valid values	Description
-1	(Default)The list is empty.
1-32767	The number of the index that most recently ended a interval selection.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the row index of the last node added to the selection.

### **Settings**

The default is -1.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets where currency symbol appears in relation to a value.

#### Runtime modification

At runtime, use `setCurrencyLeading` method to modify this property.

#### Settings

Valid values	Description
false	Currency symbol displays after value.
true	(Default) Currency symbol displays before value.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Set whether using light weight popups is disabled.

#### Runtime modification

At runtime, use the `setLightWeightPopupEnabled` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Lightweight (all-Java) popups are in use.
false	Heavyweight (native peer) popups are being used.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the number of lines contained in the area.

### **Settings**

The default is 1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the line-wrapping policy of the text area.

#### Runtime modification

At runtime, use the `setLineWrap` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Lines will wrap.
false	Lines will not wrap.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the logical style to assign to the paragraph. This method is thread safe, although most Swing methods are not.

**Runtime modification**

At runtime, use the `setLogicalStyle` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of pixels between major tick marks.

**Runtime modification**

At runtime, use the `setMajorTickSpacing` method to modify this property.

**Settings**

The default is 0. Use any integer value to set the spacing.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if your component manages focus.

#### Settings

Valid values	Description
true	Manages focus of the component and all its children.
false	(Default) Does not manage focus.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets margin space between the text component's border and its text.

**Runtime modification**

At runtime, use the `setMargin` method to modify this property.

**Settings**

The default is 0,0,0,0. Use integer values to change the margin.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Controls the user input to the component.

The JMaskedTextField uses a set of predefined characters that define the mask. When defining a mask which uses a reserved character as a literal character, prefix the character with the Escape character ('\'), including spaces. For instance, to specify the mask for an 800 telephone number as: 1 8\0\0-AAA-AAAA. This example shows that the "(" ")" and "-" characters will be displayed literally and not as a special mask character.

The following are valid characters that can be used for a mask:

To restrict the digit position to:	Use:
Escape (characters are displayed as literals)	\
Digit	0
Digit or Space	9
Digit, Sign (+ or -), or Space	#
Letter (A through Z) entry required	L
Letter (A through Z) entry optional	?
Letter or Digit - entry required	A
Letter or Digit - entry optional	a
Any character or space - entry required	&
Any character or space - entry optional	C
Decimal placeholder and Thousands separator	.,
Date and Time separators	;-/
Forces all characters that follow to convert to lowercase	<
Forces all characters that follow to convert to uppercase	>

#### Runtime modification

At run time, the JMaskedTextField component can use the `setMask` method to modify this property.

#### Settings

Valid values	Description
empty	(Default) No mask is used.
valid character	One of the characters that is valid for a mask or a literal character preceded by an escape character.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether this frame can be maximized.

#### Runtime modification

At runtime, use the `setMaximizable` method to modify this property.

#### Settings

Valid values	Description
true	Frame can be maximized.
false	(Default) Frame can not be maximized.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the frame is maximized.

#### Runtime modification

At run time, use the `setMaximum` method to modify this property.

#### Settings

Valid values	Description
true	Maximizes the frame.
false	(Default) Frame is set to actual size.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the maximum value of the progress bar, scroll bar, or slider when it is in its bottom or rightmost position.

#### Runtime modification

At run time, use the `setMaximum` method to modify this property.

#### Settings

Valid values	Description
100	(Default) The maximum value is 100.
integer	The maximum scrollbar value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the maximum location of the divider from the look and feel implementation.

**Settings**

The default is an integer representing the pixel count.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines the maximum number of rows contained in a component. If there are more items in the list than the number of specified rows, a scroll bar is automatically added to the component.

#### Runtime modification

At runtime, use the `setMaximumRowCount` method to modify this property.

#### Settings

Valid values	Description
8	(Default)
1-32767	The desired number of rows to display.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the maximum number of characters to place on a line.

**Settings**

The default is 2147483647.

This is a read only property

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the largest selected cell index.

#### Runtime modification

At runtime, use the `getMaxSelectionIndex` method to modify this property.

#### Settings

Valid values	Description
-1	(Default) The list is empty
1-32767	The largest index number.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Gets the last selected row.

### **Settings**

The default is -1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the number of components on the menu.

**Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the number of items in the menu bar.

### **Settings**

The default is 0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the option pane's message-object.

**Runtime modification**

At runtime, use the `setMessage` method to modify this property.

**Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the option pane's message type.

#### Runtime modification

At runtime, use the `setMessageType` method to modify this property.

#### Settings

Valid values	Description
PLAIN_MESSAGE	(Default) No icon is used in the message, can be any message.
ERROR_MESSAGE	Used for error messages
INFORMATION_MESSAGE	Used for informational messages
WARNING_MESSAGE	Used for warning messges
QUESTION_MESSAGE	Used for question messages

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the maximum value of the progress bar, scroll bar, or slider when it is in its bottom or rightmost position.

#### Runtime modification

At run time, use the `setMaximum` method to modify this property.

#### Settings

Valid values	Description
100	(Default) The maximum value is 100.
integer	The maximum scrollbar value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the maximum location of the divider from the look and feel implementation.

**Settings**

The default is an integer representing the pixel count.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of pixels between minor tick marks.

#### **Runtime modification**

At runtime, use the `setMinorTickSpacing` method to modify this property.

#### **Settings**

The default is 0. Use any integer value to set the spacing.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the smallest selected cell index.

### **Settings**

The default is -1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Gets the first selected row.

### **Settings**

The default is -1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Character and other possible modifier keys that are associated with a button, menu, or menu item.

#### **Runtime modification**

At runtime, use the `setMnemonic` method to modify this property.

Convert string characters to integers by casting, for example:

```
jButton1.setMnemonic((int) 'K');
```

#### **Settings**

Any alphabetic or numeric character can be entered.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Controls the dialog's modality.

#### Runtime modification

At runtime, use the `setModal` method to modify this property.

#### Settings

Value	Description
true	The dialog box blocks input to other windows. While it is visible dialog box retrieves all user input.
false	(Default) The dialog box is modeless. Other windows can be activated and used while the dialog box is open.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Selects the model the component will use. There are five models provided in Visual Cafe. You can also create your own models.

#### Runtime modification

At runtime, use the `setModel` method to modify this property.

#### Settings

The default is Default.

Valid values	Description
FileTableModel	Used with table components to provide data.
StringComboBoxModel	Used with JComboBox to provide list items.
StringListModel	Used with JList and JListChooser to provide list items.
StringTableModel	Used with table components to provide data.
StringTreeModel	Used with JTree to provide tree items.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Selects the model the component will use. You can also create your own models by extending `DefaultStringModel` or `StringListModel`.

#### Settings

---

### Available List property

Selects an instance of `StringListModel` that the Available List will use.

#### Runtime modification

At runtime, use the `setAvailableModel` method to modify this property.

#### Settings

The default is Default.

Valid values	Description
<code>StringListModel</code>	Used with <code>JList</code> and <code>JListChooser</code> to provide list items.

### Selected List property

Selects an instance of `StringListModel` that the Selected List will use.

#### Runtime modification

At runtime, use the `setSelectedModel` method to modify this property.

#### Settings

The default is Default.

Valid values	Description
<code>StringListModel</code>	Used with <code>JList</code> and <code>JListChooser</code> to provide list items.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the next component to get the focus after this one.

**Runtime modification**

At runtime, use the `setNextFocusableComponent` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the number of major grid increments on the Y axis.

**Runtime modification**

At runtime, use the `setNumYTicks` method to modify this property.

**Settings**

The default is 5.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the JSplitPane provides a UI widget on the divider to quickly expand/collapse the divider.

#### Runtime modification

At runtime, use the `setOneTouchExpandable` method to modify this property.

#### Settings

Valid values	Description
false	(Default) No widget is provided to expand/collapse the divider.
true	A widget is provided to expand/collapse the divider.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if the component is transparent or opaque.

#### Runtime modification

At runtime, use the `setOpaque` method to modify this property.

#### Settings

Valid values	Description
false	The component is transparent.
true	(Default) The component is not transparent.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



determines if the component's children do not overlap. If the property returns true, the component tiles its children.

#### Runtime modification

At runtime, use the `isOptimizedDrawingEnabled` method to modify this property.

#### Settings

Valid values	Description
false	The component's children overlap.
true	(Default) The component's children do not overlap.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Defines the set of option buttons that appear at the bottom of the dialog box:.

#### Runtime modification

At runtime, use the `setOptionType` method to modify this property.

#### Settings

Valid value	Description
DEFAULT_OPTION	(Default) Uses default buttons.
YES_NO_OPTION	Uses buttons with Yes and No labels.
YES_NO_CANCEL_OPTION	Uses buttons with Yes, No, and Cancel labels.
OK_CANCEL_OPTION	Uses buttons with OK and Cancel labels.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the component to display either horizontally or vertically.

#### Runtime modification

At runtime, use the `setOrientation` method to modify this property.

#### Settings

Valid value	Description
VERTICAL	Specifies vertical orientation.
HORIZONTAL	(Default) Specifies horizontal orientation.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the component to display either horizontally or vertically.

#### Runtime modification

At runtime, use the `setOrientation` method to modify this property.

#### Settings

Valid value	Description
VERTICAL_SPLIT	Specifies vertical orientation.
HORIZONTAL_SPLIT	(Default) Specifies horizontal orientation.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether labels are painted on the slider.

#### Runtime modification

At runtime, use the `setPaintLabels` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Labels are not painted on the slider.
true	Labels are painted on the slider.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether tick marks are painted on the slider.

#### Runtime modification

At runtime, use the `setPaintTicks` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Tick marks are not painted on the slider.
true	Tick marks are painted on the slider.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the component is currently painting a tile.

Use this method to keep some state you might need between tiles.

#### Runtime modification

At runtime, use the `isPaintingTile` method to modify this property.

#### Settings

Valid values	Description
false	(Default) The component is not painting a tile or if the last tile is painted.
true	Paint is called again for another tile.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Controls the number of decimal places displayed along the y-axis.

**Runtime modification**

At runtime, use `setPrecision` method to modify this property.

**Settings**

The default is 0. Valid values are positive integers.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Computes the size of the viewport needed to display visibleRowCount rows.

**Settings**

The default is a set of X,Y coordinate integers.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the cell prototype value, used to compute cell width and height.

**Runtime modification**

At runtime, use the `setPrototypeCellValue` method to modify this property.

**Settings**

The default is Null.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the renderer that paints the item selected from the list in the JComboBox field. The renderer is used if the JComboBox is not editable. If it is editable, the editor is used to render and edit the selected item.

**Runtime modification**

At runtime, use the `setRenderer` method to modify this property.

**Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether the component can obtain the focus by calling `requestFocus`.

#### Runtime modification

At runtime, use the `setRequestFocusEnabled` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Focus obtained by calling <code>requestFocus</code> .
false	Focus obtained without calling <code>requestFocus</code> .

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether rollover effects should be enabled.

#### Runtime modification

At runtime, use the `setRolloverEnabled` method to modify this property.

#### Settings

Valid values	Description
true	Rollover effects are enabled.
false	(Default) Rollover effects are not enabled.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether or not the root node from the `TreeModel` is visible.

#### Runtime modification

At runtime, use the `setRootVisible` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Root node is visible.
false	Root node is not visible.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of rows displayed by a text component.

**Runtime modification**

At runtime, use the `setRow` method to modify this property.

**Settings**

The default is 0. Any integer value above 0 is also acceptable.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the number of rows that are currently being displayed in the component.

**Settings**

The default is an integer.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the height of each cell of the component.

#### Runtime modification

At runtime, use the `setRowHeight` method to modify this property.

#### Settings

Valid values	Description
16	(Default) The row height is set to 16
1-32767	Use a valid number to set the row height.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether the rows in this model can be selected.

#### Runtime modification

At runtime, use the `setRowSelectionAllowed` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Rows may be selected.
false	Rows may not be selected.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines when scrollbars appear in the scrollpane.

#### Settings

---

### HorizontalScrollBarPolicy property

Determines when the horizontal scrollbar appears in the scrollpane.

#### Runtime modification

At runtime, use the `setHorizontalScrollBarPolicy` method to modify this property.

#### Settings

Valid values	Description
HORIZONTAL_SCROLLBAR_AS_NEEDED	(Default) The horizontal scrollbar appears when the text extends beyond the scrollpane.
HORIZONTAL_SCROLLBAR_NEVER	The horizontal scrollbar never appears.
HORIZONTAL_SCROLLBAR_ALWAYS	The horizontal scrollbar always appears.

### VerticalScrollBarPolicy property

Determines when the vertical scrollbar appears in the scrollpane.

#### Runtime modification

At runtime, use the `setVerticalScrollBarPolicy` method to modify this property.

#### Settings

Valid values	Description
VERTICAL_SCROLLBAR_AS_NEEDED	(Default) The vertical scrollbar appears when the rows of text extend beyond the scrollpane.
VERTICAL_SCROLLBAR_NEVER	The vertical scrollbar never appears.
VERTICAL_SCROLLBAR_ALWAYS	The vertical scrollbar always appears.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the scroll offset.

**Runtime modification**

At runtime, use the `setScrollOffset` method to modify this property.

**Settings**

The default is 0. Use an integer value.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets whether the component should track the viewport's height.

#### Runtime modification

At runtime, use the `getScrollableTracksViewportHeight` method to modify this property.

#### Settings

Valid values	Description
true	Tracks the height of the viewport.
false	(Default) Does not track the height of the viewport.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets whether the component should track the viewport's width.

#### Runtime modification

At runtime, use the `getScrollableTracksViewportWidth` method to modify this property.

#### Settings

Valid values	Description
true	Tracks the width of the viewport.
false	(Default) Does not track the width of the viewport.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Set the default state of the component to either selected (true) or not selected (false).

#### Runtime modification

At runtime, use the `setSelected` method to modify this property.

#### Settings

Valid values	Description
true	Component is selected or a toggle is visible.
false	(Default) Component is not selected or no toggle is visible.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the index of the last column selected or added to the selection.

**Settings**

The default is -1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of selected columns.

**Settings**

The default is 0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the selected component for this tabbed pane.

#### **Runtime modification**

At runtime, use the `setSelectedComponent` method to modify this property.

#### **Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the first selected index.

**Runtime modification**

At runtime, use the `getSelectedIndex` method to modify this property.

**Settings**

The default is -1.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the selected item in the JComboBox.

**Runtime modification**

At runtime, use the `setSelectedItem` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the index of the last row selected or added to the selection.

**Settings**

The default is -1.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the number of selected rows.

**Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the portion of the text that is selected.

### **Settings**

The default is Null.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the current color used to render the selected text.

### Runtime modification

At runtime, use the `setSelectedTextColor` method to modify this property.

### Settings

Valid values	Description
black	(Default) Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The default selection text.

#### **Runtime modification**

At runtime, use the `setSelection` method to modify this property.

#### **Settings**

The default is Null.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Controls the settings for the pane components such as JTextPane. Some selection properties controls cell, row, and column selection properties for the [JTable component](#) only.

## Settings

---

### CellSelectionEnabled property (JTable only)

Determines cell selection behavior in the component.

#### Runtime modification

At runtime, use the `setCellSelectionEnabled` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Selecting a single cell also selects entire row and/or column.
true	Only the chosen cell is selected.

### ColumnSelectionAllowed property (JTable only)

Determines whether cell selection also selects a column in the component.

#### Runtime modification

At runtime, use the `setColumnSelectionAllowed` method to modify this property.

#### Settings

Valid values	Description
false	(Default) Selecting a cell in a column does not select the entire column.
true	Selecting a cell in a column selects the entire column.

### RowSelectionAllowed property (JTable only)

Sets whether the rows in this model can be selected.

#### Runtime modification

At runtime, use the `setRowSelectionAllowed` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Rows may be selected.
false	Rows may not be selected.

### SelectedColumn property (JTable only)

Sets the index of the last column selected or added to the selection.

#### Settings

The default is -1.

This is a read only property.

## SelectedColumnCount property (JTable only)

Sets the number of selected columns.

### Settings

The default is 0.

This is a read only property.

## SelectedRow property (JTable only)

Sets the index of the last row selected or added to the selection.

### Settings

The default is -1.

This is a read only property.

## SelectedRowCount property (JTable only)

Sets the number of selected rows.

### Settings

The default is 0.

This is a read only property.

## SelectedText property

Returns the portion of the text that is selected.

### Settings

The default is Null.

This is a read only property.

## SelectedTextColor property

Sets the current color used to render the selected text.

### Runtime modification

At runtime, use the `setSelectedTextColor` method to modify this property.

### Settings

Valid values	Description
black	(Default) Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

## SelectionBackground property (JTable only)

Sets the background color for selected cells.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setSelectionBackground` method to modify this property.

### Settings

The default value of this property is defined by the look and feel implementation.

Valid values	Description
black	Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

## SelectionColor property (JTable only)

Sets the current color used to render the selection. A null selection make the color white.

### Runtime modification

At runtime, use the `setSelectionColor` method to modify this property.

### Settings

Valid values	Description
black	Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

The default is 204,204,255 for JPasswordField only.

## SelectionForeground property (JTable only)

Sets the foreground color for selected cells.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setSelectionForeground` method to modify this property.

### Settings

The default value of this property is defined by the look and feel implementation.

Valid values	Description
black	Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

### SelectionModel property (JTable only)

Sets the component's selection model.

### Runtime modification

At runtime, use the `setSelectionModel` method to modify this property.

### Settings

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the background color for selected cells.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setSelectionBackground` method to modify this property.

### Settings

The default value of this property is defined by the look and feel implementation.

Valid values	Description
black	Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the current color used to render the selection. A null selection make the color white.

#### Runtime modification

At runtime, use the `setSelectionColor` method to modify this property.

#### Settings

Valid values	Description
black	Makes the component text black.
blue	Makes the component text blue.
cyan	Makes the component text cyan.
darkGray	Makes the component text dark gray.
gray	Makes the component text gray.
green	Makes the component text green.
lightGray	Makes the component text light gray.
magenta	Makes the component text magenta.
orange	Makes the component text orange.
pink	Makes the component text pink.
red	Makes the component text red.
white	Makes the component text white.
yellow	Makes the component text yellow.

The default is 204,204,255 for JPasswordField.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the number of nodes selected.

**Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns true if nothing is selected.

#### Settings

Valid values	Description
true	(Default) Nothing is selected.
false	A selection was made.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the foreground color for selected cells.

To set a custom color, choose Custom from the drop-down in the Property List window and choose a color from the color palette.

### Runtime modification

At runtime, use the `setSelectionForeground` method to modify this property.

### Settings

The default value of this property is defined by the look and feel implementation.

Valid values	Description
black	Makes the cell background black.
blue	Makes the cell background blue.
cyan	Makes the cell background cyan.
darkGray	Makes the cell background dark gray.
gray	Makes the cell background gray.
green	Makes the cell background green.
lightGray	Makes the cell background light gray.
magenta	Makes the cell background magenta.
orange	Makes the cell background orange.
pink	Makes the cell background pink.
red	Makes the cell background red.
white	Makes the cell background white.
yellow	Makes the cell background yellow.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether single-item or multiple-item selections are allowed in the component.

#### Runtime modification

At runtime, use the `setSelectionMode` method to modify this property.

#### Settings

Valid values	Description
SINGLE_SELECTION	Only one item can be selected at a time
SINGLE_INTERVAL_SELECTION	One contiguous group of items can be selected at a time.
MULTIPLE_INTERVAL_SELECTION	No restrictions on what can be selected.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the component's selection model.

**Runtime modification**

At runtime, use the `setSelectionModel` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Selects the node identified by the specified path.

**Runtime modification**

At runtime, use the `setSelectionPath` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the character used to separate numbers when the value is greater than one thousand (1,000).

#### **Runtime modification**

At runtime, use `setSeparator` method to modify this property.

#### **Settings**

The default is a comma. Use a period for most international formats.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines whether the separator is visible.

#### Runtime modification

At runtime, use `setSeparatorEnabled` method to modify this property.

#### Settings

Valid values	Description
true	(Default) The separator is visible.
false	The separator is invisible.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if you want a grid behind the graph.

#### Runtime modification

At runtime, use the `setShowGrid` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Grid will display.
false	No grid will display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether horizontal lines should be drawn in between the cells.

#### Runtime modification

At runtime, use the `setShowHorizontalLines` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Lines will be drawn between cells.
false	No lines will be drawn between cells.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if you want a legend for the data displayed. ShowLegend uses the column headings created in the model as the text source.

#### Runtime modification

At runtime, use the `setShowLegend` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Legend will display.
false	No legend will display.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether vertical lines should be drawn in between the cells.

#### Runtime modification

At runtime, use the `setShowVerticalLines` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Lines will be drawn between cells.
false	No lines will be drawn between cells.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether the node handles are to be displayed.

#### Runtime modification

At runtime, use the `setShowsRootHandles` method to modify this property.

#### Settings

Valid values	Description
true	Node handles are displayed.
false	(Default) Does not display node handles.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if the knob (and the data value it represents) resolve to the closest tick mark next to where the user positioned the knob.

#### Runtime modification

At runtime, use the `setSnapToTicks` method to modify this property.

#### Settings

Valid values	Description
true	(Default) Snaps the knob to the nearest tick mark.
false	Does not snap the knob to the nearest tick mark.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the component above, below, to the left or to the right of the divider.

### Settings

---

#### BottomComponent property

Sets the component below, or to the right of the divider.

#### Runtime modification

At runtime, use the `setBottomComponent` method to modify this property.

#### Settings

The default is Default.

#### LeftComponent property

Sets the component to the left (or above) the divider.

#### Runtime modification

At runtime, use the `setLeftComponent` method to modify this property.

#### Settings

The default is Default.

#### RightComponent property

Sets the component to the right (or below) the divider.

#### Runtime modification

At runtime, use the `setRightComponent` method to modify this property.

#### Settings

The default is Default.

#### TopComponent property

Sets the component above, or to the left of the divider.

#### Runtime modification

At runtime, use the `setTopComponent` method to modify this property.

#### Settings

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the style of graph used to display the data.

#### Runtime modification

At runtime, use the `setStyle` method to modify this property.

#### Settings

Valid values	Description
LINE_STYLE(Default)	Displays data in a Line Chart.
BAR_STYLE	Displays data in a Bar Chart.
PIE_STYLE	Displays data in a Pie Chart.
AREA_STYLE	Displays data in an Area Chart.
SCATTER_STYLE	Displays data in a Scatter Chart.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Associates the editor with a text document. The currently registered factory is used to build a view for the document, which gets displayed by the editor.

**Runtime modification**

At runtime, use the `setStyledDocument` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the currency symbol used with a value.

**Runtime modification**

At runtime, use `setCurrencySymbol` method to modify this property.

**Settings**

The default is \$. Any international currency symbol may be used.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Returns the number of tabs in this tabbedpane.

### **Settings**

The default is 0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the tab placement for this tabbedpane.

#### Runtime modification

At runtime, use the `setTabPlacement` method to modify this property.

#### Settings

Valid values	Description
TOP	(Default) The tabs appear at the top of the pane.
BOTTOM	The tabs appear at the bottom of the pane.
LEFT	The tabs appear to the left of the pane.
RIGHT	The tabs appear to the right of the pane.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the number of rows if the `TabPlacement` is `TOP` or `BOTTOM` or the number of columns if `TabPlacement` is `LEFT` or `RIGHT`, or 0 if there is no UI set on this tabbedpane.

### **Settings**

The default is 0.

This is a read only property.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the number of characters to expand tabs to.

#### Runtime modification

At runtime, use the `setTabSize` method to modify this property.

#### Settings

Valid values	Description
8	(Default) The tab size is set to 0.
1-32767	Valid values are within this range.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the horizontal alignment of the text in the wrapping label

#### Runtime modification

At runtime, use `setTextAlignment` method to modify this property.

#### Settings

Valid values	Description
CENTER	The text is centered.
LEFT	(Default) The text is left justified.
RIGHT	The text is right justified

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the title that will display above the chart.

**Runtime modification**

At runtime, use the `setTitle` method to modify this property.

**Settings**

The default is null.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Defines the text strings to use for each tab label in a `TabbedPane` component.

#### Runtime modification

At run time, use the `setTitles` method to modify this property.

#### Settings

Valid values	Description
[empty]	There is no data currently in the component.
[list]	The component currently has a list of values.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Provides a short description that pops up when the cursor lingers over a component.

**Runtime modification**

At runtime, use the `setToolTipText` method to modify this property.

**Settings**

Any string may be used.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the menu is a 'top-level menu', that is, if it is the direct child of a menubar.

**Settings**

The default is false.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the name of the UIFactory class that generates the look and feel for this component.

**Runtime modification**

At runtime, use the `getUIClassID` method to modify this property.

**Settings**

This property is read only.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the increment unit for a JScrollBar component. This is the amount that is added or subtracted from this scroll bar's Value property when the user presses the up or down scroll arrow.

#### Runtime modification

Use the `setUnitIncrement` method to modify this property.

#### Settings

Valid values	Description
1	(Default)
0-32767	Valid values are within this range.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines if the entire tree beginning with this root will be validated.

JScrollPane overrides this method and returns true.

#### Runtime modification

At runtime, use the `isValidRoot` method to modify this property.

#### Settings

Valid values	Description
true	Entire tree beginning with this root is validated.
false	(Default) Entire tree is not validated starting from this root.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if a scrollbar knob is being dragged.

Scrollbar look and feel implementations should set this property to true when a knob drag begins and to false when the drag ends.

#### Runtime modification

At runtime, use the `setValueIsAdjusting` method to modify this property.

#### Settings

Valid values	Description
true	Scrollbar knob is being dragged.
false	(Default) Scrollbar knob is not being dragged.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Specifies the justification of text drawn in a component.

#### Runtime modification

At runtime, use the `setVerticalAlignment` method to modify this property.

#### Settings

Valid values	Description
CENTER	(Default) The text should be in the center.
TOP	The text should be placed at the top.
BOTTOM	The text should be placed at the bottom.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines when the vertical scrollbar appears in the scrollpane.

#### Runtime modification

At runtime, use the `setVerticalScrollBarPolicy` method to modify this property.

#### Settings

Valid values	Description
VERTICAL_SCROLLBAR_AS_NEEDED	(Default) The vertical scrollbar appears when the rows of text extend beyond the scrollpane.
VERTICAL_SCROLLBAR_NEVER	The vertical scrollbar never appears.
VERTICAL_SCROLLBAR_ALWAYS	The vertical scrollbar always appears.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Positioning property. Controls the justification of text drawn in a JLabel or JButton component.

#### Runtime modification

At runtime, use the `setVerticalTextPosition` method to modify this property.

#### Settings

Valid values	Description
CENTER	(Default) The text should be centered.
LEFT	The text should be placed towards the left.
RIGHT	The text should be placed towards the right.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the Viewport's one lightweight child, view.

**Runtime modification**

At runtime, use the `setView` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the view coordinates that appear in the upper left hand corner of the viewport.

**Runtime modification**

At runtime, use the `setViewPosition` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns a Rectangle giving the visible part of the view using view coordinates.

**Settings**

The default is 0,0,0,0.

This is a read only property.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the view coordinates that appear in the upper left hand corner of the viewport, and the size of the view.

**Runtime modification**

At runtime, use the `setViewSize` method to modify this property.

**Settings**

The default is 0,0. Any integer coordinates can be used.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Add a border around the viewport.

#### **Runtime modification**

At runtime, use the `setViewportBorder` method to modify this property.

#### **Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the size of the scrollbar knob.

#### Runtime modification

At run time, use the `setVisibleAmount` method to modify this property.

#### Settings

Valid Values	Description
0	(Default)
0-32767	Valid values are within this range.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Returns the component's "visible rectangle".

**Runtime modification**

At runtime, use the `setVisibleRect` method to modify this property.

**Settings**

The default is 0,0,0,0.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the preferred number of rows in the component that can be displayed without a scrollbar.

#### **Runtime modification**

At runtime, use the `setVisibleRowCount` method to modify this property.

#### **Settings**

The default is 8 for the `JList` component.

The default is 20 for the `JTree` component.

<b>Valid values</b>	<b>Description</b>
0-32767	Valid values are within this range.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if a parent component will be provided for the user to input.

#### Runtime modification

At runtime, use the `setWantsInput` method to modify this property.

#### Settings

Valid values	Description
true	Parent component is provided.
false	(Default) No parent component is provided.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the style of wrapping used if the text area is wrapping lines.

#### Runtime modification

At runtime, use the `setWrapStyleWord` method to modify this property.

#### Settings

Valid values	Description
true	Lines will be wrapped at word boundaries.
false	(Default) Lines will be wrapped at character boundaries.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the maximum value of the Y axis in a JChart.

#### Runtime modification

At runtime, use the `setYAxisMax` method to modify this property.

#### Settings

Valid values	Description
inf	(Default) Y Axis will extend past maximum value of y coordinates.
integer	any decimal value, either positive or negative.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the minimum value of the Y axis in a JChart.

#### Runtime modification

At runtime, use the `setYAxisMin` method to modify this property.

#### Settings

Valid values	Description
integer	any decimal value, either positive or negative.
0.0	(Default) Current minimum value.

## Model properties

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the names of the columns in a table model.

**Runtime modification**

At runtime, use the `setColumnHeaders` method to modify this property.

**Settings**

The default is Null.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the type of delimiter the file is using to separate items.

#### Runtime modification

At runtime, use `setDelimiter` method to modify this property.

#### Settings

Valid values	Description
COMMA_DELIMITED	Each field in the table is seperated by a comma.
TAB_DELIMITED	(Default) Each field in the table is seperated by a tab.
SPACE_DELIMITED	Each field in the table is seperated by a space.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the initial value displayed in a text component using the `UnsignedNumberDocument` model component.

#### Runtime modification

At runtime, use the `setDefaultValue` method to modify this property.

#### Settings

Valid values	Description
integer	any decimal value, either positive or negative.
0	(Default) Current value.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines whether a default value is displayed in a text component using the UnsignedNumberDocument model component.

#### Runtime modification

At runtime, use `setDefaultValueEnabled` method to modify this property.

#### Settings

Valid values	Description
true	A default value is displayed.
false	(Default) No default value is displayed.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the name of the uppermost (or root) folder in a tree.

**Runtime modification**

At runtime, use `setRootName` method to modify this property.

**Settings**

The default is root.

## Border properties

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sels the type of bevel.

#### Runtime modification

At runtime, use the `setBevelType` method to modify this property.

#### Settings

Valid values	Description
LOWERED	Bevel border looks lowered.
RAISED	(Default) Bevel border looks raised.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the border color.

**Runtime modification**

At runtime, use the `setColor` method to modify this property.

**Settings**

The default is Black.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets whether Etched border is lowered or raised.

#### Runtime modification

At runtime, use the `setEtchType` method to modify this property.

#### Settings

Valid values	Description
LOWERED	(Default) Etched border looks lowered.
RAISED	Etched border looks raised.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the inner highlight border color.

#### **Runtime modification**

At runtime, use the `setHighlightInnerColor` method to modify this property.

#### **Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the outer highlight border color.

**Runtime modification**

At runtime, use the `setHighlightOuterColor` method to modify this property.

**Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the highlight border color.

**Runtime modification**

At runtime, use the `setHighlight` method to modify this property.

**Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the border type for the inside border.

**Runtime modification**

At runtime, use the `setInsideBorder` method to modify this property.

**Settings**

The default is Default.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the border Line color.

#### **Runtime modification**

At runtime, use the `setLineColor` method to modify this property.

#### **Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the border type for the outside border.

**Runtime modification**

At runtime, use the `setOutsideBorder` method to modify this property.

**Settings**

The default is Default.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines if the corners of the border are rounded.

#### Runtime modification

At runtime, use the `setRoundedCorners` method to modify this property.

#### Settings

Valid values	Description
true	Corners are rounded.
false	(Default) Corners are not rounded.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the shadow border color.

**Runtime modification**

At runtime, use the `setShadowColor` method to modify this property.

**Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the inner shadow border color.

#### **Runtime modification**

At runtime, use the `setShadowInnerColor` method to modify this property.

#### **Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the outer shadow border color.

**Runtime modification**

At runtime, use the `setShadowOuterColor` method to modify this property.

**Settings**

The default is Black.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the border thickness.

**Runtime modification**

At runtime, use the `setThickness` method to modify this property.

**Settings**

The default is 1. Use an integer value to set the thickness.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the name of the instance of ImageIcon you want to use in a matte border.

**Runtime modification**

At runtime, use the `setTileIcon` method to modify this property.

**Settings**

The default is None.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the color of the border title.

#### **Runtime modification**

At runtime, use the `setTitleColor` method to modify this property.

#### **Settings**

The default is the custom color 102,102, 153.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Determines the font for a border title. This property is broken into the subproperties that are accessed from the TitleFont property dialog box.

#### Runtime modification

At run time, the `setTitleFont` method is typically used to modify this property.

---

#### Font Name

Specifies the name of the font displaying text within a border. Available fonts are listed in the dialog box the default font is Dialog.

#### Font Size

Sets the font size in points.

Valid values	Description
--------------	-------------

12	(Default) 12 point font.
----	--------------------------

1-32767	Valid points sizes are within this range.
---------	---

#### Font Style

Sets the font style.

Valid values	Description
--------------	-------------

Plain	no style applied to title.
-------	----------------------------

Italic	Italic applied to title.
--------	--------------------------

Bold	(Default) Bold applied to title.
------	----------------------------------

BoldItalic	Bold and Italic applied to title
------------	----------------------------------

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the title-justification of the titled border.

#### Runtime modification

At runtime, use the `setTitleJustification` method to modify this property.

#### Settings

Valid values	Description
CENTER	The text should be centered.
LEFT	(Default) The text should be placed towards the left.
RIGHT	The text should be placed towards the right.
DEFAULT_JUSTIFICATION	The text should be placed towards the left.

{ewc HLP25632,HLP256\_UPPER\_LEFT,beans.bmp}



Sets the title-position of the titled border.

#### Runtime modification

At runtime, use the `setTitlePosition` method to modify this property.

#### Settings

Valid values	Description
ABOVE_TOP	The text is drawn above the top of the border.
TOP (Default)	The text is drawn at the top of the border.
BELOW_TOP	The text is drawn below the top of the border.
ABOVE_BOTTOM	The text is drawn above the bottom of the border.
BOTTOM	The text is drawn at the bottom of the border.
BELOW_BOTTOM	The text is drawn below the bottom of the border.
DEFAULT_POSITION	The text is drawn at the top of the border.

## Action properties

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Sets the name of the action object.

**Runtime modification**

At runtime, use the `setActionName` method to modify this property.

**Settings**

The default is Null.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Determines the image used for the component, usually a button.

**Runtime modification**

At runtime, use the `setSmallIcon` method to modify this property.

**Settings**

The default is Default.

## Icons properties

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



The URL of the image you are displaying.

#### **Runtime modification**

At runtime, use the `setImageLocation` method to modify this property.

#### **Settings**

The default is Null.

```
{ewc HLP25632,HLP256_UPPER_LEFT,beans.bmp}
```



Set the image observer for the image. Use this property when the ImageIcon contains an animated GIF.

**Runtime modification**

At runtime, use the `setImageObserver` method to modify this property.

**Settings**

The default is Default.

