No Help

There is no help available on this topic.

Top of file F1_Edit

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.

Using the Visual Cafe window

{button Concepts,AL(`Visual_Cafe_welcome_overview;Menus_F1;Toolbars_visual_F1;Palette_F1')} {button How To,AL(`Opening_a_Project_howto',0,`',`')}

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.

Using Visual Cafe menus

{button Concepts,AL(`Visual_Cafe_window_F1')}     {button See Also,AL(`Visual_Cafe_Tools_overview;Toolbars_visual_F1',0,`',`')}

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

These are the standard menus that are always available:

File	Edit	Window
Search	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:

Insert	Project	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.

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.

Help menu

{button See Also,AL(`Menus_F1',0,`',`')}

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.

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.

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.

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.

File menu

{button Concepts,AL(`New_Project_dialog_F1;New_File_Dialog_F1;Find_in_Files_dialog_F1;Environment_Options_Dialog_F1',0,`',`')} {button See Also,AL(`Menus_F1',0,`',`')}

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

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, or Java source file (**.java**).

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.

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.”

Print Setup, Print...

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

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, or Java source file (**.java**).

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.

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

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.

Edit menu

{button How To,AL(`Using_the_Menu_Editor_F1',0,`',`')}     {button See Also,AL(`Menus_F1',0,`',`')}

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:

If the active window is	Selects all
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:

If the active window is	Selects all
Form Designer	components
Source window	source in the current method
Breakpoint Window	breakpoints in the list

Shortcut: CONTROL + A

Project menu

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Menus_F1;Project_Options_Dialog_F1;Projects_overview',0,`',`')}

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.

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.

JAR

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

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.

Add *Item*

Adds 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, 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.

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.

JAR

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

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.

Add *Item*

Adds 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, and directory options.

Insert menu

{button How To,AL(`Adding_a_new_file_to_the_project_howto;Class_Browser_using_F1;Adding_Files_to_a_Project_howto',0,`',`')} {button See Also,AL(`Menus_F1;Insert_form_dialog_F1;New_Project_dialog_F1;Insert_Object_dialog_F1;Insert_class_dialog_F1;Insert_Member_dialog_F1',0,`',`')}

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

Component into Library…

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

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.

Component into Library…

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

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

Tools menu

{button Concepts,AL(`Environment_Options_Dialog_F1',0,`',`')} {button See Also,AL(`Menus_F1',0,`',`')}

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.

Version Control...

Choose a version control system to use 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.

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.

Window menu

{button Concepts,AL(`Workspace_customization_overview;Visual_Cafe_Tools_overview;Debugging Your Program',0,`',`')} {button How To,AL(`Workspaces_AUD_howto',0,`',`')} {button See Also,AL(`Menus_F1;Class_Browser_using_F1;Hierarchy_Editor_F1;Property_Inspector_using_F1;Object_Library_F1;F1_Breakpoints_Window;F1_Call_Stack_Window;F1_Messages_Window;F1_Thread_Window;F1_Variables_Window;F1_Watch_Window',0,`',`')}

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.

Workspaces…

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

New…	allows you to create a new workspace.
Rename…	allows you to rename a workspace.
Delete	allows you to delete a workspace.

You can activate the display of these tools:

Property List	Component Library
Hierarchy Editor	Class Browser

You can activate the display of these debugging windows:

Watch	Breakpoint
Messages	Threads
Variables	Call Stack

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.

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.

Workspaces…

workspace names

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

Edit	refers to the development workspace.
Debug	refers to the debug workspace.
New…	allows you to create a new workspace.
Rename…	allows you to rename a workspace.
Delete	allows you to delete a workspace.

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

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.

Object menu

{button See Also,AL(`Menus_F1;Source_Window_F1;Using_the_Interaction_Wizard_F1;Add_to_Library_dialog_F1',0,`',`')}

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.

Add to Library...

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

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.

Add to Library...

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

▸

Layout menu

{button See Also,AL(`Menus_F1',0,`',`')}

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.

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.

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."

Invisibles

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

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.

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."

Invisibles

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

Search menu

{button See Also,AL(`Menus_F1;Find_dialog_F1;Find_in_Files_dialog_F1;Compare_Files_Dialog_F1;Members_window_F1;Setting_a_Conditional_Breakpoint;Goto_Line_Window_F1;Goto_Function_F1;Bookmark_window_F1',0,`',`')}

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.

Source menu

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Menus_F1;F1_Watch_Window;Setting_a_Conditional_Breakpoint;Goto_Line_Window_F1;Bookmark_window_F1;Format_Options_Dialog_F1',0,`',`')}

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.

Set 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.

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.

Set 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.

Debug menu

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Menus_F1;Running_a_Project_howto;Stepping_Code_After_a_Breakpoint;Running_to_the_Cursor_Location;Running_to_Program_End;Project_Options_dialog_F1',0,`',`')}

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)

Variables menu

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Menus_F1;Projects_overview;Goto_Line_Window_F1;Bookmark_window_F1',0,`',`')}

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 is there is no selection and adds an entry to the Watch window.

Breakpoint menu

{button See Also,AL(`Menus_F1;Projects_overview',0,`',`')}

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.

Threads menu

{button See Also,AL(`Menus_F1;Projects_overview',0,`',`')}

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.

Calls menu

{button See Also,AL(`Menus_F1;Projects_overview',0,`',`')}

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.

Hierarchy menu

{button Concepts,AL(`Hierarchy_Editor_F1',0,`',`')}     {button How To,AL(`Deleting_a_Class_Inheritance_howto',0,`',`')}   {button See Also,AL(`Insert_class_dialog_F1',0,`',`')}

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.

Classes menu

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Class_Browser_using_F1;Menus_F1;Insert_class_dialog_F1;Insert_class_dialog_F1;WinStyle_Controlling_Class_and_Member_Display_F1;WinStyle_Grouping_and_Sorting_Class_and_Members_F1',0,`',`')}

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

Using the Form Designer pop-up menu

{button See Also,AL(`Menus_F1;Forms_adding_objects;Arranging_Components_on_a_Form_howto;Connecting_Form_components_howto;Source_Window_F1;Using_the_Interaction_Wizard_F1;Add_to_Library_dialog_F1',0,`',`')}

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

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.

Using the Object View pop-up menu

{button Concepts,AL(`Projects_overview',0,`',`')}     {button See Also,AL(`Adding_objects_to_project_howtoMenus_F1;Add_Remove_Files_Dialog_F1;Connecting_Form_components_howto',0,`',`')}

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.

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.

Using the Package View pop-up menu

{button Concepts,AL(`Projects_overview;Source_Window_F1',0,`',`')}     {button See Also,AL(`Menus_F1;Add_Remove_Files_Dialog_F1',0,`',`')}

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

Opens the selected source file in the Source window.

Insert/Remove Files…

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

Using the Project tabs pop-up menu

{button Concepts,AL(`Projects_overview;Source_Window_F1',0,`',`')}     {button See Also,AL(`Menus_F1;Add_Remove_Files_Dialog_F1',0,`',`')}

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.

Using the Class Pane pop-up menu

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`Class_Browser_using_F1;Menus_F1;Insert_class_dialog_F1;Insert_class_dialog_F1;WinStyle_Controlling_Class_and_Member_Display_F1;WinStyle_Grouping_and_Sorting_Class_and_Members_F1',0,`',`')}

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.

Using the Member Pane pop-up menu

{button Concepts,AL(`Class_Browser_using_F1',0,`',`')}     {button How To,AL(`Adding_methods_ClassBrowser_howto',0,`',`')}
{button See Also,AL(`Menus_F1;Member_Attributes_dialog_F1;Insert_Member_dialog_F1;WinStyle_Controlling_Class_and_Member_Display_F1;WinStyle_Grouping_and_Sorting_Class_and_Members_F1',0,`',`')}

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.

Using the Hierarchy Editor pop-up menu

{button See Also,AL(`Menus_F1;Hierarchy_Editor_F1;Insert_class_dialog_F1',0,`',`')}

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

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.

Using the Component Library pop-up menu

{button See Also,AL(`Menus_F1',0,`',`')}

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.

Using the Editing Window pop-up menu

{button See Also,AL(`Menus_F1;Source_Window_F1;Class_Browser_F1;Members_window_F1',0,`',`')}

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.

Add Variable

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

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

Using Visual Cafe toolbars

{button How To,AL(`Controlling_Position_and_Visibility_howto',0,`',`')}     {button See Also,AL(`Toolbars_standard_F1;Toolbars_layout_F1;Toolbars_views_F1;Toolbars_debug_F1;Visual_Cafe_window_F1',0,`',`')}

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

Using the Standard toolbar

{button How To,AL(`Adding_a_new_file_to_the_project_howto',0,`',`')}     {button See Also,AL(`File_menu_F1;Applets_step_overview;Applications_step_overview',0,`',`')}

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

Using the Layout toolbar

{button See Also,AL(`Form_Designer_F1;Layout_Menu_F1',0,`',`')}

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

Using the Views toolbar

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Projects_overview;Property_Inspector_using_F1;Class_Browser_using_F1;Hierarchy_Editor_F1;Window_Menu_F1',0,`',`')}

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

Using the Workspace toolbar

{button Concepts,AL(`Workspace_customization_overview',0,`',`')}

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.

Using the Debug toolbar

{button See Also,AL(`Debugging Your Program',0,`',`')}

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

Using the Add to Library dialog box

{button Concepts,AL(`Object_Library_F1')}

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.

▸

Using the Insert/Remove Files dialog box

{button Concepts,AL(`Projects_overview',0,`',`')}     {button See Also,AL(`Adding_Files_to_a_Project_howto;Adding_Bean_to_Library_howto;Projects_overview',0,`',`')}

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.

Using the Create Project Template dialog box

{button Concepts,AL(`Working_with_Templates_overview',0,`',`')}     {button See Also,AL(`Creating_a_Project_howto',0,`',`')}

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

Using the Conditional Breakpoint dialog box

{button Concepts,AL(`Setting Breakpoints',0,`',`')}

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.

▶

Using the Evaluate Expression dialog box

{button See Also,AL(`Watching a Variable or Expression ;Watching Variables and Expressions',0,`',`')}

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.

Using the Images dialog box

{button See Also,AL(`SlideShow component;URL property;',0,`',`')}

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.

Using the Insert Applet dialog box

{button Concepts,AL(`Applets_step_overview;Projects_overview')}     {button See Also,AL(`Creating_a_Template_howto;Deleting_Objects_from_the_Object_Library_howto',0,`',`')}

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 uose 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

▸

Using the Insert Form dialog box

{button See Also,AL(`Form_Designer_F1',0,`',`')}

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.

Using the Insert Member dialog box

{button See Also,AL(`Class_Browser_using_F1;Member_Attributes_dialog_F1',0,`',`')}

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.

Using the Insert Object dialog box

{button Components,JI(`VCAFE.HLP',`Visual_Components_and_Containers')}

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.

Tasks

Adding objects to a form

Adding objects to a project

Adding objects to the Palette

Using the New Project dialog box

{button Concepts,AL(`Projects_overview')}     {button See Also,AL(`Creating_a_Template_howto;Popup_menu_New_Project__F1',0,`',`')}

File ▸ New Project

The New Project dalog 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

▸

Using the Open File dialog box

{button How To,AL(`Adding_Files_to_a_Project_howto',0,`',`')}     {button See Also,AL(`Source_Window_F1',0,`',`')}

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.

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

▶

Using the Members window

{button See Also,AL(`Source_Window_F1;Search_Menu_F1;Class_Browser_using_F1',0,`',`')}

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.

▶

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.MAC is deleted from disk, Visual Cafe Basic creates a stub file when it starts up.

Macros

Lists existing macros that are associated with the current project. ScriptMaker macro files have the extension .MAC and are stored in \Bin\Macs. 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 File ▶ Macro menu.

Reorder Commands

Allows you to change the order of the macros which 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 project's 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.

Using the URL Chooser dialog box

{button See Also,AL(`URL_property;ImageURL',0,`',`')}

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.

Using the Windows dialog box

Window ▸ Windows

Use this dialog box to select a recently access 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.

Using the Open File dialog box

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

Using the Save File dialog box

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

Using the About Visual Cafe dialog box

Provides information about your Visual Cafe release.

Using the Create Derived Class dialog box

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}

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 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

Visual Cafe components and containers

{button Component List,PI(`vcafe.hlp',`Components_standard_list_popup')}

Visual Cafe helps you to design Java applets and applications, without writing source code by using a Components Library.

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 subclassed 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. For information about dbAWARE classes, such as Session, ConnectionInfo, and RelationView, see the dbANYWHERE API Reference manual. These manuals are available from the Visual Cafe Help menu.

Visual Cafe Pro components

Visual Cafe Pro provides two types of database components:

- connection components
- dbAWARE components

Connection components are invisible components that define the database connection, the database view, and how data is retrieved.

A dbAWARE component is a Visual Cafe component that has additional properties for binding to data in a database. dbAWARE components are bound to the current row in a single database column and some can be bound to another database column for component population. Some dbAWARE components provide projection binding, others provide list binding.

dbAWARE components extends classes in package symantec.itools.db.awt.

To define the data binding of a dbAWARE component, use a RelationView component.

Several of the standard Visual Cafe components have an equivalent dbAWARE component.

Tip You can quickly add dbAWARE components by using the dbAWARE Project Wizard (File ▸ New Project) and the Add Table Wizard (Insert

▸ Add Table Wizard, and the Add Table Wizard icon on the Component Palette.)

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.

Components with dbAWARE properties are available with Visual Cafe Pro only. Components that are general components and are dbAWARE components are marked with a ▲.

dbAWARE

- Add Table Wizard
- ▶Checkbox
- ▶ComboBox
- ConnectInfo
- DateTimeStamp
- ▶FormattedTextField
- Grid
- ▶ImageViewer
- ▶Label
- ▶List
- ▶NervousText
- RadioBox
- ▶RadioButton
- RecordStateLabel
- RecordNumberLabel
- RelationView
- Session
- ▶StateCheckBox
- ▶TextArea
- ▶TextField

Additional

- ▶ComboBox
- DaySpinner
- DirectionButton
- ▶FormattedTextField
- HorizontalSlider
- ImageButton
- ImageHTMLLink
- ImageListBox
- InvisibleButton
- InvisibleHTMLLink
- LabelButton
- LabelHTMLLink
- Label3D
- ListSpinner
- MonthSpinner
- MultiList
- NumericSpinner
- RollOverButton
- ScrollingText
- ▶StateCheckBox
- StatusBar
- Tstamp
- VerticalSlider
- WrappingLabel

Multimedia

- Animator
- Emblaze20
- Firework
- ▶ImageViewer
- MovingAnimation
- ▶NervousText
- Plasma
- SlideShow
- SoundPlayer

Standard

- Button
- Canvas
- ▶Checkbox
- Choice
- Horizontal Scrollbar
- ▶Label
- ▶List
- Panel
- ▶RadioButton
- ▶TextArea
- ▶TextField
- Vertical Scrollbar

Panels

- BorderPanel
- ImagePanel
- KeyPressManagerPanel
- RadioButtonGroupPanel
- ScrollingPanel
- SplitterPanel
- TabPanel
- ToolBarPanel
- ToolBarSpacer

Forms

- AboutDialog
- Applet
- AttentionDialog
- Dialog
- Frame
- Open File Dialog
- PasswordDialog
- ProgressDialog
- Save File Dialog
- Window

Menus & Items

- CheckboxMenuItem
- Menu
- MenuBar
- MenuItem

Shapes

- Circle
- Ellipse
- HorizontaLine
- Line
- Rect
- Square
- VerticalLine

Utility

- Calendar
- ProgressBar
- StatusScroller
- Timer
- TreeView

Predefined TextFields

- IntLongDistPhoneNumber
- LocalPhoneNumber
- LongZipCode
- PostalCode
- SocialInsuranceNumber
- SocialSecurityNumber
- USLongDistPhoneNumber
- ZipCode

▶

Understanding the component class

{button See Also,AL(`Container_class;Visual_Cafe_components_and_containers',0,`',`')} {button API,JI(`APIRef.hlp',`java.awt.Component.html-_top_')}

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.

Understanding the container class

{button See Also,AL(`Component_class;Visual_Cafe_components_and_containers',0,`',`')}  {button API,JI(`APIRef.hlp',`java.awt.Container.html-_top_')}

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:

Window	A blank modal container that must be set into a Frame.
Frame	A top-level window. Extends Window. It supports a title and menu bar, and can be minimized. It cannot be placed inside other containers.
Panel	A basic organizational container. It can be placed inside other containers including other panels.
Applet	A Panel that can be run by another program and supports multimedia features. Applet containers are parents to applet programs. Applets cannot be nested.

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 with dbAWARE properties

Checkbox component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`Checkbox_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_CheckBox')} {button API (standard),JI(`APIRef.hlp',`java.awt.Checkbox.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.Checkbox.html-_top_')}

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

dbAWARE features

A dbAWARE version of the Checkbox component is available in Visual Cafe Pro.

One usage for a dbAWARE checkbox is to link a check box and list box to the same column, and define True and False value for the check box. Then, when your user selects the check box option the True value displays in the list box; when the check box is cleared, the False value displays.

Although CheckBox components change their appearance to become RadioButton components when they are added to a RadioBoxGroupPanel component, it is recommended that you use the appropriate component to begin with.

Runtime modification

To add a CheckBox to a checkbox group at runtime, 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.

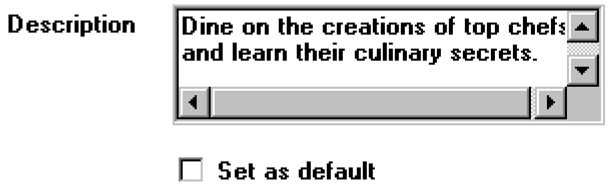
In project source code, use this syntax to create a new Checkbox:

```
checkbox1 = new java.awt.Checkbox(String label, CheckboxGroup group, boolean state);
```

Part	Type	Description
checkbox1	Checkbox	The variable name used to identify the checkbox in code.
label	String	The text to display next to the checkbox.
group	CheckBoxGroup	The CheckboxGroup, 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.



Description

Dine on the creations of top chefs
and learn their culinary secrets.

☐ Set as default

Checkbox Properties

Background
Binding (dbAWARE only)
Bounds
Class
Cursor
Enabled
False Value (dbAWARE only)
Font
Foreground
Group
Inherit Background
Inherit Font
Inherit Foreground
Label
Name
State
True Value (dbAWARE only)
Visible

ComboBox component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`ComboBox_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_ComboxBox')} {button API (standard),JI(`APIRef.hlp',`symantec.itools.awt.ComboBox.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.ComboBox.html-_top_')}

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 runtime.

dbAWARE features

A dbAWARE version of the ComboBox component is available in Visual Cafe Pro.

Although the Binding property is not called directly, four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component that this component is bound to.

Projection Name is the name of the database column that that this component is bound to.
Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

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) or RETURN (on Macs) 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.

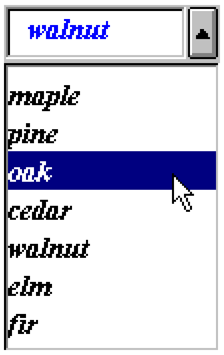
To use a different font in the ComboBox, set the Font property.

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.


walnut
maple
pine
oak
cedar
walnut
elm
fir

ComboBox Properties

Background
Bounds
Case-sensitive
Class
Cursor
Dynamic Update (dbAWARE only)
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
List Down
List Items
Name
Projection Name (dbAWARE only)
RelationView Name (dbAWARE only)
Searchable
Show Horizontal Scrollbar
Show Vertical Scrollbar
Text
Treat Blank As (dbAWARE only)
Visible

FormattedTextField component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`FormattedTextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_FormattedTextField')}   {button API (standard),JI(`APIRef.hlp',`symantec.itools.awt.FormattedTextField.html-_top_')}   {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.FormattedTextField.html-_top_')}

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. 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.

dbAWARE features

A dbAWARE version of the FormattedTextField component is available in Visual Cafe Pro.

The dbAWARE version allows you to control the display of a column value by using a text mask.

Although the Binding property is not called directly, four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column that this component is bound to.
Note   You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

Properties

To specify the required format for the text entry, use the Mask property.

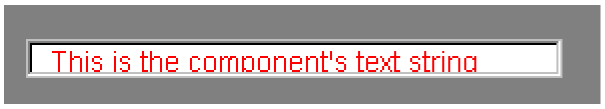
To specify a text string to display in the field by default, use Text property.

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 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, Edit Font is 14 pt., and a predefined Text string. The Field's parent container's background is gray.


This is the component's text string

FormattedTextField Properties

Background
Bounds
Caret Position
Class
Columns
Cursor
Dynamic Update (dbAWARE only)
Edit Font
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Mask
Name
Projection Name (dbAWARE only)
Selection End
Selection Start
Text
Treat Blank As (dbAWARE only)
Visible

ImageViewer component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`ImageViewer_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_ImageViewer')}   {button API (standard),JI(`APIRef.hlp',`symantec.itools.multimedia.ImageViewer.html-_top_')}   {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.ImageViewer.html-_top_')}

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.

dbAWARE features

A dbAWARE version of this component is available in Visual Cafe Pro.

Use this component to

- display images that are stored in a database column (projection name) in a Relation View,
- load an image from a non-database source, display the image, and then store the image.

Currently only JPG format is supported. The URL and Image Style properties are not support by the dbAWARE version of ImageViewer.

Properties

Two of the binding subproperties are present: RelView Name, and Projection Name.

RelView Name is the name of the RelationView component to which this component is bound.

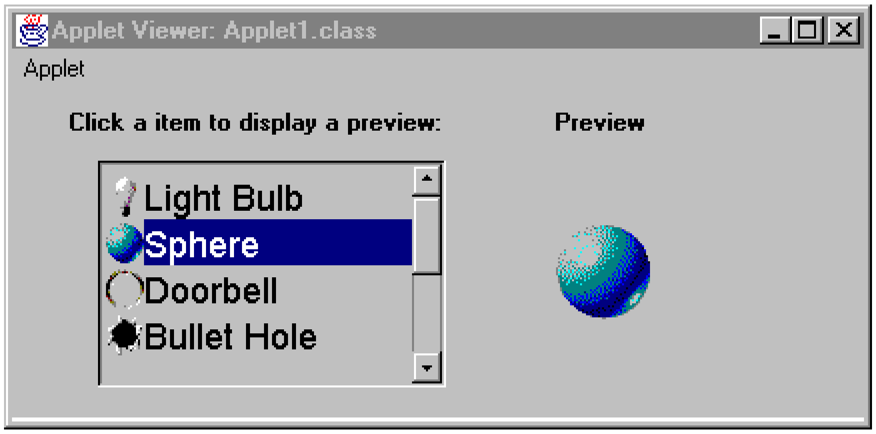
Projection Name is the name of the database column to which this component is bound.

Note: You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

To insert a non-database image, use the URL and Image Style to identify the image and define the image's centering.

Example

This picture shows an ImageListBox component (left) and an ImageViewer component (right, under Preview).


Applet Viewer: Applet1.class
Applet
Click a item to display a preview:
Preview
Light Bulb
Sphere
Doorbell
Bullet Hole

ImageViewer Properties

Background
Binding (dbAWARE only)
Bounds
Class
Cursor
Enabled
Font
Foreground
Image Style (standard only)
Inherit Background
Inherit Font
Inherit Foreground
Name
URL (standard only)
Visible

Label component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`Label_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Label')} {button API (standard),JI(`APIRef.hlp',`java.awt.Label.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.Label.html-_top_')}

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.

The following predefined dbAWARE components are provided for your convenience:

RecordNumberLabel   RecordStateLabel

dbAWARE features

A dbAWARE version of the Label component is available in Visual Cafe Pro.

Four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column that this component is bound to.
Note   You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

Using a dbAWARE label allows you to provide labels on components that change due to some criteria. dbAWARE labels function like a TextField, but are not editable. The text string for the label is obtained from the database column at runtime.

One usage of a dbAWARE label is to display different labels for form fields based on a prior language type selection. For example, if your user selects Italian as the preferred language, you can retrieve your Italian label values and use them when the form displays.

Properties

To specify the label text, type the text in the Text property .

Coding the component

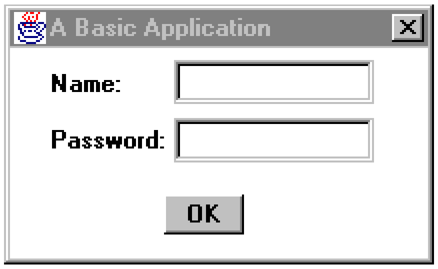
In code, use this syntax to create a new Label:

```
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.


A Basic Application
Name:
Password:
OK

Label Properties

Alignment
Background
Binding (dbAWARE only)
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Text
Visible

List component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`List_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API (standard),JI(`APIRef.hlp',`java.awt.List.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.List.html-_top_')}

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 DblClicked 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.

dbAWARE features

A dbAWARE version of the List component is available in Visual Cafe Pro.

Four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column that this component is bound to.
Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

The List component is bound to a database column (projection) and the component is populated as defined in the Lookup property.

Scrollbars display on the list box when the selected data fills the display area.

When both the Items property and Lookup properties are defined for a List component, the list of values are combined.

A master view/List component relationship is very much like a master/detail relationship. In a master/List relationship, the list changes as a master value changes. In a master/detail view, the detail changes when the master record changes, and the SQL statement is run.

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) or RETURN (on Macs) 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	`addItem(String item)`
Add items at a particular position	`addItem(String item, int index)`. 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	`removeAll()` `removeAll()`
Delete an item	`delItem(int position)` Position 0 is the first item in the list.

Get the current selected item	getSelectedIndex() or getSelectedIndexes(), for lists with multiple selection turned on.
Remove all selected items	A code sequence like the following:

```
int[] selectedItems;
selectedItems=list1.getSelectedIndexes();
for (int index=selectedItems.size; index>=0; index--)
   list1.delItem(selectedItems[index]);
```

Coding the component

To create a new List component, use this syntax:

```
list1 = new java.awt.List(rows, numSelections);
```

The second constructor 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.
NumSelections	boolean	Whether the list allows multiple selections.

The settings for *numSelections* are:

Value	Description
true	Multiple selections are allowed.
false	(Default) Only one item can be selected at a time.

List Properties

Background
Binding (dbAWARE only)
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Items
Lookup (dbAWARE only)
Multiple Mode
Name
Visible
Visible Rows

NervousText component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`NervousText_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_NervousText')} {button API (standard),JI(`APIRef.hlp',`symantec.itools.multimedia.NervousText.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.NervousText.html-_top_')}

Description

Creates animated text in which each letter moves independently of all other letters. This multimedia component is provided for novelty and programming convenience.

dbAWARE features

A dbAWARE version of the NervousText component is available in Visual Cafe Pro.

Although the Binding property is not called directly, four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component that this component is bound to.

Projection Name is the name of the database column to which this component is bound.

Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView. The text display is gotten from this column.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

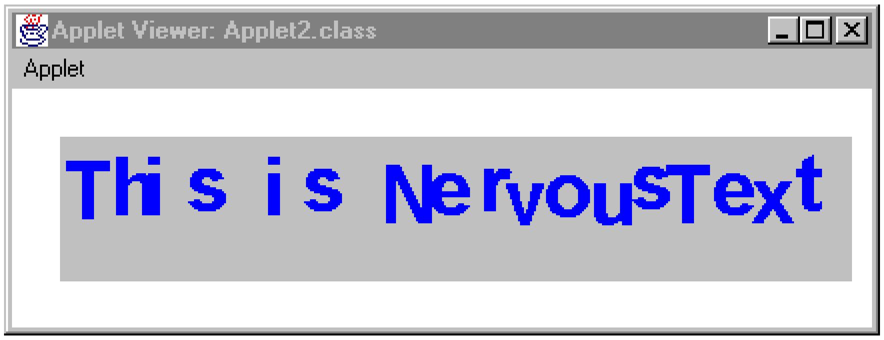
Treat Blank As defines how a component with no value (empty) is saved in the database.

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
Dynamic Update (dbAWARE only)
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Paused
Projection Name (dbAWARE only)
Text
Treat Blank As (dbAWARE only)
Visible

RadioButton component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`RadioButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_RadioButton')}   {button API (standard),JI(`APIRef.hlp',`java.awt.Checkbox.html-_top_')}   {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.RadioButton.html-_top_')}

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.

If the RadioButton will be a member of a group, make sure first to add the RadioButtonGroupPanel component from the Component Library window.

In a group of radio buttons, only one can be selected at a time. When the user selects a radio button, other radio buttons in the group are automatically deselected.

Use the Interaction Wizard to trigger an action when a RadioButton is clicked.

dbAWARE features

The column value is used as the component label value.

When you add the first dbAWARE RadioButton to a project, a RadioBox component is created.

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 Checkbox group.

```
group1 = new CheckboxGroup();
```

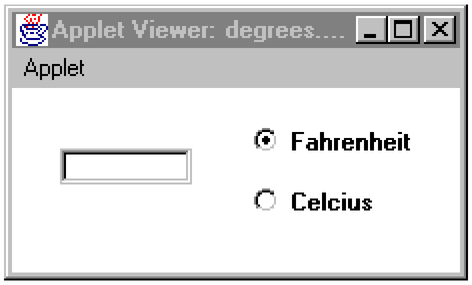
To create the radio button, create a check box and add it to a check box group. Use this syntax to create a radio button:

```
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.



RadioButton Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Group
Inherit Background
Inherit Font
Inherit Foreground
Label
Name
State
Visible

StateCheckBox component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`StateCheckBox_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API (standard),JI(`APIRef.hlp',`symantec.itools.awt.StateCheckBox.html-_top_')}   {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.StateCheckBox.html-_top_')}

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
Dynamic Update (dbAWARE only)
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Projection Name (dbAWARE only)
State
Style
Treat Blank As (dbAWARE only)
Visible

TextArea component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`TextArea_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_TextArea')} {button API (standard),JI(`APIRef.hlp',`java.awt.TextArea.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.TextArea.html-_top_')}

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.

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.

dbAWARE features

A dbAWARE version of this component is available in Visual Cafe Pro.

Use this component to display long database strings or other text data.

Four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column that this component is bound to.
Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

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.

TextArea Properties

Background
Binding (dbAWARE only)
Bounds
Class
Columns
Cursor
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Rows
Scrollbar Visibility
Text
Visible

TextField component (standard/dbAWARE)

{button Properties,PI(`vcafe.hlp',`TextField_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_TextField')} {button API (standard),JI(`APIRef.hlp',`java.awt.TextField.html-_top_')} {button API (dbAWARE),JI(`APIRef.hlp',`symantec.itools.db.awt.TextField.html-_top_')}

Description

A box in which your user can type text.

Use TextField to:

- dispay 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.

Visual Cafe Pro provides the following additional TextField component subclasses:

FormattedTextField
DBTstamp

dbAWARE features

A dbAWARE version of this component is available in Visual Cafe Pro.

Use this component to display database strings. This component can be resized, but does not have horizontal or vertical scrollbars.

Four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column that this component is bound to.
Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

See Adding a dbAWARE field to an existing form

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.

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.

TextField Properties

Background
Binding (dbAWARE)
Bounds
Class
Columns
Cursor
Echo
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Text
Visible

PRO-only Components

Add Table Wizard

The Add Table Wizard component starts the corresponding Visual Cafe Pro wizard. Use this wizard to quickly add tables to your application or form.

This component is available in Visual Cafe Pro only.

ConnectionInfo (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`ConnectionInfo_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.pro.ConnectionInfo.html-_top_')}

Description

An invisible component that defines a data source. Connection information includes the name of the data source, user login name, and user password.

This component is available in Visual Cafe Pro only.

At design time, the ConnectionInfo component is represented by the following icon:

You can drag the ConnectionInfo component from the Component Palette or Library onto a form at any location, or into the Project window. When you use the Project Wizard or the Add Table Wizard any necessary ConnectionInfo, Session and RelationView components are generated as you fill in the required information.

A project can contain more than one ConnectionInfo component. But each data source can have only one ConnectionInfo component. The ConnectionInfo component must have a Session component to connect it to the dbANYWHERE server.

See For more information on the Session class, see the dbANYWHERE API online HTML manual.

Properties

Note Do not change ConnectionInfo component properties at runtime.

Use Data Source Name property to indicate the data source to connect to.

The Name property contains the name of the actual Java object that instantiates the ConnectionInfo class. This value usually closely matches the data source name. Characters, such as spaces or punctuation in the data source name, are replaced with underscores to create a valid Java variable name.

Use the Password and User Name properties to provide logon requirements.

ConnectionInfo Properties

Auto Disconect
Class
Data Source Name
Name
Password
User Name

DateMask (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`FormattedTextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.awt.DateMask.html-_top_')}

Description

Extends FormattedTextField.

Creates a box in which your user can type text that is limited to a date. 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.

This component is available in Visual Cafe Pro only.

To post Date events or text to other components, use the Interaction Wizard.

Properties

The Mask property for the Date component is predefined and cannot be modified. If you want to use a custom mask, use a dbAWARE FormattedTextField component and define its Mask property.

DateTimeMask (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`FormattedTextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.awt.DateTmeMask.html-_top_')}

Description

Extends FormattedTextField.

Creates a box in which your user can type text that is limited to a date-time format. 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.

This component is available in Visual Cafe Pro only.

To post Date events or text to other components, use the Interaction Wizard.

Properties

The Mask property for the DateTime component is predefined and cannot be modified. If you want to use a custom mask, use a dbAWARE FormattedTextField component and define its Mask property.

DBTstamp component (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`Tstamp_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.db.awt.DBTstamp.html-_top_')}

Description

A component that creates a box in which a date or timestamp can be read or entered using any common date or time format.

This component is available in Visual Cafe Pro only.

Use DBTstamp to:

- Edit a date or a timestamp.
- Display a date or timestamp in a pre-defined display format.

To add a DBTstamp component to your project, drag the DBTstamp icon from the Palette to your Form Designer window. This component can be resized, but does not have horizontal or vertical scrollbars.

Properties

DBTstamp controls how the data is entered using the property Format on Entry.

DBTstamp controls how the data is displayed using the property Format on Display.

You can choose among Date, Time and Timestamp. This selection will set the following default formats:

• Date:	Friday October 12th, 1973
• Time:	10:34:52 PM
• Timestamp:	Friday October 12th,1973 10:34:54,000 AM

Grid component (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`Grid_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_Grid')} {button API,JI(`APIRef.hlp',`symantec.itools.db.awt.Grid.html-_top_')}

Description

A component for displaying multiple data records.

This component is available in Visual Cafe Pro only.

Provides an array, or spreadsheet-like display, of specified data. You can also specify predefined buttons that will appear at the bottom of the grid. These buttons are defined in the Binding property.

You can adjust column widths at runtime.

Column heading text can be customized in code only.

Note When you drop a Grid on a form, the Grid displays empty. You do not see data populated in the Grid until runtime.

Tip The easiest way to add a RelationView component for a Grid component is to simply drag a table name from the dbNAVIGATOR window onto a form. Visual Cafe PRO creates the RelationView and fills in its properties. Then, select the Grid component from the dbAWARE tab on the Component Palette, drag to the Form Designer, and size as needed. Last, set the Grid's RelationView element in the Binding property .

See:

Adding a Grid component

Adding a Grid component to a form as a detail view

Changing Grid cell attributes

Protecting a Grid column

Changing Grid column attributes

Defining automatic Grid row numbering

Defining automatic Grid redraw

Modifying the Grid toolbar

Properties

To associate a Grid with a table, use the RelView Name property to specify the appropriate RelationView component that is associated with the target table. Grid uses the RelationView to obtain data from a database at runtime.

Two of the binding subproperties are present: RelView Name, and Buttons.

RelView Name is the name of the RelationView component this component is bound to.

The Buttons property specifies the buttons present in the Grid component.

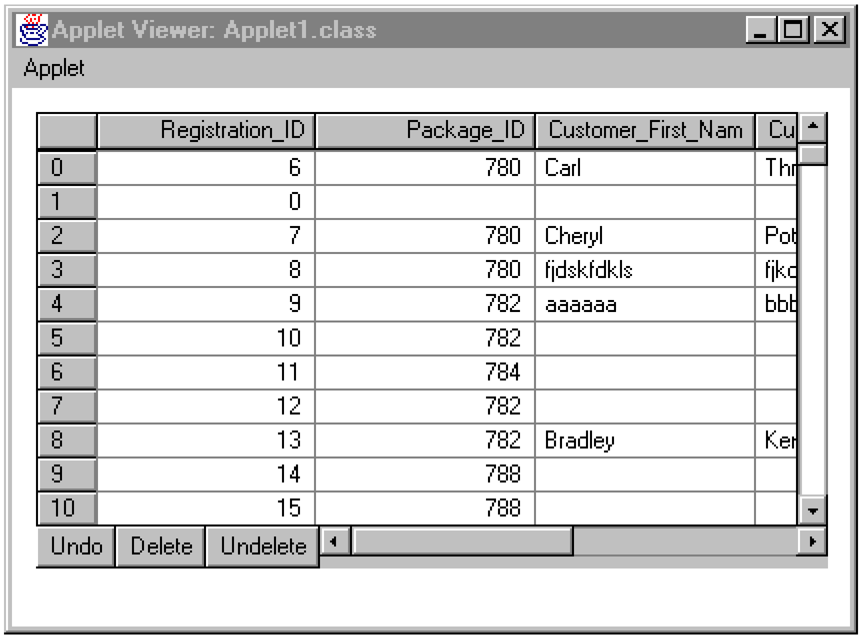
Example

The screen shot below show the default Grid for the Visual Cafe tutorial's DBA.registration table.

The columns display in the Grid, from left to right, in the same order that they appear in the dbNAVIGATOR window. If query columns do not fit within the Grid component, a scrollbar is provided.

The Grid component uses the database column names as the Grid column headings. You can narrow the queried columns by adjusting the SELECT clause in the associated RelationView component's Select Clause property.

Notice the placement of the Grid buttons.



	Registration_ID	Package_ID	Customer_First_Nam	Cu
0	6	780	Carl	Thr
1	0			
2	7	780	Cheryl	Pot
3	8	780	fjdskfdkls	fjkd
4	9	782	aaaaaa	bbb
5	10	782		
6	11	784		
7	12	782		
8	13	782	Bradley	Ker
9	14	788		
10	15	788		

Grid Properties

Background
Binding (dbAWARE only)
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Number Of Columns
Number Of Rows
Visible

RecordStateLabel component (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`Record_Label_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.awt.RecordStateLabel.html-_top_')}

Description

At runtime, this component displays the state of the current record.

This component is available in Visual Cafe Pro only.

Use this component to provide record status information on your form.

Possible record states are:

- Existing
- Modified
- New
- Marked for Deletion
- Deleted
- New Modified
- Invalid

Properties

Although the Binding property is not called directly, the RelView Name subproperty is used.

RelView Name is the name of the RelationView component that this component is bound to.

Note   The Text property has no effect in this component.

RecordNumberLabel (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`Record_Label_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.awt.RecordNumberLabel.html-_top_')}

Description

At runtime, displays the current record's number in the retrieved record set. This may be a useful addition to your form.

This component is available in Visual Cafe Pro only.

Properties

Although the Binding property is not called directly, the RelView Name subproperty is used.

RelView Name is the name of the RelationView component that this component is bound to.

Note   The Text property has no effect in this component.

Record Label Properties

Alignment
Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
RelView Name
Text
Visible

RadioBox component (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`RadioBox_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button API,JI(`APIRef.hlp',`symantec.itools.db.awt.RadioBox.html-_top_')}

Description

A dbAWARE component with no visual properties that acts like a container to group radio button components. The RadioBox component allows you to bind RadioButton components to a database column.

This component is available in Visual Cafe Pro only.

Four of the binding subproperties are present: RelView Name, Projection Name, Dynamic Update, and Treat Blank As.

RelView Name is the name of the RelationView component this component is bound to.

Projection Name is the name of the database column to which this component is bound.

Note You can also use the column number instead of the column name. The column number is the one-relative index of the projection in the RelationView.

Dynamic Update defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Treat Blank As defines how a component with no value (empty) is saved in the database.

Important RadioBox properties need to be defined before any dbAWARE RadioButton properties are defined.

Important Only use the dbAWARE version of the RadioButton component in a RadioBox.

Properties

This component has three properties: Binding property (dbAWARE only), Class property, and Name property.

RadioBox Properties

Binding (dbAWARE only)
Class
Name

RelationView component (dbAWARE only)

**{button Properties,PI(`vcafe.hlp',`RelationView_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.pro.RelationView.html-_top_')}**

Description

An invisible component that defines a view of a database table and defines how the data is retrieved from the dbANYWHERE server in   a query against the database.

This component is available in Visual Cafe Pro only.

When a RelationView is defined for a form, you see the following icon on the form at design time:

Note   The RelationView is must be added *after* adding the Session and ConnectionInfo component. The easiest way to add these components is by using the Project Wizard and Add Table Wizard.

- Access the Project wizard by selecting the dbAWARE Project Wizard from the File ▶ New Project dialog.
- Access the Add Table Wizard by selecting Insert ▶ Add Table Wizard or selecting the component from the Component Palette or Library.

A project can contain multiple RelationViews. These views can have a joined relationship, or they can be independent. In the case of a join, the first RelationView in a project is used as a master view. When you add a second RealtionView component, you create a detail view, thereby forming a master/detail relationship . The “join” between the two tables is created by defining a Parent RelationView and other associated Join properties.

Use the Project Wizard for your first RelationView (the master view) and then Add Table Wizard for additional views.

When a master/detail relationship is formed, the relationship is considered to be within the same transaction. When the master view is saved, the detail view is also saved, and the entire transaction gets committed. Similarly, when the detail view is saved, the master view is also saved.

For details on creating a RelationView for a Grid component, see Grid component .

Using RelationView and MultiView classes

MultiView   is a container class for the dbANYWHERE database transaction object. A MultiView contains a transaction's RelationViews. To interact with a dbANYWHERE Server you do not have to interface directly with a MultiView object, the RelationView class can encapsulate a MultiView object. For example, RelationView.saveMultiView( ) translates to MultiView.save( ) for the RelationView's parent MultiView.   However, a   MultiView object can be useful to global transactions that have a handle to a MultiView object. For more information on MultiView, see the dbANYWHERE API online HTML reference.

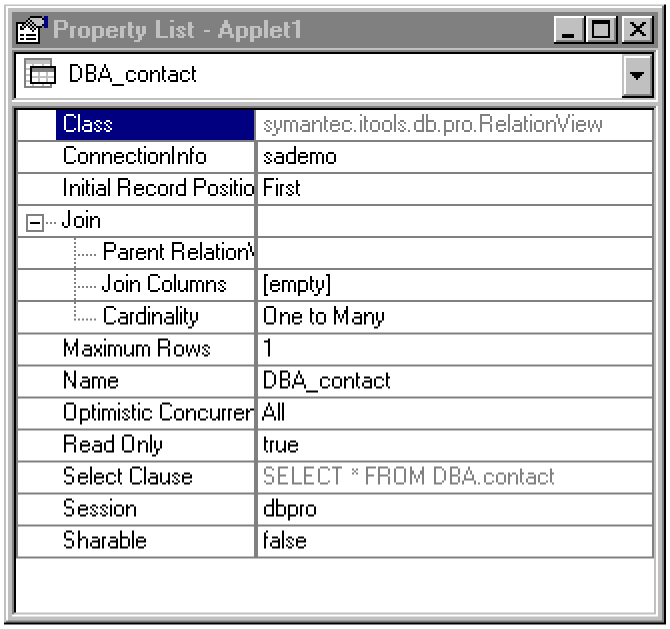
See   For more information on RelationView, MultiView, and AutoDetail classes, see the dbANYWHERE API online HTML manual.

Properties

Note   Do not change RelationView component properties at runtime.

A RelationView's SQL statement (Select Clause property) defines the view. You can create a view that is a detail view or a view that is not a detail view.

This screen shot shows the Property List for a RelationView component that was automatically created when a table name from the dbNAVIGATOR window was dragged in to the Form Designer.



Property List - Applet1

DBA_contact

Class	symantec.itools.db.pro.RelationView
ConnectionInfo	sademo
Initial Record Positio	First
Join	
Parent Relation\	
Join Columns	[empty]
Cardinality	One to Many
Maximum Rows	1
Name	DBA_contact
Optimistic Concurrer	All
Read Only	true
Select Clause	SELECT * FROM DBA.contact
Session	dbpro
Sharable	false

RelationView Properties

Class
ConnectionInfo
Initial Record Position
Join
Name
Optimistic Concurrency
Select Clause
Session

Session component (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`Session_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.db.pro.Session.html-_top_')}

Description

An invisible component that defines the connection to a dbANYWHERE server, via a TCP network connection, and provides access to the dbANYWHERE classes. At runtime, the Session component generates code that creates a session service for the dbANYWHERE server. A session service detects and discards duplicate information packets in a two-way exchange of information. Each session can only service one connection or uniform resource locator (URL).

This component is available in Visual Cafe Pro only.

Add a Session component prior to adding the RelationView component. Use the name of this component as the value for the RelationView component's Session property. The easiest way to add these components is by using the Project Wizard and Add Table Wizard.

- Access the Project wizard by selecting the dbAWARE Project Wizard from the File ▶ New Project dialog.
- Access the Add Table Wizard by selecting Insert ▶ Add Table Wizard or selecting the component from the Component Palette or Library.

Note   It is recommended that all connections to a single dbANYWHERE server utilize the same Session component. This helps to reduce the number of server connections.

Note   A session can only represent one URL, so you must   add multiple Session components to show data from different dbANYWHERE servers.

See: For more information the Session class, see the dbANYWHERE API online HTML manual.

Properties

Note   Do not change Session component properties at runtime.

The URL property defines the dbANYWHERE server's URL (location and port number), for example, dbaw://123.45.1.10:8889. The URL is a combination of the access protocol ("dbaw:"), the host name or IP address ("//123.45.1.10"), and the port number that the dbANYWHERE server is listening to ("8889").

It is common to use the server name "//locahost" while you are developing a project. This indicates that the same machine is running Visual Cafe PRO and the dbANYWHERE server. The port number :8889 usually works for most environments, and is the default.

The Name property defines the name of the dbANYWHERE server that the project expects to access.

Session Properties

Class
Name
URL

TimeMask (dbAWARE only)

{button Properties,PI(`vcafe.hlp',`FormattedTextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.db.awt.TimeMask.html-_top_')}

Description

Extends FormattedTextField.

Creates a box in which your user can type text that is limited to a time format. 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.

This component is available in Visual Cafe Pro only.

To post Date events or text to other components, use the Interaction Wizard.

Properties

The Mask property for the Time component is predefined and cannot be modified. If you want to use a custom mask, use a dbAWARE FormattedTextField component and define its Mask property.

Container components

RadioButtonGroupPanel component

{button Properties,PI(`vcafe.hlp',`RadioButtonGroupPanel_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_RadioGroupPanel')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.RadioButtonGroupPanel.html-_top_')}

Description

Extends Panel.

Creates an rectangular panel area that automatically groups the radio buttons that it contains.

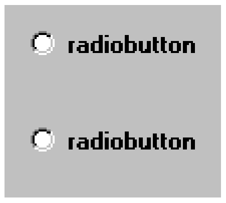
Add radio buttons to the panel by dragging them from the Component Palette onto the panel.

> 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.



RadioGroupBox

RadioButtonGroupPanel Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Layout
Name
Visible

Standard Components

No Help Available

Help is not currently available for this component.

AboutDialog component

{button Properties,PI(`vcafe.hlp',`Common_Dialog_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_AboutDialog')}

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.

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.

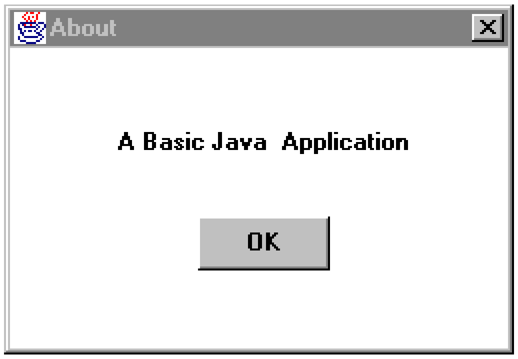
  When you create a new project with the Basic Application template, a project is created automatically for you that contains a Frame, an AboutDialog, and QuitDialog (a custom built Dialog component). The frame contains a menu bar that has menu items preconnected to the dialog boxes.
- The easiest way 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

Animator component

{button Properties,PI(`vcafe.hlp',`Animator_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Animator')} {button API,JI(`APIRef.hlp',`symantec.itools.multimedia.Animator.html-_top_')}

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

Applet component

{button Properties,PI(`vcafe.hlp',`Applet_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.applet.Applet.html-_top_')}

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>
```

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

AttentionDialog component

{button Properties,PI(`vcafe.hlp',`Common_Dialog_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button Example,JI(`vcafe.hlp',`Example_AttentionDialog')}

Description

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.

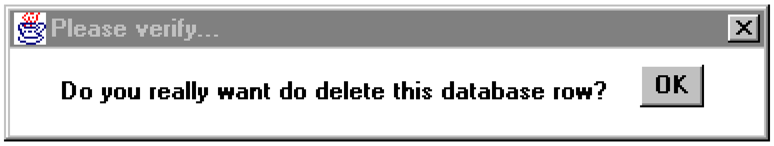
Properties

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.


Please verify...

Do you really want do delete this database row?

BorderPanel component

{button Properties,PI(`vcafe.hlp',`BorderPanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_BorderPanel')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.BorderPanel.html-_top_')}

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

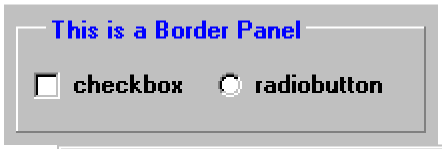
BorderPanel provides layout managers, which you can choose by using the Layout property.

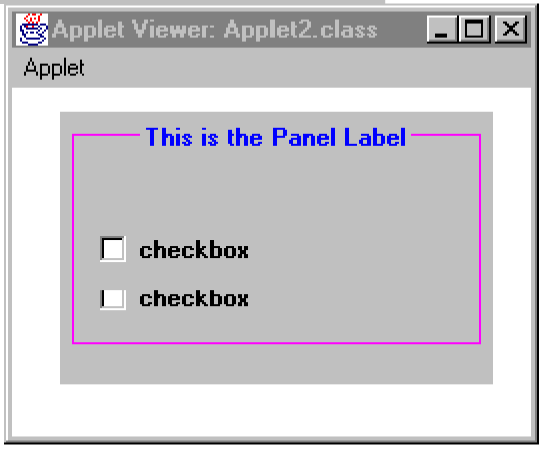
Use the Border Color property to set the color of the panel's border.

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

Button component

{button Properties,PI(`vcafe.hlp',`Button_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_Button')} {button API,JI(`APIRef.hlp',`java.awt.Button.html-_top_')}

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.

  Under MS Windows, a button appears outlined when it possesses input focus. This indicates that it can receive a user event. When the user clicks the button, it appears depressed to indicate that it has detected the Click event.

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

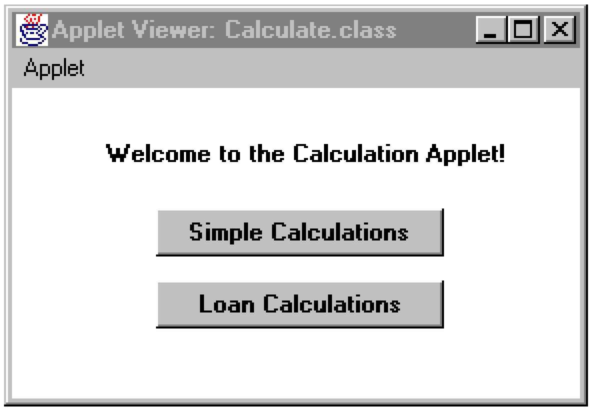
Properties

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.


Applet Viewer: Calculate.class
Applet
Welcome to the Calculation Applet!
Simple Calculations
Loan Calculations

Button Properties

Action Command
Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Label
Name
Visible

Calendar component

{button Properties,PI(`vcafe.hlp',`Calendar_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Calendar')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.Calendar.html-_top_')}

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 numbers (NumericSpinner component) and a list of the months in the year (MonthSpinner component). At runtime, 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

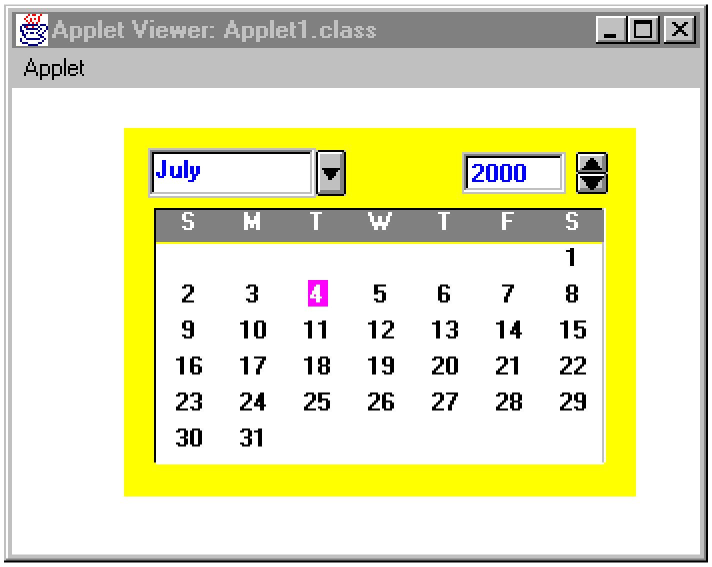
To specify a start date, use the Date property . The string accepts many date syntaxes; including the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT"

To set the color of the text in the month and year text field, use the Foreground property .

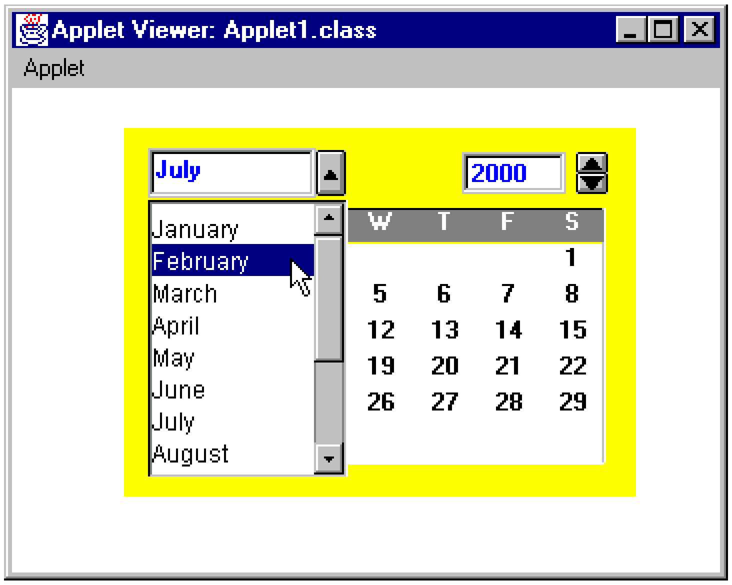
To set the color of the selected/highlighted date on the calendar, use the Selection Color property .

Example

The first screen shot show 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.



S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					


Applet Viewer: Applet1.class
Applet
July
2000
January
February
March
April
May
June
July
August
W T F S
1
5 6 7 8
12 13 14 15
19 20 21 22
26 27 28 29

Calendar Properties

Background
Bounds
Class
Cursor
Date
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Selection Color
Visible

Canvas component

{button Properties,PI(`vcafe.hlp',`Canvas_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.awt.Canvas.html-_top_')}

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:

Animator	Circle	DirectionButton
Ellipse	Firework	HorizontalLine
HorizontalSlider	ImageButton	ImageHTMLLink
ImageViewer	InvisibleButton	InvisibleHTMLLink
Label3D	LabelButton	LabelHTMLLink
MovingAnimation	Nervous Text	Plasma
ProgressBar	Rect	ScrollingText
Square	StateCheckBox	ToolBarSpacer
VerticalLine	VerticalSlider	WrappingLabel

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

CheckboxMenuItem component

{button Properties,PI(`vcafe.hlp',`CheckboxMenuItem_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_CheckboxMenuItem')}   {button API,JI(`APIRef.hlp',`java.awt.CheckboxMenuItem.html-_top_')}

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);
```

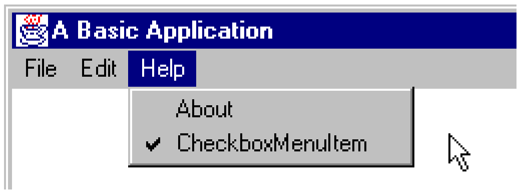
To add a check box menu item to an existing menu, call the menu component's add method:

```
menu1.add( new java.awt.CheckboxMenuItem(label) );
```

Part	Type	Description
menuItem1	CheckboxMenuItem	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

Choice component

{button Properties,PI(`vcafe.hlp',`Choice_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Choice')}   {button API,JI(`APIRef.hlp',`java.awt.Choice.html-_top_')}

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) or RETURN (on Macs) 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

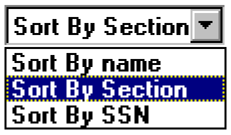
To create a new Choice containing no text labels in project source code, use this syntax:

```
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

Circle component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Circle')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.Circle.html-_top_')}

Description

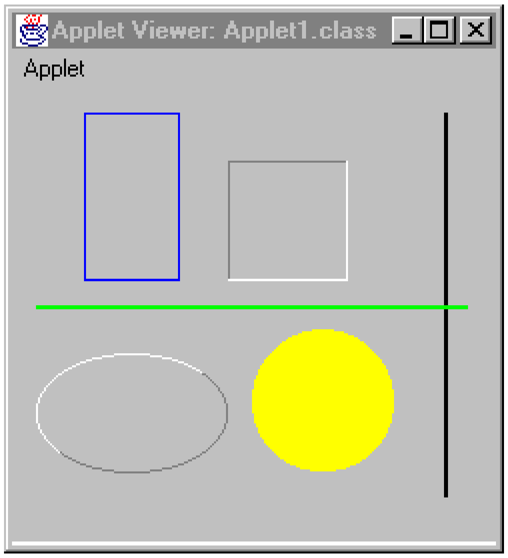
A simple circle. This component is provided for programming convenience, so that you don't have to draw the shape.

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

DaySpinner component

{button Properties,PI(`vcafe.hlp',`Spinner2_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_DaySpinner')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.spinner.DaySpinner.html-_top_')}

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 runtime, only the selected value is displayed in the text box.

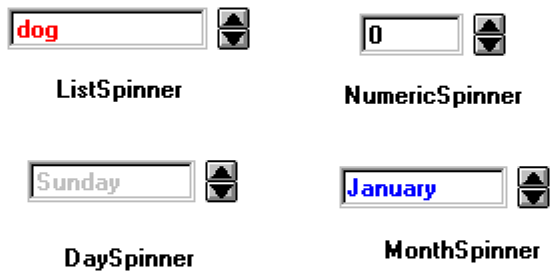
Properties

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 component. Each component has a label component under it. Various colors were chosen for the Foreground to display the colored text.



Dialog component

{button Properties,PI(`vcafe.hlp',`Common_Dialog_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Dialog')} {button API,JI(`APIRef.hlp',`java.awt.Dialog.html-_top_')}

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);
```

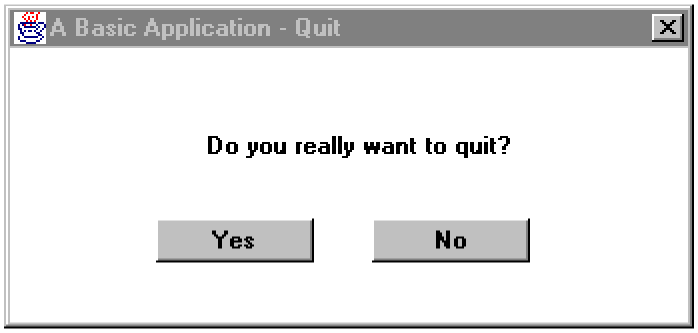
Part	Type	Description
dialog1	Dialog	The variable name used to refer to the component in code.
parent	Frame	The main application window from which the Dialog can be shown.
title	String	The text to display in the dialog box's title bar.
modal	boolean	Whether the dialog is modal or not.

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.


A Basic Application - Quit

Do you really want to quit?

DirectionButton component

{button Properties,PI(`vcafe.hlp',`DirectionButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_DirectionButton')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.DirectionButton.html-_top_')}

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

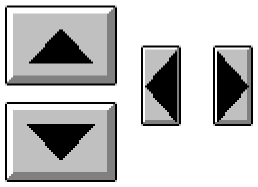
To set the direction of the arrow, use the Direction property.

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
Background
Bevel Height
Border Color
Bounds
Button Color
Class
Cursor
Direction
Enabled
Font
Foreground
Frame Name
HTML Link URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Notify Delay
Notify While Pressed
Show Focus
Show Link URL Status
Use Offset
Visible

Ellipse component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Ellipse')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.Ellipse.html-_top_')}

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.

Emblaze20 component

{button Properties,PI(`vcafe.hlp',`Emblaze20_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Emblaze20')} {button API,JI(`APIRef.hlp',`geo.Emblaze20.html-_top_')}

Description

Creates a box that displays a Emblaze animation file (.BLZ.).

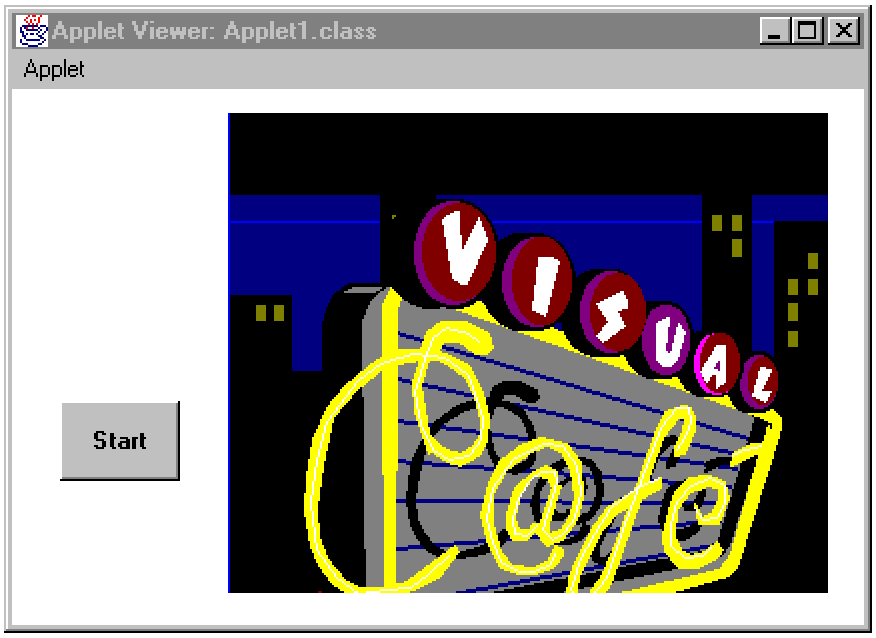
To run the animation, create a button and then use the Interaction Wizard to connect the button to the Emblaze20 component. Select the button trigger action as clicked and the Emblaze action as Start the animation. This connection sends the start method to the Emblaze component when the button is clicked 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

Firework component

{button Properties,PI(`vcafe.hlp',`FileDialog_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_Firework')} {button API,JI(`APIRef.hlp',`symantec.itools.multimedia.Firework.html-_top_')}

Description

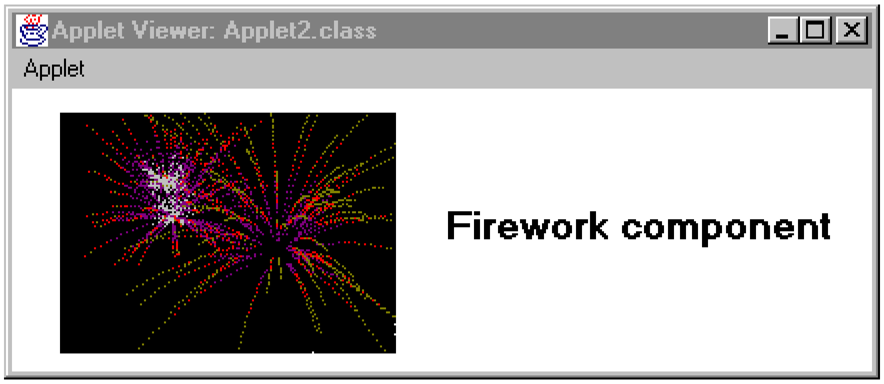
An animated fireworks image. This multimedia component is provided for novelty and programming convenience.

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

▸

Frame component

{button Properties,PI(`vcafe.hlp',`Frame_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.awt.Frame.html-_top_')}

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.

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
Name
Resizable
Title
Visible

HorizontalLine component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_HorizontalLine')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.HorizontalLine.html-_top_')}

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.

Horizontal and Vertical Scrollbar components

{button Properties,PI(`vcafe.hlp',`Scrollbar_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`java.awt.Scrollbar.html-_top_')}

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
Visible
Visible AmountVisible_Amount

HorizontalSlider and VerticalSlider components

{button Properties,PI(`vcafe.hlp',`Slider_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_HorizontalSlider')}   {button API (h),JI(`APIRef.hlp',`symantec.itools.awt.HorizontalSlider.html-_top_')}   {button API (v),JI(`APIRef.hlp',`symantec.itools.awt.VerticalSlider.html-_top_')}

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

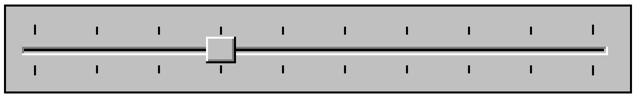
To specify the range of values that the slider indicates, change the Min Value property and the Max Value property.

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

ImageButton component

{button Properties,PI(`vcafe.hlp',`ImageButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_ImageButton')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.ImageButton.html-_top_')}

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

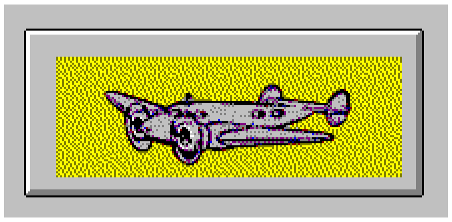
To select and display an image on the button, use the Image URL property. Specify a file name for the JPEG or GIF image you want displayed. Then, if desired, scale the image to fit in the button by using the Image Style property.

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

Background
Bevel Height
Border Color
Bounds
Button Color
Class
Cursor
Enabled
Font
Foreground
Frame Name
HTML Link URL
Image Style
Image URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Notify Delay
Notify While Pressed
Show Focus
Show Link URL Status
Use Offset
Visible

ImageHTMLLink component

{button Properties,PI(`vcafe.hlp',`ImageHTMLLink_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.ImageHTMLLink.html-_top_')}

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.

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_Stryle 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

Background
Bevel Height
Border Color
Bounds
Button Color
Class
Cursor
Enabled
Font
Foreground
Frame Name
HTML Link URL
Image URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Notify Delay
Notify While Pressed
Show Focus
Show Link URL Status
URL
Use Offset
Visible

ImageListBox component

{button Properties,PI(`vcafe.hlp',`ImageListBox_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_ImageListBox')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.ImageListBox.html-_top_')}

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 hilight action of the ImageListBox.   When ComboBox Mode is true list selections will be hilighted anytime the mouse cursor is over them.   When false, the normal ListBox hilight is used.

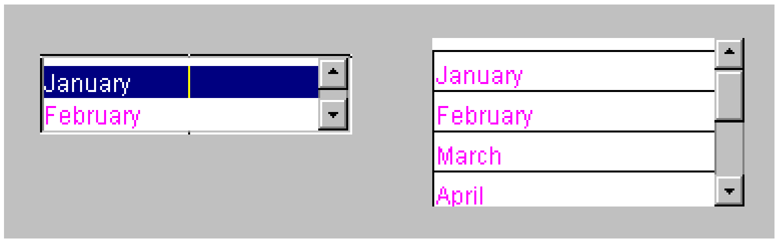
The Enabled property effects the entire ImageListBox component.   Individual list items may be enabled and disabled in program code using the `setEnabled` method.

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).

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

ImagePanel component

{button Properties,PI(`vcafe.hlp',`ImagePanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.ImagePanel.html-_top_')}

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 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
Inherit Background
Inherit Font
Inherit Foreground
Layout
Name
URL
Visible

IntlLongDistPhoneNumber component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.IntlLongDistPhoneNumber.html-_top_')}

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.

InvisibleButton component

{button Properties,PI(`vcafe.hlp',`InvisibleButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_InvisibleButton')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.InvisibleButton.html-_top_')}

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

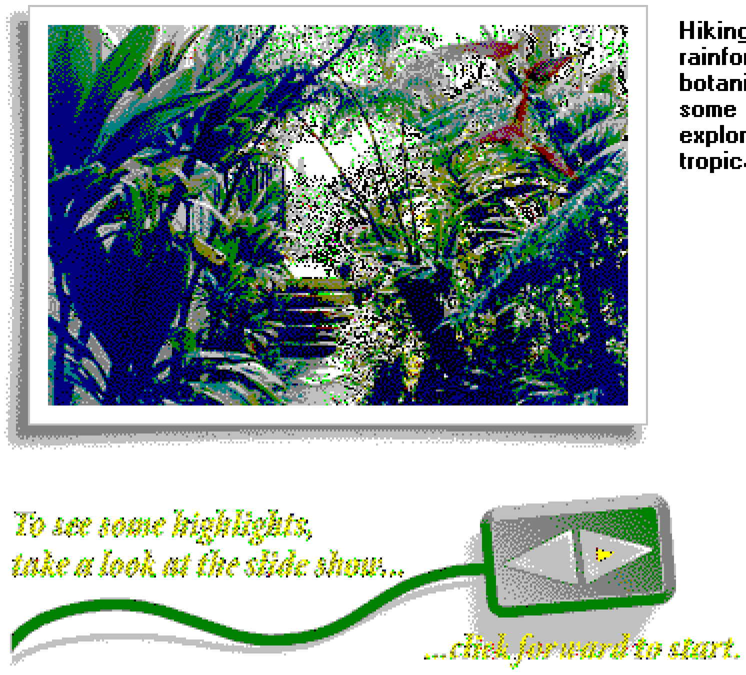
Button tips:

- 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 tour 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.

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

Action Command
Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Visible

InvisibleHTMLLink component

{button Properties,PI(`vcafe.hlp',`InvisibleHTMLLink_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.InvisibleHTMLLink.html-_top_')}

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.

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

Action Command
Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Frame Name
HTML Link URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Visible

KeyPressManagerPanel component

{button Properties,PI(`vcafe.hlp',`KeyPressManagerPanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.KeyPressManagerPanel.html-_top_')}

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

Label3D component

{button Properties,PI(`vcafe.hlp',`Label3D_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Label3D')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.Label3D.html-_top_')}

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.

Note   For most components, labels are usually created by specifying text for the Label property of that component.

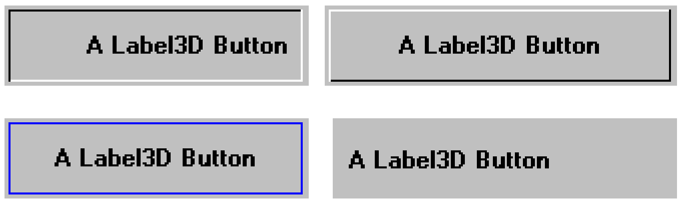
Properties

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

LabelButton component

{button Properties,PI(`vcafe.hlp',`LabelButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_labelButton')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.LabelButton.html-_top_')}

Description

A button that has a 3D appearance and whose text can be aligned left, center, or right. In additon, 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 clicks the button, it appears depressed to indicate that it has detected the click event.

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 .

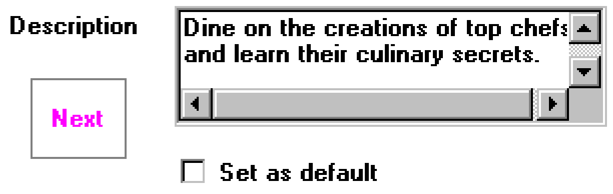
To specify the destination URL, double-click the HTML Link URL property and select the target file.

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 bevel style of BEVEL_LINE and magenta text color.



Dine on the creations of top chefs
and learn their culinary secrets.

LabelButton Properties

Alignment Style
Background
Bevel Height
Border Color
Bounds
Button Color
Class
Cursor
Enabled
Font
Foreground
Frame Name
HTML Link URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Notify Delay
Notify While Pressed
Show Focus
Show Link URL Status
Text
Text Color
Use Offset
Vertical Alignment Style
Visible

LabelHTMLLink component

**{button Properties,PI(`vcafe.hlp',`LabelHTMLLink_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.LabelHTMLLink.html-_top_')}**

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
Background
Bevel Height
Border Color
Bounds
Button Color
Class
Cursor
Enabled
Font
Foreground
Frame Name
HTML Link URL
Inherit Background
Inherit Font
Inherit Foreground
Name
Notify Delay
Notify While Pressed
Show Focus
Show Link URL Status
Text
Text Color
URL
Use Offset
Vertical Alignment Style
Visible

Line component

{button Properties,PI(`vcafe.hlp',`Line_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.Line.html-_top_')}

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

ListSpinner component

{button Properties,PI(`vcafe.hlp',`Spinner_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_ListSpinner')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.spinner.ListSpinner.html-_top_')}

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 runtime, 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

LocalPhoneNumber component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.LocalPhoneNumber.html-_top_')}

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
Caret Position
Class
Columns
Cursor
Edit Font
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Mask
Name
Selection End
Selection Start
Text
Visible

LongZipCode component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.LongZipCode.html-_top_')}

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.

Menu component

{button Properties,PI(`vcafe.hlp',`Menu_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_Menu')}   {button API,JI(`APIRef.hlp',`java.awt.Menu.html-_top_')}

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

There is no event associated with clicking a menu to pull down the menu. This means that component interactions cannot be defined with   Menu components.

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);
```

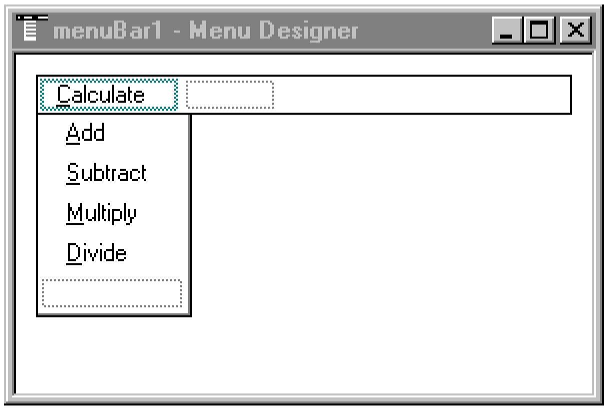
Part	Type	Description
menu1	Menu	The variable name used to reference the component in code.
Label	String	The text that displays in the menu on the menu bar.
tearOff	boolean	A setting that determines whether or not the menu is a tear-off menu.

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

MenuBar component

{button Properties,PI(`vcafe.hlp',`MenuBar_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_MenuBar')}   {button API,JI(`APIRef.hlp',`java.awt.MenuBar.html-_top_')}

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 runtime, you can display a different menu bar by calling the Frame's `setMenuBar` method.

Coding the component

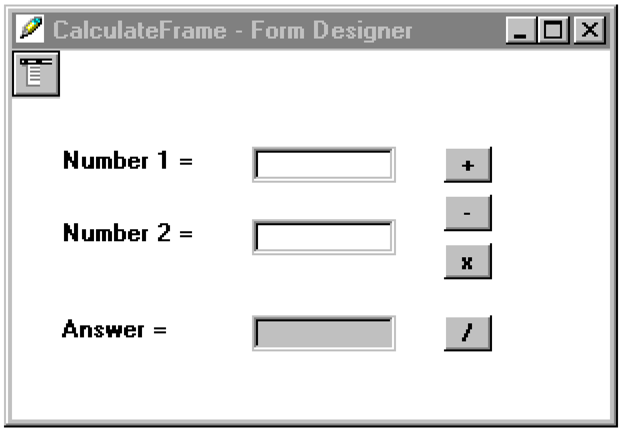
To create a new menu bar in source code, use this syntax:

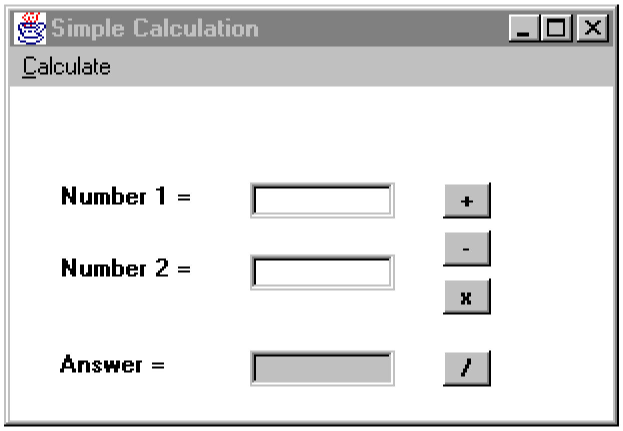
```
mb = new java.awt.MenuBar();
```

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.





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
Main MenuBar
Name

MenuItem component

{button Properties,PI(`vcafe.hlp',`MenuItem_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_MenuItem')} {button API,JI(`APIRef.hlp',`java.awt.MenuItem.html-_top_')}

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

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

he 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 your 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.

MenuItem Properties

Class
Enabled
Label
Menu Shortcut
Name
Separator

MonthSpinner component

{button Properties,PI(`vcafe.hlp',`Spinner2_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_MonthSpinner')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.spinner.MonthSpinner.html-_top_')}

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 runtime, 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 component. Each component has a label component under it. Various colors were chosen for the Foreground property to display the colored text.

Month/DaySpinner Properties

Background
Bounds
Class
Cursor
Editable
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
List Items
Maximum
Minimum
Name
Orientation
Value
Visible
Wrappable

MovingAnimation component

{button Properties,PI(`vcafe.hlp',`MovingAnimation_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_MovingAnimation')} {button API,JI(`APIRef.hlp',`symantec.itools.multimedia.MovingAnimation.html-_top_')}

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.

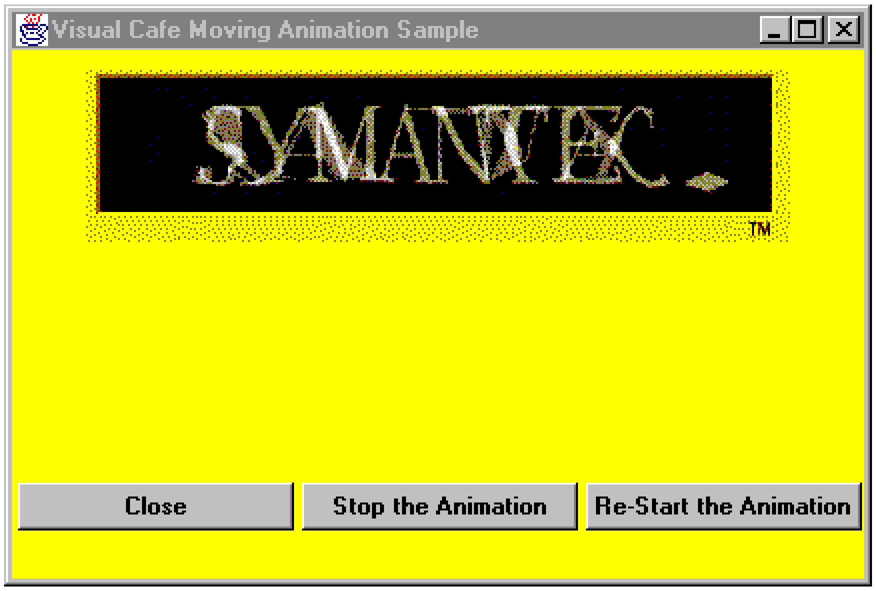
To view your animation at design time, set the Preview Component property to true.

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

MultiList component

{button Properties,PI(`vcafe.hlp',`MultiList_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_MultiList')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.MultiList.html-_top_')}

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.

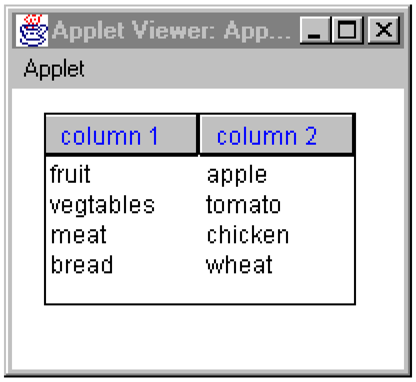
Use the Allow Sorting property to specify whether the list gets sorted when a column heading is clicked.

Use the Multiple Mode property to allow multilple 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
```



column 1	column 2
fruit	apple
vegtables	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

NumericSpinner component

{button Properties,PI(`vcafe.hlp',`NumericSpinner_component')}   {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example2_NumericSpinner')}   {button Example,JI(`vcafe.hlp',`Example_NumericSpinner')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.spinner.NumericSpinner.html-_top_')}

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 runtime, 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 component. 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 component. 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

OpenFileDialog component

{button Properties,PI(`vcafe.hlp',`FileDialog_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button API,JI(`APIRef.hlp',`java.awt.FileDialog.html-_top_')}

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.

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”.

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

Panel component

{button Properties,PI(`vcafe.hlp',`Panel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.awt.Panel.html-_top_')}

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.

Applet	BorderPanel	Calendar
ComboBox	DaySpinner	Grid
ImageListBox	ImagePanel	KeyPressManagerPanel
ListSpinner	MonthSpinner	MultiList
NumericSpinner	RadioButtonGroupPanel	ScrollingPanel
SlideShow	SplitterPanel	StatusBar
TabPanel	ToolBarPanel	TreeView

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

PasswordDialog component

{button Properties,PI(`vcafe.hlp',`Common_Dialog_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_PasswordDialog')}

Description

A dialog that prompts the user for a username 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.

Plasma component

{button Properties,PI(`vcafe.hlp',`Plasma_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Plasma')}   {button API,JI(`APIRef.hlp',`symantec.itools.multimedia.Plasma.html-_top_')}

Description

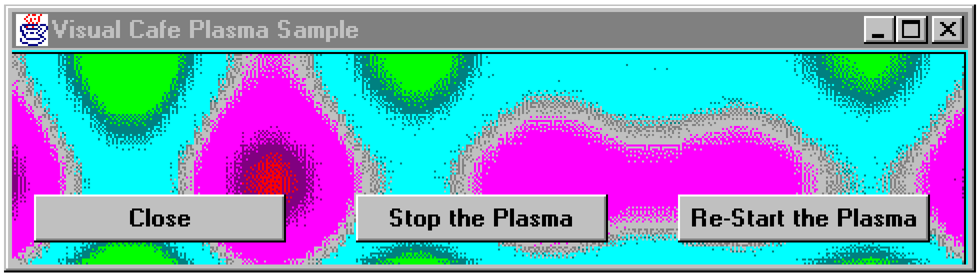
Creates an animation of colored amorphous shapes, where colors gradually modulate as shapes merge and separate. This multimedia component is provided for novelty and programming convenience.

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

PostalCode component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.PostalCode.html-_top_')}

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.

ProgressBar component

{button Properties,PI(`vcafe.hlp',`ProgressBar_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button Example,JI(`vcafe.hlp',`Example_ProgressBar')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.ProgressBar.html-_top_')}

Description

A bar that displays the percentage completed of a particular process.   It it 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 graphicly 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 Bar 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.

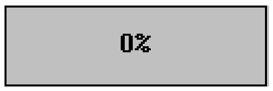
To specify the size of the gap between boxes when the progress bar is drawn as a series of boxes, use the Gap Width property.

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
Bar Color
Bevel Style
Border Indent
Bordered Style Color
Bounds
Box Bar Style
Box Width
Class
Cursor
Enabled
Font
Foreground
Gap Width
Inherit Background
Inherit Font
Inherit Foreground
Name
Show Percentage
Text Color
Value
Visible

ProgressDialog component

{button Properties,PI(`vcafe.hlp',`Common_Dialog_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_ProgressDialog')}

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.

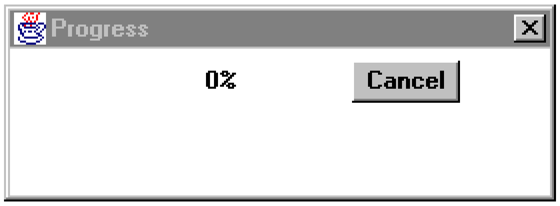
See also ProgressBar component

Properties

To change the dialog window title, use the Title property .

Example

This example is of the default Progress dialog box.



▸

Rect component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Rect')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.Rect.html-_top_')}

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.

▸

RollOverButton component

{button Properties,PI(`vcafe.hlp',`RollOverButton_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.RollOverButton.html-_top_')}

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 URLl
Inherit Background
Inherit Font
Inherit Foreground
Mouse Down Image
Mouse Over Image
Name
Standard Image
Visible

SaveFileDialog component

{button Properties,PI(`vcafe.hlp',`FileDialog_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button API,JI(`APIRef.hlp',`java.awt.FileDialog.html-_top_')}

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.

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”.

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.

ScrollingPanel component

{button Properties,PI(`vcafe.hlp',`ScrollingPanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.ScrollingPanel.html-_top_')}

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

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: Horizontal and Vertical Gap, Minimum Height and Width, Scroll Increment, and separate control over scrollbar visibility using Show Horizontal Scrollbar and Show Vertical Scrollbar.

Use Show Horizontal Scrollbar and Show Vertical Scrollbar properties to choose whether the scroll bars get displayed as needed, or never are displayed.

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.

To adjust the gap between the scrollbars and the contained component (panel), use the Horizontal and Vertical Gap properties.

ScrollingPanel Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Horizontal Gap
Inherit Background
Inherit Font
Inherit Foreground
Layout
Minimum Height
Minumum Width
Name
Scroll Increment
Show Horizontal Scrollbar
Show Vertical Scrollbar
Vertical Gap
Visible

ScrollPane component

{button Properties,PI(`vcafe.hlp',`ScrollPane_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.awt.ScrollPane.html-_top_')}

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

ScrollingText component

**{button Properties,PI(`vcafe.hlp',`ScrollingText_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.multimedia.ScrollingText.html-_top_')}**

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) or RETURN (on Macs) 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) or RETURN (on Macs) 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

SlideShow component

{button Properties,PI(`vcafe.hlp',`SlideShow_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_SlideShow')} {button API,JI(`APIRef.hlp',`symantec.itools.multimedia.SlideShow.html-_top_')}

Description

Displays a series of images in an applet or application. You can provide descriptive text for each image.

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 PRO Tutorial. The SlideShow component displays tour 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.

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.

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.

SlideShow Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
URL List
Visible

SocialInsuranceNumber component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.SocialInsuranceNumber.html-_top_')}

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.

SocialSecurityNumber component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.SocialSecurityNumber.html-_top_')}

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.

SoundPlayer component

{button Properties,PI(`vcafe.hlp',`SoundPlayer_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.multimedia.SoundPlayer.html-_top_')}

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.

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 Synchnonized 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
Synchnonized Mode
URL List

SplitterPanel component

{button Properties,PI(`vcafe.hlp',`SplitterPanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.SplitterPanel.html-_top_')}

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 MouseDrag 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 Dimiension
Font
Foreground
Gap Color
Inherit Background
Inherit Font
Inherit Foreground
Layout
Name
Propagate Resize
Use 3D Border
Visible

Square component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_Square')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.Square.html-_top_')}

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.

StatusBar component

{button Properties,PI(`vcafe.hlp',`StatusBar_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_StatusBar')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.StatusBar.html-_top_')}

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 Style property .

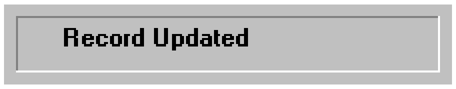
To align a StatusBar to the bottom of a window, set the containing window's Layout property to BorderLayout, and then set the StatusBar's Placement property to South.

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
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
Style
Visible

StatusScroller component

{button Properties,PI(`vcafe.hlp',`StatusScroller_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.StatusScroller.html-_top_')}

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

TabPanel component

{button Properties,PI(`vcafe.hlp',`TabPanel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_TabPanel')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.TabPanel.html-_top_')}

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.

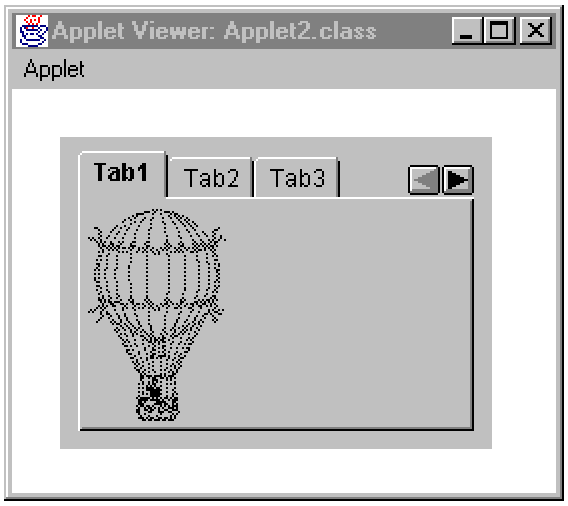
To access any tabbed subpanel in project source code, use the `getTabPanel` method.

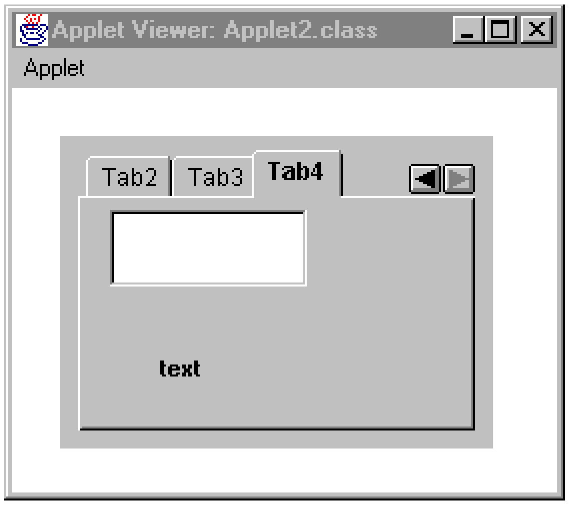
To get the index of the currently active subpanel in project source code, use the `getCurrentPanelNdx` method.

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.




Applet Viewer: Applet2.class
Applet
Tab2
Tab3
Tab4
text

TabPanel Properties

Active Tab
Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Tab Labels
Tabs On Bottom
Visible

Timer component

{button Properties,PI(`vcafe.hlp',`Timer_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button API,JI(`APIRef.hlp',`symantec.itools.util.Timer.html-_top_')}

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)

ToolBarPanel component

{button Properties,PI(`vcafe.hlp',`ToolBarPanel_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button Example,JI(`vcafe.hlp',`Example_TooBarPanel')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.ToolBarPanel.html-_top_')}

Description

This component creates a panel to which you can add buttons to create a toolbar in a window. This toolbar can allows 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.

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 Style property .

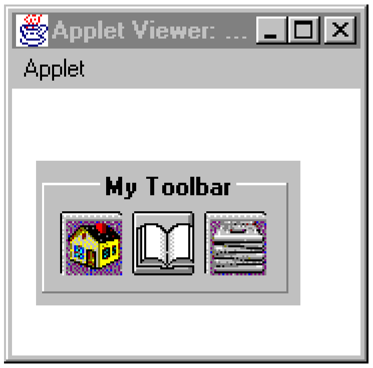
To title the toolbar, use the Label property, the Label Alignment property and the Label Color property.

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 ToolBarSpacer components and three ImageButton components



ToolBarPanel 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
Orientation
Padding Bottom
Padding Left
Padding Right
Padding Top
Style
Visible

ToolBarSpacer component

{button Properties,PI(`vcafe.hlp',`ToolBarSpacer_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_ToolBarSpacer')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.ToolBarSpacer.html-_top_')}

Description

Use ToolBarSpacer to add a gap between elements in a ToolBarPanel component.

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 ToolBarSpacer components and three ImageButton components

ToolBarSpacer Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Space
Visible

TreeView component

{button Properties,PI(`vcafe.hlp',`TreeView_Properties')} {button Events,PI(`vcafe.hlp',`Events_summary')} {button Example,JI(`vcafe.hlp',`Example_TreeView')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.TreeView.html-_top_')}

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.

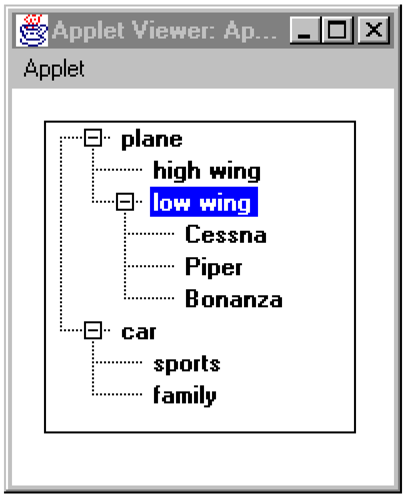
See Creating a TreeView

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) or RETURN (on Macs) 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

Tstamp component

{button Properties,PI(`vcafe.hlp',`Tstamp_Properties')}   {button Events,PI(`vcafe.hlp',`Events_summary')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.Tstamp.html-_top_')}

Description

A component that creates a box in which a date or timestamp can be read or entered using any common date or time format.

Use Tstamp to:

- Edit a date or a timestamp.
- Display a date or timestamp in a pre-defined display format.

To add a Tstamp component to your project, drag the Tstamp icon from the Palette to your Form Designer window. This component can be resized, but does not have horizontal or vertical scrollbars.

Properties

Tstamp controls how the data is entered using the property Format on Entry.

Tstamp controls how the data is displayed using the property Format on Display.

You can choose among Date, Time and Timestamp. This selection will set the following   default formats:

• Date:	Friday October 12th, 1973
• Time:	10:34:52 PM
• Timestamp:	Friday October 12th,1973 10:34:54,000 AM

Tstamp Properties

Background
Binding (dbAWARE)
Bounds
Class
Cursor
Enabled
Font
Foreground
Format on Display
Format on Entry
Inherit Background
Inherit Font
Inherit Foreground
Name
Text
Visible

USLongDistPhoneNumber component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.USLongDistPhoneNumber.html-_top_')}

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.

VerticalLine component

{button Properties,PI(`vcafe.hlp',`Shape_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button Example,JI(`vcafe.hlp',`Example_VerticalLine')}   {button API,JI(`APIRef.hlp',`symantec.itools.awt.shape.VerticalLine.html-_top_')}

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.

Window component

{button Properties,PI(`vcafe.hlp',`Window_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}   {button API,JI(`APIRef.hlp',`java.awt.Window.html-_top_')}

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 runtime.

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
window1	Window	The variable name used to reference the component in code.
parent	Frame	The Frame that is the parent of the Window. A Window requires a parent Frame in order to be displayed.

Window Properties

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Layout
Name
Visible

WrappingLabel component

{button Properties,PI(`vcafe.hlp',`WrappingLabel_Properties')}   {button Events,PI(`vcafe.hlp',`Events_Summary')}
{button API,JI(`APIRef.hlp',`symantec.itools.awt.WrappingLabel.html-_top_')}

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 Text Alignment property.

Use the Property List window to specify a font, color, and point size.

Example

See the SlideShow component example.

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

Background
Bounds
Class
Cursor
Enabled
Font
Foreground
Inherit Background
Inherit Font
Inherit Foreground
Name
Text
Text Alignment
Visible

ZipCode component

{button Properties,PI(`vcafe.hlp',`Predefined_TextField_Properties')} {button Events,PI(`vcafe.hlp',`Events_Summary')} {button API,JI(`APIRef.hlp',`symantec.itools.awt.util.edit.ZipCode.html-_top_')}

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.

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

Understanding the Java AWT

{button How To,AL(`Property_Inspector_using_F1;editing_menus_howto;arranging_components_on_a_form_howto;adding_a_menu_to_a_form_howto',0,`',`')}     {button See Also,AL(`Animation_and_Multithreading;Form_Designer_F1;Creating_Interactions;Nesting_Panels_and_Components;Object_Library_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Understanding_Workspaces',0,`', `')}

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.

Adding a dialog

{button Concepts,AL(`Understanding_Containers',0,`',`')}     {button See Also,AL(`Forms_adding_objects;Form_Designer_F1',0,`',`')}

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.

Details on this step

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

Details on this step

JavaBeans and Component Library

Using the Component Library

{button How To,AL(`Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Form_Designer_F1;binding_code_to_a_form_component_howto;deleting_objects_from_the_object_library_howto;Adding_Bean_to_Library_howto;adding_objects_to_the_palette_howto;deleting_objects_from_the_palette;Palette_F1',0,`',`')} {button See Also,AL(`Projects_overview;Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment;Overview_of_Visual_Caf',0,`',`')}

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 Insert ▶ Component into 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.

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 Window ▶ Component Library.

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

▸

Using JAR files in Visual Cafe

{button How To,AL(`Adding_Bean_to_Library_howto',0,`',`')}     {button See Also,AL(`Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Creating_beans_overview',0,`',`')}

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.

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 ▸ JAR. See Creating a JAR file for more information. To expand a JAR, see Expanding 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 (,).

▸

Adding a group to the Component Library

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1',0,`',`')} {button See Also,AL(`Adding_Bean_to_Library_howto;Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Creating_beans_overview',0,`',`')}

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.

Adding a component to the Component Library

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1',0,`',`')} {button See Also,AL(`Adding_Bean_to_Library_howto;Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Creating_beans_overview',0,`',`')}

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.

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.

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.

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.

Adding a bean to the Component Library

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1',0,`',`')} {button See Also,AL(`Adding_Bean_to_Library_howto;Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Creating_beans_overview',0,`',`')}

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.

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.

   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 Insert ▶ Component into 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 Window ▶ Component Library.)

Note

- If a bean is not added to the Component Library, it most likely does not comply with the JavaBeans standard.
- If you change or move a **class** or **jar** file after it is in the Component Library, the change is not recognized until you restart Visual Cafe, or 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.

Creating JavaBeans components in Visual Cafe

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')} {button See Also,AL(`Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Adding_Bean_to_Library_howto',0,`',`')}

You can create JavaBeans componentswithin the Visual Cafe environment. Visual Cafe provides tools to make your job easier. You should be familiar with the JavaBeans standard before you create custom JavaBeans components.

Tip You can create a component template by dragging a component from the Project window (Objects view) to the Component Library. The source file is copied by Visual Cafe, so you 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.

1. Create a new project for developing one or more JavaBeans components.

   You can start with the Basic JavaBean project template to get started quickly.

   ▶ Details on this step

2. If you are converting a custom component that has a description file, convert the description file.

   ▶ 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.

   ▶ Details on this step

5. To package the bean(s) in a JAR file, choose Project ▶ JAR.

   ▶ Details on this step

   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.

   ▶ Details on this step

Creating a JAR file

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')} {button See Also,AL(`Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Adding_Bean_to_Library_howto',0,`',`')}

Visual Cafe provides a tool you can use within its environment to quickly create JAR files.

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.

1. While the project you want to work with is active, choose Project ▸ JAR.
2. In the JAR name field, type the name and full path that you want the JAR file to have. Click … to browse.

   Note You cannot add to an existing JAR file, but you can create a new JAR file.
3. Click More to display the files and set options for them.

   The JAR utility sets up much of the JAR for you.

   Initially, the display includes all of the classes in your project, any classes they depend on, and other files you have added to the project. Graphics files associated with Visual Cafe components might also be added for you.
   - To add more files, click Add Files and select the files to add.
   - To remove a file, select the file and click Remove.

   Select a file to specify options for it:
   - Select Is Bean if the class is a bean.
   - Select Design Time if it is a class only needed at design time, such as a BeanInfo file.
   - To specify dependencies within the JAR, click Depends and select a file that the class depends on, such as a graphics file or another support file. (It will appear subordinate to the class in the display; this command is the same as the depends flag in the manifest file.) A file must be added to the JAR before you can specify dependencies with it; clicking Depends does not let you add files.
4. Click OK to create the JAR.

Expanding a JAR file

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')}     {button See Also,AL(`Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Adding_Bean_to_Library_howto',0,`',`')}

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
```

Adding Visual Cafe information to a bean

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1',0,`',`')}     {button See Also,AL(`Adding_Bean_to_Library_howto;Add_to_Libray_dialog_F1;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Pallette_howto;Creating_beans_overview',0,`',`')}

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.

SymantecBeanDescriptor

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

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);
```

▶

Converting component description files to JavaBeans

{button Concepts,AL(`Description_File_overview',0,`',`')}     {button See Also,AL(`Example_desc_file;Adding_Bean_to_Library_howto',0,`',`')}

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 ▶ Environment.

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.

Description of the component description file

{button Example,JI(`VCAFE.HLP',`Example_desc_file')}

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:

`%name%`	defines the name of the class/component
`%class%`	full classname of the class/component
`%arg%`	method argument used for `output` 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
```

Example .desc file

{button Concepts,AL(`Description_File_overview',0,`',`')}

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"
```

Moving components in the Component Library

{button Concepts,AL(`Object_Library_F1',0,`',`')}     {button See Also,AL(`Adding_Bean_to_Library_howto',0,`',`')}

You can drag components within the Component Library to another group or location.

Deleting components from the Component Library

{button Concepts,AL(`Object_Library_F1',0,`',`')} {button See Also,AL(`Adding_Bean_to_Library_howto',0,`',`')}

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.

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

Understanding the Component Palette

{button How To,AL(`Customizing_the_Palette_from_the_Palette_Tab_F1;Palette_F1;Forms_adding_objects;Connecting_Form_components_howto;Controlling_Position_and_Visibility_howto',0,`',`')}     {button See Also,AL(`Object_Library_F1;Form_Designer_F1;Creating_a_Project_howto',0,`',`')}

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.

▸

Customizing the Component Palette

{button Concepts,AL(`Adding_Objects_to_the_Palette_howto',0,`',`')}     {button See Also,AL(`Object_Library_F1;Connecting_Form_components_howto;Controlling_Position_and_Visibility_howto;Creating_a_Project_howto;Forms_adding_objects;Customizing_the_Palette_from_the_Palette_Tab_F1',0,`',`')}

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

▸ Component Palette tab.

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

▸ 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.

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

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.

Using the Component Palette tab

{button Concepts,AL(`Adding_Objects_to_the_Palette_howto',0,`',`')}     {button See Also,AL(`Object_Library_F1;Palette_F1',0,`',`')}

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

Using Visual Cafe components

{button How To,AL(`Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}     {button See Also,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}

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.

▸

Using lightweight components

{button How To,AL(`Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}     {button See Also,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}

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

Using the Timer component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`Timer_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the Calendar component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`Calendar_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the TreeView component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`TreeView_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the ProgressBar component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`ProgressBar_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the StatusScroller component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`StatusScroller_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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

Using the SlideShow component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`SlideShow_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

Use the SlideShow component to display a series of images.

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.

   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.

Using the Emblaze20 component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`Emblaze_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

Use the Emblaze20 component to display an Emblaze animation.

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.

Using the ImageViewer component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`ImageViewer_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

Use the ImageViewer component to display a graphics file, such as **jpg** or **gif** files.

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.

Using the SoundPlayer component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`SoundPlayer_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the ScrollingText component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`ScrollingText_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the Plasma component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`Plasma_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the Firework component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`Firework_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the NervousText component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`NervousText_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the Animator component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`Animator_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

Use the Animator component to display a series of graphics files, such as **jpg** or **gif** files, to create an animation.

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.

   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.

Using the MovingAnimation component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`MovingAnimation_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

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.

   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

Using the InvisibleHTMLLink component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`InvisibleHTMLLink_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the MultiList component

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}    {button See Also,AL(`MultiList_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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

Using the TabPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}    {button See Also,AL(`TabPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

   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.

Using the ToolBarPanel and ToolBarSpacer components

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers;Arranging_Objects_with_Layout_Managers',0,`',`')}     {button See Also,AL(`ToolBarPanel_component;ToolBarSpacer_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1;Arranging_Components_on_a_Form_howto',0,`',`')}

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.

Using the KeyPressManagerPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers;Arranging_Objects_with_Layout_Managers',0,`',`')} {button See Also,AL(`KeyPressManagerPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1;Arranging_Components_on_a_Form_howto',0,`',`')}

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.

Using the BorderPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers;Arranging_Objects_with_Layout_Managers',0,`',`')} {button See Also,AL(`BorderPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1;Arranging_Components_on_a_Form_howto',0,`',`')}

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.

Using the ScrollingPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}   {button See Also,AL(`ScrollingPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the RadioButtonGroupPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')}     {button See Also,AL(`RadioButtonGroupPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

Using the ImagePanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`ImagePanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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.

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.

Using the SplitterPanel container

{button Concepts,AL(`GUI_development_with_Visual_Caf;Working_with_Basic_User_Interface_Components;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Understanding_Component_Connections_overview;Visual_Cafe_components_and_containers',0,`',`')} {button See Also,AL(`SplitterPanel_component;Designing_and_Building_GUIs_with_Visual_Caf;Form_Designer_F1;Object_Library_F1;Palette_F1;Forms_adding_objects;Creating_a_Form_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

A SplitterPanel container can be divided into a 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

Understanding custom component integration

{button How To,AL(`Library_adding_objects_howto',0,`',`')}     {button See Also,AL(`Description_File_overview;Example_desc_file',0,`',`')}

<<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.

Adding custom components to the Component Library

{button Concepts,AL(`Object_Library_F1;Example_desc_file',0,`',`')} {button See Also,AL(`Adding_Files_to_a_Project_howto;Deleting_Objects_from_the_Object_Library_howto',0,`',`')}

<<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.

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**).

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 Window ▸ Component Library.
7. Choose Insert ▸ Component into 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.

▸

Adding Palette objects from the Custom Palette dialog box

{button Concepts,AL(`Palette_F1',0,`',`')}     {button See Also,AL(`Adding_Palette_Objects_from_ProjWindow_howto;Adding_Palette_Objects_with_Insert_howto',0,`',`')}

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.

Adding Palette objects from the Component Library

{button Concepts,AL(`Palette_F1',0,`',`')} {button See Also,AL(`Customizing_the_Palette_from_the_Palette_Tab_F1;Adding_Palette_Objects_from_ProjWindow_howto;Adding_Palette_Objects_from_the_Palette_howto;Adding_Palette_Objects_with_Insert_howto',0,`',`')}

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.

Adding Palette objects from the Project Window

{button Concepts,AL(`Palette_F1',0,`',`')}     {button See Also,AL(`Customizing_the_Palette_from_the_Palette_Tab_F1;Adding_Palette_Objects_from_ProjWindow_howto;Adding_Palette_Objects_from_the_Library_howto;Adding_Palette_Objects_with_Insert_howto',0,`',`')}

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.

▸

Creating a Component Palette tab

{button See Also,AL(`Adding_Objects_to_the_Palette_howto',0,`',`')}

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.

▸

Deleting objects from the Component Palette

{button See Also,AL(`Adding_Objects_to_the_Palette_howto;Library_adding_objects_howto;Forms_adding_objects',0,`',`')}

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
▸ Palette tab.
2. Select the object to delete.
3. Click Remove or press the DELETE key.

▸

Moving Palette objects

{button Concepts,AL(`Palette_F1',0,`',`')}     {button See Also,AL(`Adding_Objects_to_the_Palette_howto',0,`',`')}

Palette components can be moved within their group or to another group.

1. Choose File ▸ Environment Options

▸ 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.

Building a custom Component Palette

{button How To,AL(`Adding_Objects_to_the_Palette_howto;Deleting_Objects_form_the_Palette_howto',0,`',`')}

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.

Debugging

Understanding the Visual Cafe debugger

{button How To,AL(`conditional_breakpoint_dialog_f1;toggling a breakpoint;setting a conditional breakpoint;enabling or disabling a breakpoint;setting a breakpoint on a line number',0,`',`')} {button See Also,AL(`Debugging_applets_and_Applications;Understanding_Workspaces;Working_with_Exceptions_in_Java_and_Visual_Caf',0,`',`')}

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

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. hile 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 recompiled, you can restart the debugger.

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.

Debugging applets and applications

{button How To,AL(`stopping a program;pausing a program;running your program;correcting your source code;starting remote applet application debugging;ending remote applet application debugging;stepping into a method',0,`',`')} {button See Also,AL(`Debugging_applets_and_Applications;Understanding_the_Visual_Caf_Debugger;Understanding_Workspaces;Working_with_Exceptions_in_Java_and_Visual_Caf',0,`',`')}

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.

▶

Working with exceptions in Java and Visual Cafe

{button How To,AL(`stopping a program;pausing a program;running your program;correcting your source code;starting remote applet application debugging;ending remote applet application debugging;stepping into a method',0,`',`')}     {button See Also,AL(`debugging_applets_and_applications;understanding_the_visual_cafe_debugger;Understanding_the_Visual_Caf_Debugger',0,`',`')}

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 the 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.

Incremental_debugging_overview $

Incremental debugging features
This is hidden for Free 1.1 release

{button How To,AL(`Correcting your Source Code;Debugging Threads;Viewing and Modifying Variables;F1_Call_Stack_Window;Watching Variables and Expressions;Managing Breakpoints',0,`',`')}	{button See Also,AL(`Understanding_the_Visual_Caf_Debugger;The_Debugging_Workspace_and_Tools;Debugging_Applets_and_Applications;Debugging Remotely;Handing Control to your Program',0,`',`')}

Native_debugging_overview $

Native debugging features
This is hidden for Free 1.1 release

{button How To,AL(`Correcting your Source Code;Debugging Threads;Viewing and Modifying Variables;F1_Call_Stack_Window;Watching Variables and Expressions;Managing Breakpoints;DLL_debug_program',0,`',`')}
{button See Also,AL(`Understanding_the_Visual_Caf_Debugger;The_Debugging_Workspace_and_Tools;Debugging_Applets_and_Applications;Debugging Remotely;Handing Control to your Program',0,`',`')}

Running in Debugger

Debugging your program

{button How To,AL(`Correcting your Source Code;Debugging Threads;Viewing and Modifying Variables;F1_Call_Stack_Window;Watching Variables and Expressions;Managing Breakpoints',0,`',`')} {button See Also,AL(`Understanding_the_Visual_Caf_Debugger;The_Debugging_Workspace_and_Tools;Debugging_Applets_and_Applications;Debugging Remotely;Handing Control to your Program',0,`',`')}

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

Handing control to your program

{button How To,AL(`Running to the Cursor Location;Running to First Line;Running to Program End;Running your Program',0,`',`')}     {button See Also,AL(`Defining_exceptions_debugger;Pausing a Program;Restarting a Program;Resuming a Program;Stopping a Program',0,`',`')}

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.

▶

Running your program in the debugger

{button Concepts,AL(`Handing_Control_to_your_Program',0,`',`')}     {button See Also,AL(`Running to the Cursor Location;Running to First Line;Running to Program End',0,`',`')}

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.

▸

Running to the first line

{button Concepts,KL(`Handing Control to your Program',0,`',`')}     {button See Also,AL(`Running to the Cursor Location;Running to Program End;Running your Program',0,`',`')}

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.

Running to the program end

{button Concepts,AL(`Handing_Control_to_your_Program',0,`',`')}     {button See Also,AL(`Running to the Cursor Location;Running to First Line;Running your Program',0,`',`')}

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.

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.

▸

Running to the cursor location

{button Concepts,AL(`Handing_Control_to_your_Program',0,`',`')}     {button See Also,AL(`Running to First Line;Running to Program End;Running your Program',0,`',`')}

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.

Note   If the selected line does not get executed before the end of the program, your program will not pause.

Pausing a program in the debugger

{button See Also,AL(`Stopping a Program;Resuming a Program',0,`',`')}

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.

Resuming a program in the debugger

{button See Also,AL(`Restarting a Program;Stopping a Program;Pausing a Program',0,`',`')}

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.

▶

Restarting a program in the debugger

{button See Also,AL(`Resuming a Program;Stopping a Program',0,`',`')}

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.

▶

Stopping a debug session

{button See Also,AL(`Restarting a Program;Resuming a Program;Running your Program',0,`',`')}

You can completely stop a program (as opposed to pausing it temporarily). Stopping a program terminates it completely you can work further on it. Execution of a stopped program cannot be continued.

- Choose Debug ▶ Stop.

  The Java program also is terminated.

Debugger Windows

▶

Using the Messages window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}    {button Concepts,AL(`Understanding_Error_Messages_in_Java_and_Visual_Caf',0,`',`')}

This window displays all Visual Cafe messages. For example, if an error is encountered during parsing, the error displays in this window.

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 Window ▶ Messages.

Using the Variables window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}

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 Window ▸ 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

Using the Calls window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}

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 Window ▸ 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

Using the Breakpoints window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}

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 Window ▶ 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

▶
Using the Watch window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}

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 Window ▶ Watch.

Tasks

Deleting a Variable or Expression from the Watch Window

Modifying a Variable or Expression in the Watch Window

▸

Using the Threads window

{button Concepts,AL(`Adding_a_Dialog_howto;Debugging Your Program',0,`',`')}

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 chair

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)

To open the Threads window

- Choose Window ▸ 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

▶

Correcting your source code

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Adding_Code_to_a_Java_Source_File_howto',0,`',`')}

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 Window ▶ 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.

Using the Source window during debugging

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Adding_Code_to_a_Java_Source_File_howto',0,`',`')}

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

Setting breakpoints

{button How To,AL(`Setting a Breakpoint on a Line Number;Setting a Breakpoint on a Method Name;Setting a Breakpoint on a Variable or Expression;Setting a Conditional Breakpoint;Setting_Breakpoints',0,`',`')}     {button See Also,AL(`Clearing a Breakpoint;Toggling a Breakpoint;Stepping Code After a Breakpoint',0,`',`')}

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

Managing breakpoints

{button How To,AL(`Clearing a Breakpoint;Editing Breakpoints;Enabling or Disabling a Breakpoint;Setting a Breakpoint;Viewing all Breakpoints;Viewing Source for a Breakpoint',0,`',`')}     {button See Also,AL(`Setting Breakpoints;Stepping Code After a Breakpoint;Ignoring Breakpoints',0,`',`')}

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.

▸

Setting a breakpoint

{button Concepts,AL(`Setting Breakpoints',0,`',`')}   {button See Also,AL(`Clearing a Breakpoint;Setting a Conditional Breakpoint',0,`',`')}

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.

Tip You can also set and remove breakpoints on the current line by pressing the F9 function key.

Setting a breakpoint at a line number

{button See Also,AL(`Clearing a Breakpoint;Setting a Breakpoint on a Method Name;Setting a Breakpoint on a Variable or Expression',0,`',`')}

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.

▸

Setting a breakpoint at a method

{button See Also,AL(`Clearing a Breakpoint;Setting a Breakpoint on a Line Number;Setting a Breakpoint on a Variable or Expression',0,`',`')}

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.

▶

Setting a conditional breakpoint

{button Concepts,AL(`Setting_Breakpoints',0,`',`')}  {button See Also,AL(`Clearing a Breakpoint;Modifying a Conditional Breakpoint',0,`',`')}

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.

   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.

Clearing a breakpoint

{button Concepts,AL(`Managing Breakpoints',0,`',`')} {button See Also,AL(`Enabling or Disabling a Breakpoint;Setting a Breakpoint',0,`',`')}

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.

Enabling or disabling a breakpoint

{button Concepts,AL(`F1_Breakpoints_Window',0,`',`')}

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.

Toggling a breakpoint

{button Concepts,AL(`Setting Breakpoints',0,`',`')} {button See Also,AL(`Clearing a Breakpoint;Setting a Breakpoint',0,`',`')}

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.

Tip You can also use the F9 function key to toggle breakpoints.

Modifying a conditional breakpoint

{button Concepts,AL(`Managing Breakpoints',0,`',`')}     {button See Also,AL(`Setting a Conditional Breakpoint',0,`',`')}

You modify a conditional breakpoint by changing the conditions of the breakpoint in the Breakpoints window.

1. Choose Window ▸ 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.

▶

Ignoring breakpoints

{button Concepts,AL(`Managing Breakpoints',0,`',`')}     {button How To,AL(`Running to Program End;Running to the Cursor Location',0,`',`')}

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.

▶

Viewing source for a breakpoint

{button See Also,AL(`Viewing Source for a Method on the Call Stack;Viewing Source for a Selected Thread',0,`',`')}

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 Window ▶ 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.

Stepping through code when the program is paused

{button How To,AL(`Stepping into a Method;Stepping out of a Method;Stepping over a Method',0,`',`')}

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 an 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.

▶

Stepping into a method

{button Concepts,AL(`Stepping_Through_Code_When_the_Program_is_Paused',0,`',`')}     {button See Also,AL(`Stepping out of a Method;Stepping over a Method',0,`',`')}

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.

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.

Stepping over a method

{button Concepts,AL(`Stepping_Through_Code_When_the_Program_is_Paused',0,`',`')} {button See Also,AL(`Stepping into a Method;Stepping out of a Method',0,`',`')}

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.

Note If the next line to be executed does not contain a method call, Step Over performs a single step of the line.

▶

Stepping out of a method

{button See Also,AL(`Stepping into a Method;Stepping over a Method',0,`',`')}

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

Viewing and modifying variables

{button How To,AL(`Modifying a Variable in the Variables Window;Modifying a Variable or Expression in the Watch Window;Viewing the Value of a Variable',0,`',`')}     {button See Also,AL(`Viewing Type Information for a VariableWindow;Viewing the Active Variables in a Thread;Viewing Variables for a Method on the Call Stack;Watching a Variable or Expression',0,`',`')}

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.

Viewing the value of a variable

{button Concepts,AL(`Viewing and Modifying Variables',0,`',`')}     {button See Also,AL(`Modifying a Variable in the Variables Window',0,`',`')}

You can view a variable by selecting its name from the Variables window.

1. Choose Window ▸ 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.

▸

Viewing type information for a variable

{button Concepts,AL(`Viewing and Modifying Variables',0,`',`')}

You can view a variable’s type from the Variables window.

1. Choose Window ▸ 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.

Watching variables and expressions

{button How To,AL(`Watching_a_Variable_or_Expression',0,`',`')}     {button See Also,AL(`Deleting a Variable or Expression from the Watch Window;Modifying a Variable or Expression in the Watch Window',0,`',`')}

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.

▶
Watching a variable or expression in the Watch window

{button Concepts,KL(`Watching Variables and Expressions',0,`',`')}     {button See Also,AL(`Deleting a Variable or Expression from the Watch Window;Modifying a Variable or Expression in the Watch Window',0,`',`')}

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 Window ▶ 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.

Adding a variable in the Watch window

{button Concepts,AL(`Watching Variables and Expressions',0,`',`')} {button See Also,AL(`Deleting a Variable or Expression from the Watch Window;Watching a Variable or Expression',0,`',`')}

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 Window ▶ 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.

▶

Modifying a variable or expression in the Watch window

{button Concepts,AL(`Watching Variables and Expressions',0,`',`')} {button See Also,AL(`Deleting a Variable or Expression from the Watch Window;Watching a Variable or Expression;Adding_a_variable_in_watch_window_howto',0,`',`')}

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 Window ▶ 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.

▶

Deleting a variable or expression from the Watch window

{button Concepts,KL(`Watching Variables and Expressions',0,`',`')}     {button See Also,AL(`Watching a Variable or Expression;Modifying a Variable or Expression in the Watch Window;Adding_a_variable_in_watch_window_howto',0,`',`')}

When you have finished watching a run-time variable or expression, you can delete it directly from the Watch window.

1. Choose Window ▶ 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.

Modifying a variable in the Variables window

{button Concepts,KL(`Viewing and Modifying Variables',0,`',`')}

You can modify a variable by selecting its name from the Variables window and typing a value into the value column.

1. Choose Window ▸ 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.

Viewing parameters for a method on the call stack

{button Concepts,AL(`F1_Call_Stack_Window',0,`',`')}     {button See Also,AL(`Viewing Variables for a Method on the Call Stack;Viewing Code for a Method on the Call Stack',0,`',`')}

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 Window ▶ 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.

▶

Viewing variables for a method on the call stack

{button Concepts,AL(`F1_Call_Stack_Window',0,`',`')}  {button See Also,AL(`Viewing Parameters for a Method on the Call Stack;Viewing Code for a Method on the Call Stack',0,`',`')}

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 Window ▶ 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.

▸

Viewing source for a method on the call stack

{button Concepts,AL(`F1_Call_Stack_Window',0,`',`')}   {button See Also,AL(`Viewing Parameters for a Method on the Call Stack;Viewing Variables for a Method on the Call Stack',0,`',`')}

Visual Cafe allows you to choose any method entered on the call stack and view the source code for that method.

1. Choose Window ▸ 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

Working with threads

{button How To,AL(`adding_thread_to_a_project_howto;debugging a single thread;viewing the call stack for a thread;resuming a suspended thread',0,`',`')}    {button See Also,AL(`Projects_overview;Running_and_Debugging_Applications;Object_Library_F1;Differences_between_Applets_and_Applications',0,`',`')}

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 Window ▸ Threads.

Debugging threads

{button How To,AL(`Debugging a Single Thread;Suspending a Thread;Viewing the Active Variables in a Thread;Viewing the Call Stack for a Thread;Viewing Source for a Selected Thread',0,`',`')} {button Concepts,AL(`Working_with_Threads',0,`',`')}

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.

▶

Debugging a single thread

{button Concepts,AL(`Debugging Threads;Working_with_Threads',0,`',`')}

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.

1. Choose Window ▶ Threads to display the Thread 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.

▶

Suspending a thread

{button Concepts,AL(`Debugging Threads;Working_with_Threads',0,`',`')}     {button See Also,AL(`Resuming a Suspended Thread;Resuming Other Suspended Threads;Suspending Other Threads',0,`',`')}

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.

1. Choose Window ▶ Threads to display the Thread window.
2. Click the thread you want to suspend.
3. Choose the Thread ▶ Suspend.

   The selected thread is suspended.

Resuming a suspended thread

{button Concepts,AL(`Debugging Threads',0,`',`')}     {button See Also,AL(`Resuming Other Suspended Threads;Suspending a Thread;Suspending Other Threads',0,`',`')}

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.

1. Choose Window ▶ Threads to display the Threads window.
2. Click the thread you want to resume.
3. Choose Threads ▶ Resume.

   The selected thread is resumed.

▶

Suspending other threads

{button Concepts,AL(`Debugging Threads',0,`',`')}     {button See Also,AL(`;Resuming a Suspended Thread;Resuming Other Suspended Threads;Suspending a Thread',0,`',`')}

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.

1. Choose Window ▶ Threads to display the Thread swindow.
2. Click the thread you want to work on.

   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.

▶

Resuming other suspended threads

{button Concepts,AL(`Debugging Threads',0,`',`')}   {button See Also,AL(`;Resuming a Suspended Thread;Suspending a Thread;Suspending Other Threads',0,`',`')}

If you have chosen to suspend any threads in a multithreaded program, you can resume all other threads at any time.

1. Choose Window ▶ Threads to display the Threads window.
2. Click the thread you do not want to resume.

   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.

Viewing source for a selected thread

{button Concepts,AL(`Debugging Threads;Working_with_Threads',0,`',`')} {button See Also,AL(`Viewing the Active Variables in a Thread;Viewing the Call Stack for a Thread',0,`',`')}

If you want to look at the source code for a specific thread, you can do so from the Threads window.

1. Choose Window ▶ 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.

▶

Viewing the active variables in a thread

{button Concepts,AL(`Debugging Threads;Working_with_Threads',0,`',`')}     {button See Also,AL(`Viewing Source for a Selected Thread;Viewing the Call Stack for a Thread',0,`',`')}

You can view the values of variables for any thread currently executing.

1. Choose Window ▶ 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 Window ▶ Variables.

   The Variables window opens displaying the chosen thread’s variables.

Viewing the call stack for a thread

{button Concepts,AL(`Debugging Threads;Working_with_Threads',0,`',`')}     {button See Also,AL(`Viewing Source for a Selected Thread;Viewing the Active Variables in a Thread',0,`',`')}

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.

1. Choose Window ▶ 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 Window ▶ Call Stack.

   The Calls window opens on the thread you selected.

5. Click the Method whose code you want to view.

Remote Debugging

Debugging remotely

{button How To,AL(`Ending Remote Applet/Application Debugging;Starting Remote Applet/Application Debugging',0,`',`')}

Visual Cafe allows you to run a program on one computer and debug it remotely on another. 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.

▶

Starting remote program debugging

{button Concepts,AL(`Projects_overview;Debugging Remotely',0,`',`')} {button See Also,AL(`Ending Remote Applet/Application Debugging;Setting_Environment_Variables_in_scini_howto',0,`',`')}

You can debug an applet or application remotely on another computer. You cannot debug native applications or DLLs remotely.

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 class path set in the environment before running caferemote.exe. Make sure the class path is set correctly in autoexec.bat for Windows 95 or in the Control Panel for Windows NT; caferemote.exe can also read classpath information from the sc.ini file in your Bin directory.

1. 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 and executing caferemote.

   Caferemote displays some version information, and then an agent password and the computer's IP address.

2. Click OK.

   Remote debugging is now enabled.

3. On the computer that is running the debugger, load the project corresponding to the class and HTML files you are debugging on the remote computer.
4. Choose Project ▶ Options.
5. If you are remotely debugging an applet, from the Project tab, specify an HTML file including the full path.
6. From the Debugger tab, choose General from the Category list.
7. Check Enable Remote Debugging and type the Host Address and Agent Password as reported by caferemote:

Field	**Description**
Host Address	The IP address of the remote computer.
Agent Password	The remote computer's password, which is returned by caferemote.exe run on the remote computer.

6. Click OK.

Ending remote program debugging

{button Concepts,AL(`Debugging Remotely',0,`',`')}     {button See Also,AL(`Starting Remote Applet/Application Debugging',0,`',`')}

On the remote computer, select the Caferemote console window and press CONTROL+C. This terminates the debug session currently in process.

To continue, you need to re-execute caferemote to get a new agent password.

▸

Understanding errors in Java and Visual Cafe

{button How To,AL(`clearing a breakpoint;toggling a breakpoint',0,`',`')}     {button See Also,AL(`debugging_applets_and_applications;Understanding_Workspaces;Working_with_Exceptions_in_Java_and_Visual_Caf;understanding_the_visual_cafe_debugger',0,`',`')}

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.

The Debugging views

{button How To,AL(`viewing source for a breakpoint;viewing the value of a variable;viewing type information for a variable;watching a variable or expression;viewing variables for a method on the call stack',0,`',`')}     {button See Also,AL(`Creating_Interactions;Maintaining_and_Porting_Your_Java_Applets_and_Applications;Projects_overview;Running_and_Debugging_Applications;Object_Library_F1;Understanding_Workspaces;Working_with_Exceptions_in_Java_and_Visual_Caf',0,`',`')}

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.

Handling exceptions

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.

▸

Setting a breakpoint at a variable or expression

{button See Also,AL(`Clearing a Breakpoint;Setting a Breakpoint on a Line Number;Setting a Breakpoint on a Method Name',0,`',`')}

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 Pro, 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 robortf.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

Glossary

A

abstract class

access type

action

active window

adapter

anchor component

applet

applet tag

application

B

G

garbage collection

H

hidden component

Hierarchy Editor

I

identifier

inheritance

inner class

instance

interaction

interface

introspection

J

Java API

Java file

Java Archive (JAR) file

Java Virtual Machine

JavaBeans

JavaBeans component

JDK: Java Developer's Kit

join

K

(none)

L

layout

list binding

listener

M

macro

master/detail relationship

member

method

Messages window

modal

modeless

N

nested class

O

ODBC

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

(none)

R

refresh

result set

regular expression syntax

RelationView

run mode

runtime

S

scope

server

Source pane

Source window

SQL

SQL statement

stored procedure

T

thread

Threads window

top-level component

U

(none)

V

ValueTip

variable

Variables window

view

Visual Cafe window

W

Definitions

A

abstract class

A class that 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

Action performed on a database:

- delete: Deletes the current record from the Data Source.
- first: Moves the database cursor to the first record in the database table.
- new: Creates a new record in the database table.
- next: Moves the database cursor to the next record in the database table.
- prev: Moves the database cursor to the previous record in the database table.

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 Pro.)

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 (and Visual Cafe Pro) when you choose File ▶ New Project ▶ Basic Application. This template creates a class that is an extension of the 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 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 Window ▶ 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, Window ▶ 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 Window ▸ Class Browser.

client machine

Machine that communicates with a dbANYWHERE Server machine to access a Data Source.

client software

Software that dbANYWHERE uses to communicate 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 runtime 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 runtime, such as a Timer, or displays in a different way in the Form Designer and at runtime, 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:

Window ▶ 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

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

database template

Forms which are pre-designed. You can choose amongst these by using the dbAWARE Template Wizard.   Each Visual Cafe database template is designed for a different kind of information.

data source

Contains the information for creating a database connection, like the name of the database, its server, and its network location. A data source 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 data source is called a DSN (Data Source Name). For information about the Data Sources dbANYWHERE supports, see the "Installing dbANYWHERE Server" manual.

dbANYWHERE (naming conventions)

dbANYWHERE Server	Middleware database software that supports three-tier architecture for database access
dbANYWHERE Workgroup Server	Version of the dbANYWHERE Server which is shipped with Visual Cafe Database Development Edition that restricts the number of licenses for use to five.
dbANYWHERE Server machine	Hardware on which dbANYWHERE software is running
dbANYWHERE	Both versions of dbANYWHERE

dbAWARE component

Visual Cafe component that has additional properties for binding the component to a database row, column, table, or result set.

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:

Window ▶ 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.

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 runtime system performs garbage collection so you don't have to explicitly free the memory associated with objects and other data.

H

hidden component

Component that is visible at design time and invisible at runtime. For example, menu bars, RelationViews, and Sessions are hidden components. To toggle between displaying and hiding hidden components at design time:

Layout ▸ Invisibles

Hierarchy Editor

Window that displays the class hierarchy for your applet or application. To see the Hierarchy Editor:

Window ▶ Hierarchy Editor

identifier

Name of an item in a Java program.

# Incremental_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 runtime, 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.

J

Java API: Java Application Program Interface

API provided by the JDK. For more information, see the Java Web page (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).

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.

JDK: Java Developer's Kit

Tools for developing Java applets and applications. The JDK is included in Visual Cafe and Visual Cafe Pro. It is also available on the Java Web page (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.

master/detail relationship

A one to may relationship between two database tables. The join property of the detail RelationView defines the column of the master view   which is the common attribute in the one to many relationship between the tables. For instance,   you may 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 a SQL query that returns a list of all the customers in a Customer database table who purchased a Spectacular Widget.

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:

Window ▶ Messages

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

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 is the acronym for Open DataBase Connectivity.

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.

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:

Window ▶ Property List

Property List

Window that displays a component's properties. To see the Property List:

Window ▶ Property List

Q

R

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.

RelationView

dbAWARE component which you use to define and maintain a result set. It is invisible at runtime but contains methods used to handle navigation and data manipulation operations on the data in the result set.

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.

runtime

Time when your applet or application is running. At runtime:

- 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.

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.

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.
- Double-click in the Form Designer.
- 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 pre-compiled.

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 Window ▶ 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.

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 runtime, edit its value in this window.

To see the Variables window, choose Window ▶ Variables.

view

In a development environment

Window that shows a particular type of information.

- Global view: View that is accessible from the Window 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

Virtual database table that exists only in memory. A RelationView represents a view. A RelationView's Select Clause property (which is a SQL statement) defines the view.

- Master view: View that is used as the basis for a detail view.
- Detail view: View that is related to a master view. A detail view's join defines a master/detail relationship.
- A view can be a master view or a detail view or both or neither.

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 Window ▶ 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

Change history

3/17/97 - deleted MaxRows, Sharable, And ReadOnly per Carl T.

8/6/97 - Update/check all properties for 1.1

Properties

{button Components,JI(`vcafe.hlp',`Visual_Cafe_components_and_containers')}

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.

In Visual Café DDE, read-only properties are viewable for server, data source, table, and column items. To see these properties, click on a data item in the dbNAVIGATOR.

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

dbAware Properties

Auto Disconnect property (dbAWARE only)

Use this method to control whether to disconnect from the database when a MultiView is closed.

This property is intended for advanced users who are experiencing slow connections to their databases. This is not a necessary property of ConnectionInfo.

For more information on the MultiView class, see RelationView component and the dbANYWHERE API online HTML reference manual.

Runtime modification

This property can not be changed during runtime.

Settings

Valid values	Description
true	(Default) Any RelationView that was created using the associated ConnectionInfo component is automatically disconnected from the database when the MultiView is closed. This occurs automatically when the request/transaction is finished or no longer used.
false	The database connection is not closed when the MultiView is closed. The connection is closed when the user disconnects from the database, or when the applet terminates.

Auto Expand property (dbAware only)

Defines whether a component will recognize data not predefined in the List Items property.

Runtime modification

This property can not be changed during runtime.

Valid values	Description
true	Data from the database that does not match the items in the predefined list are added to the list.
false	(Default) Data from the database that does not match the items in the predefined list are ignored.

Binding property (dbAWARE only)

Defines how the component is bound to the database column and provides built-in data management functionality. The set of subproperties that display for the Binding property in the Property List depends on the component type.

Runtime modification

These properties should not be changed during runtime.

RelView Name

The name of the RelationView component that the component is bound to.

Projection Name

The name of the database column that the component is bound to.
Note: You can also use the column number instead on the column name. The column number is the position of the projection in the RealtionView.

Dynamic Update

Defines when update data is sent to the MultiView. Values in the MultiView are sent to the database when the RelationView save method is invoked. (For information on MultiView, see the dbANYWHERE API HTML online documentation.)

Valid values	Description
true	Each keystroke is sent to the MultiView for processing.
false	(Default) The value is sent to the MultiView when the component loses focus.

Treat Blank As

Defines how a component with no value (empty) is saved in the database.

Valid values	Description
Null	Save as a null value.
Blank	Save as a blank value or an empty string.
Default	(Default) Save as defined by its original state.

Buttons

On Grid components, the following buttons are displayable.

Button	Description	Default Value
Insert	Add a new record above the current record.	False
Goto	Go to the specified record number.	False
Undo	Undo the last change in the current record.	True
Restart	Restart the application or applet. All changes made since the last save or start are lost.	False
Delete	Delete the selected record.	True
Undelete	Undo the last delete in the session.	True
Save	Save all data in the current session.	False

Comments property (dbAWARE only)

Allows general documenting comments about the database table. This property displays the comments of the database table, if any. This property is Read-Only.

Runtime modification

This property can not be changed during runtime.

Settings

Read-only.

ConnectionInfo property (dbAWARE only)

Specifies the name of the ConnectionInfo component that the RelationView is based on.

Runtime modification

This property can not be changed during runtime.

Also see Lookup property (dbAWARE property)

Data Source Name property (dbAWARE only)

Defines the data source that the ConnectionInfo component is based on.

The data source can be either an ODBC 32 data source, which is defined in the appropriate Windows/NT registry, or a dbANYWHERE native driver data source, which is define in the dbawdsn.ini file.

Runtime modification

This property can not be changed during runtime.

Settings

The name of any data source that is associated with the RelationView's Session component.

False Value property (dbAWARE only)

Defines a value to associated with a component when the current component's column value is Boolean False.

See Checkbox component and True Value .

Join property (dbAWARE only)

Defines a master/detail relationship, which is a the relationship between a detail view (the current RelationView) and its master view (parent).

Runtime modification

These properties should not be changed during runtime.

Parent RelationView

This property is enabled only when the RelationView is a detail view. Specifies the parent RelationView that this child RelationView is based on.

Join Columns

Use this property to define a master/detail relationship. Click the ... button in the right margin to display the Join Definition dialog.

Cardinality

This property is enabled only when the RelationView is a detail view. Specifies the relationship between a record in one database table and a record in another database table. Visual Café DDE uses this setting to determine processing priorities and the order of the committing statement within a transaction(s).

Valid values	Description
One to One	A record in the first database table is related to only one record in the second database table.
One to Many	(Default) A record in the first database table is related to none, one, or more records in the second database table.
Many to One	many records in the first database table are related to one record in the second database table.
Many to Many	Many records in the first database table are related to many records in the second database table.

Initial Record Position

Defines the state of the records when they display.

Valid values	Description
First	(Default) The first record selected is displayed for editing. If there are no records in the table, the RelationView is initialized.
New	A blank record displays for data adding (Add mode)

Lookup property (dbAWARE only)

Defines how a component is populated. There are three types of lists that can be used in component population:

- hard coded
- unjoined related lookup (static values)
- joined related lookup (dynamic values)

Unjoined and joined related lookups also have a hard coded portion of the list.

Note: For List components, the Binding property is used differently than the Lookup property binding. This is similar to TextField components. A list does not have to be bound to a database column.

Runtime modification

These properties should not be changed during runtime.

ConnectionInfo

The name of the ConnectionInfo object that this component is based on. See ConnectionInfo property

Select Clause

Use this property for joined and unjoined related list lookups.

The SQL statement for populating the component. See Select Statement property.

Lookup RelView Name

Use this property for joined and unjoined related list lookups.

The name of the RelationView that the lookup should be performed on. If you are using a joined lookup, the use the name of the lookup RelationView. If you are using an unjoined lookup, then use the name of the RelationView to which the list is bound.

Join Columns

Use this property for joined related list lookups ONLY.

Defines the criteria for joining columns in a lookup for component population. This property's setting automatically updates the Select Clause property by adding a WHERE clause. In the generated WHERE clause, the question mark (?) is a place holder for a value in the master view.

The SQL WHERE clause of the Select property must be properly formatted with join conditions using "?" for parameter value substitution.

A RelationView name must be provide in order to create the joined lookup. The join columns define how the parent lookup RelationView and the SQL SELECT clause are related and joined.

The value that displays in the field identifies the current state of data in the component.

Valid values	Description
[empty]	There is no data currently in the component.
[list]	The component currently has a list of values.

Optimistic Concurrency property (dbAWARE only)

Specifies how to update the database when more than one client is accessing it. Optimistic concurrency allows muti-user access to data while maintaining data integrity and preventing update loses.

Optimistic Concurrency controls the SQL that is sent out in an update operation. It determines the fields that should be used in the WHERE SQL clause generated by Visual Café DDE.

All settings ensure that only one row is changed. The only difference between the settings is how the SQL WHERE clause is generated. If the WHERE clause does not uniquely identify the row and multiple rows are updated, a rollback occurs.

Important If the record key is made up of multiple columns, you must include all of the record key elements on the form to ensure unique key generation.

Settings

Valid values	Description
All	(Default) The SQL WHERE clause includes all fields. All column values at the time that the record is read are compared with the current column values before the new field values are inserted into the columns.
Unique	The SQL WHERE clause includes ONLY unique columns, that best identify a row, that are included on the form and bound to a column. The unique column(s) are determined by checking the database metadata in this order: JDBC calls, getBestRowIdentifier, getPrimaryKeys, getIndexInfo. The value of the bound unique column(s) when the record was read are compared with the current values before the new field values are inserted into the columns.
Unique and modified	Same as Unique, except the SQL WHERE clause includes bound unique columns as well as all modified columns.

Notes

The Unique and Unique/modified settings can only be observed and enforced if the unique columns are projected on the form. If you don't want to display the unique columns, hide the text fields.

You must include enough columns in the SQL SELECT clause to uniquely identify the row to be able to modify data. If there is no way to uniquely identify the row, all non-blob columns are used. If duplicate rows exist, data will not be changed and a rollback occurs.

LongVarChar and LongVarBinary (blob) columns are never included in SQL WHERE clauses regardless of the setting.

Password property (dbAWARE only)

The database password to the data source defined in the Data Source Name property

As the user types in a password string, the characters appear as clear text, unmasked, and visible. The password string also appears in clear text in your source code.

The password is case sensitive, if the database is case sensitive.

Also see, User Name property

Select Clause property (dbAWARE only)

The actual SQL statement that is used to query the database and initialize the RelationView. If the SQL string is not in the standard format, the database dialogs and wizards cannot parse the information; you must edit the string manually.

The standard format is:

SELECT X1, X2, X3, … from Y

where X is a series of database columns separated by commas and Y is the name of the database table.

To specify a custom query, click in the Select Statement field and then click on the ... button at the field right to display the SQL Statement editor. The editor lets you change a RelationView's Select Statement property.

Note: Additional optional SQL clauses can be added to the end of a standard formatted SQL string, for example, ORDER BY, HAVING, LIKE, WHERE, etc.

Use a WHERE clause to provide criteria that each row in the table must meet in order to be returned in the result set.

Use an ORDER BY clause to sort the retrieved records, if needed. By default, records retrieved from a query display in the order that they appear in the table.

Also see Lookup property (dbAWARE property)

Tip: When you create a RelationView with a Visual Café DDE wizard, this SQL statement is generated for you.

Session property (dbAWARE only)

Specifies the name of the Session object that the RelationView is based on.

True Value property (dbAWARE only)

Defines a value to associated with a component when the current component's column value is Boolean True.

See Checkbox component and False Value property (dbAWARE property)

URL property (general/dbAWARE)

General

Specifies a pathname in URL format.

dbAWARE (Visual Café DDE)

In the Session component, this property defines the dbANYWHERE server's URL and port, for example,
dbaw://123.45.1.10:8889

The URL is a combination of the access protocol ("dbaw:"), the host domain name or IP address ("//123.45.1.10"), and the port number that the dbANYWHERE server is listening to (":8889").

It is common to use the server name "//localhost" while you are developing a project. This indicates that the same machine is running Visual Café DDE and the dbANYWHERE server. The port number :8889 usually works for most environments, and is the default.

User Name property (dbAWARE only)

The database user name for the data source defined in the Data Source Name property.

The user name string is case sensitive, if the database is case sensitive.

Note: Do not specify a user name and/or password if you wish the applet user to be forced to login to the database each time the applet is run.

General Properties

Action Command property

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”. The default for the InvisibleButton and InvisibleHTMLLink component is “ButtonPressed”.

Active Tab property

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
0	(Default)
integer	The zero-relative index of the subpanel and tab to show.

Alignment property

Controls the justification of components.

Settings

Valid values	Description
CENTER	Justify the component(s) to the container’s center.
LEFT	Justify the component(s) to the container’s left edge.
RIGHT	Justify the component(s) to the container’s right edge.

Alignment Style property

Positions components within a container using the FlowLayout manager.

Settings

Valid values	Description
ALIGN_CENTERED	(Default) The component is centered within the container.
ALIGN_LEFT	The component is aligned at the left edge of the container.
ALIGN_RIGHT	The component is aligned at the right edge of the container.

Allow Dynamic Moving property

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.

Allow Sorting property

Determines whether the MutliList 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.

Anchor property

This property is a subproperty of Grid Bag Constraints . 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.

The Anchor property is only accessible when the component is inside a container that uses a GridBagLayout layout manager. To access the Anchor property, set the container’s Layout propertyto GridBagLayout.

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.

Arrow Color property

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.

Arrow Indent property

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.

Auto Start property

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.

Background property

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 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.

Bar Color property

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 progess 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.

Bevel Height property

Sets the height of the shadow on any component that derives from Border Panel.

Bevel Style property

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.

Block Increment property

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.

Border Color property

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.

Border Indent property

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.

Border Style property

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.

Border Style property (ImageListBox)

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.

Bordered Style Color property

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.

Bounds properties

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.

Box Bar Style property

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.

Box Width property

If the Box Bar Sytle 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 Box Bar Style 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.

Button Color property

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.

Caret Position property

The position of the text insertion caret. 0 = insert before character #0, 1 = insert before character #1, etc.

Runtime modification

At run time, use the `setCaretPosition` method to modify this property.

Settings

Valid values	Description
0	(Default) Places the insertion caret at the start of the edit text.
Integers >= 0	The location of the text insertion caret.

Case-sensitive property

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.

Cell Background property

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.

Cell Font property

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

```
    "Helvetica",java.awt.Font.PLAIN,12
```

Cell Foreground property

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.

Center Mode property

Determines where within the component area to draw the image.

Settings

Valid values	Description
true	(Default value) Center the image.
false	Image is left-justified and top justified within the component area.

Checkbox property

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.

Class property

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.

Clear Frame property

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.

Column Alignments property

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.

Column Headings property

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.

Columns property

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.

Column Sizes property

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.

Columns to Display property

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.

ComboBox Font property

Defines the font used to display the box elements. 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.

If no font is specified, the font specified in the Font property will be used.

Runtime modification

At run time, use the `setComboBoxFont` method to modify this property.

Settings

This property is empty by default.

ComboBox Mode property

This property determines the hilight action of the ImageListBox. When ComboBox Mode is true list selections will be hilighted 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 `setComboMode` method to modify this property.

Settings

Valid values	Description
true	Items are selected and hilighted when the mouse cursor is over them.
false	Items are selected and hilighted only when the mouse button is pressed.

Cursor property

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.

Date property

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.

Default Directory property

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.

Default File Name property

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.

Delay property

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 StatusScroller)
500	(Default for Animator and MovingAnimation)
integer	The delay in milliseconds.

Direction property

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.

Display Cell Borders property

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.

Drop-down list font property

Defines the font used to display the list items in the drop down list element of a ComboBox 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.

If no font is specified, the font specified in the ComboBox font property will be used. If no font is specified there, the font specified in the Font property will be used.

Runtime modification

At run time, use the `setDropDownFont` method to modify this property.

Settings

This property is empty by default.

Echo property

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.

Editable property

Specifies whether the user can edit selected text on a list in a text-based component, such as TextField or TextArea.

Runtime modification

You can change this property at run time using the `setEditable` method.

Settings

Valid values	Description
true	The user can edit text in the component.
false	The text in the component is read only.

Edit field font property

Defines the font used to display the text in the edit field part of the ComboBox component when the font property is not specified. 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.

If no font is specified, the font specified in the Font property will be used.

Runtime modification

At run time, use the `setEditFieldFont` method to modify this property.

Settings

This property is empty by default.

▶

Edit Font property

Specifies the font style of the editable text in FormattedTextField derived components. 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 `setEditFont` method to modify this property.

Settings

The default value is

```
"Courier",java.awt.Font.PLAIN,12
```

Enabled property

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.

Enforce Minimum Dimension property

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.

Fill Color property

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.

Fill Mode property

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.

Fill property

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.

Font property

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, choose 'Parent.'

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.

Foreground property

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.

Format on Display

Defines the format in which the date, time, or timestamp values are displayed. Text phrases may also be entered as long as they do not contain any of the valid values.

Example: “The date: D* M* D%TH, Y* H%:mm:ss AM“ will print “The date: Friday October 12th , 1973 2:34:54 PM“

Note: If the Format on Display values do not include all the data elements provided by the database, only the values entered into the property list will display. For example, entering “MM/DD” in Format on Display displays “07/01” even though the data from the database is entered as “07/01/1996”

Valid Value	**Description**
DD	Day - 2 digits [01-31]
D%	Day - 1 or 2 digits [1-31]
D+	Day - name of the day abbreviated [Mon.-Sun.]
D*	Day - name of the day complete [Monday-Sunday]
TH	puts the [st-nd-rd-th] element
MM	Month - 2 digits [01-12]
M%	Month - 1 to 2 digits [1-12]
M+	Month - name of the month abbreviated [Jan.-Dec.]
M*	Month - complete name of the month [January-December]
YY	Year - 2 digits
Y*	Year - 4 digits
hh	Hour - 2 digits
h%	Hour - 1 or 2 digits
HH	Hour - 2 digits, 12 hour clock
H%	Hour - 1 or 2 digits, 12 hour clock
AM	adds a AM or PM symbol
mm	Minutes - 2 digits
m%	Minutes - 1 or 2 digits
ss	Seconds - 2 digits
s%	Seconds - 1 or 2 digits
n1,n2…n9	fractions of a second, number specifies the number of digits.

See DBTstamp component, Tstamp component, and Format on Entry.

Format on Entry(dbAWARE only)

Defines the order the user must follow to enter a valid timestamp. It can be just a String containing “M”, ”D” ,”Y”, ”h”, ”m”, ”s” and “n” or a string using the values for Format on Display.

See DBTstamp component, Tstamp component, and Format on Display.

Frame Name property

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

Gap Color property

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.

Gap Width property

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.

Grid Bag Constraints properties

Determines how components are positioned inside a container that uses a GridBagLayout layout manager.   This property is not available unless the Layout property for the component's container is set to GridBagLayout.

The Grid Bag Constraints property applies 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 a separate Grid Bag Constraints property value. This means that unforeseen size and boundary conflicts between components can occur when the container is set to certain sizes and dimensions.   You must manually test your component layout design to determine that all components within a container behave appropriately.

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.

Grid property

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.

Group property

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.

Heading Background property

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.

▸

Heading Font property

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

```
    "Helvetica",java.awt.Font.PLAIN,12
```

Heading Foreground property

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.

Help Menu property

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.

Horizontal and Vertical Gap properties

The Horizontal Gap property is the size of the gap, in pixels, between the contained component (panel) and the horziontal scroll bar. This gap is along the vertical axis and increases the maximum vertical scroll value. The default value is 6.

The Vertical Gap property is the size of the gap, in pixels,   between the contained component (panel) and the vertical scroll bar. This gap is along the horziontal axis and increases the maximum horizontal scroll value. The default value is 6.

Runtime modification

At run time, use the `setHorizontalGap` or `setVerticalGap` method to modify this property.

HTML Link URL property

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.

Image Style property

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	(Default) The image is drawn repeatedly, filling the component.
IMAGE_CENTERED	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 ImageButtons) The image is drawn once in the upper-left corder of the component.

▸

Image URL property

The URL of the image to display while 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 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.

Increment property

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.

Inherit Background property

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 Background property .
false	The component uses its own Background property .

Inherit Font property

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.

Inherit Foreground property

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 Foreground property .
false	The component uses its own Foreground property .

Inset Padding properties

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 Sides

Sets the number of pixels between the 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.

Internal Pad X property

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.

Internal Pad Y property

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.

Items property

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.

Label Alignment property

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.

Label Color property

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.

Label property

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.

Layout property

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 wrinting a class that implements the LayoutManager interface.

Visual Café 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 specifiy the LayoutManager 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.
BorderLayout	Positions components within a container using individual component placement. When you specify BorderLayout, the Placement property 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 Alignment property 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 GridBagLayout 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. GridBagLayout manages a rectangular grid of cells, with each component occupying one or more cells. Components within the GridBagLayout are positioned based on their Grid Bag 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.

Line Thickness property

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.

List Down property

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.

List Items property

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.

Loop Count property

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.

Main MenuBar property

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.

Mask property

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:

To restrict the digit position to:	Use:
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:

Component	Default mask
IntlLongDistPhoneNumber	999/-999/-999/-9999
LocalPhoneNumber	999/-9999
LongZipCode	99999/-9999
PostalCode	A9A/-9A9
SocialInsuranceNumber	999/-999/-999
SocialSecurityNumber	999/-99/-9999
USLongDistPhoneNumber	999/-999/-9999
ZipCode	99999

Runtime modification

At run time, the FormattedTextField component can use the `setMask` method to modify this property.

Settings

Valid values	Description
empty	(Default) The default mask is used.

Maximum and Minimum properties

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.

Maximum property

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.

Max Value property

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.

Menu Shortcut property

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.

Message Hilite Color property

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 hilighted message text black.
blue	Makes the hilighted message text blue.
cyan	Makes the hilighted message text cyan.
darkGray	Makes the hilighted message text dark gray.
gray	Makes the hilighted message text gray.
green	Makes the hilighted message text green.
lightGray	Makes the hilighted message text light gray.
magenta	Makes the hilighted message text magenta.
orange	Makes the hilighted message text orange.
pink	Makes the hilighted message text pink.
red	(Default) Makes the hilighted message text red.
white	Makes the hilighted message text white.
yellow	Makes the hilighted message text yellow.

Message Links property

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.

Messages property

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.

Minimum Height and Width properties

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.

Minimum property

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.

Min Value property

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.

Mode property

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:

Valid values	Description
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.

▸

Mouse Down Image property

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.1

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.

Mouse Over Image property

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.

Multi-Column Mode property

This mode determines whether the ImageListBox grid should be displayed. The grid outlines each cell with a border. When true, the border type property is set to BORDER_NONE and the ComboBox Mode property is set to false.

Runtime modification

Use the `setMultiColumnMode` method to modify this property.

Settings

Valid values	Description
true	Show the ImageListBox grid.
false	(Default) Don’t show the ImageListBox grid.

Multiple Mode property

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.

Multiple Selections property

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.

Name property

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.

Notify Delay property

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.

Notify While Pressed property

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.

Number Of Columns property

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.

Number Of Rows property

The number of rows to initialize the Grid with. This also determines the value returned by `getPreferredSize`.

Runtime modification

At run time, use the `setPreferredRowCount` method to modify this property.

Settings

Valid Values	Description
0	(Default)
integer	The desired number of rows for the Grid to have.

Orientation property (Scrollbar, ToolBarPanel)

Sets the component to display either horizontally or vertically.

Runtime modification

At run time, use the `setOrientation` method to modify this property.

Settings

Valid value	Description
VERTICAL	(Default) Specifies vertical orientation.
HORIZONTAL	Specifies horizontal orientation.

Orientation property (Spinner)

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.

Padding properties

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.

Paused property

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.

Placement property

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

- 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.
- 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.

Positive Slope property

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.

Preview Component property

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.

Propagate Resize property

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.

Repeat property

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.

Repeat Automatically property

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.

Repeat Count property

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.

Repeat Mode property

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.

Resizable property

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.

Right To Left property

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.

Rows property

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.

Rows to Display property

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.

Scale Mode property

Scales an image to fit the component.

Runtime modification

At run time, use the `setScaleMode` method to modify this property.

Settings

Valid values	Description
true	The image is scaled to the size of the component.
false	(Default) The image is not scaled. You must size the component to an appropriate size.

Scroll Clean property

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.

Scroll Direction property

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.

Scroll Increment property

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.

Scroll Interval property

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 >= 30	The delay in milliseconds.

Scroll Unit property

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.

Scrollbar Display Policy property

Scrollbar display options.

Runtime modification

This property cannot be changed after the ScrollPane is constructed.

Settings

Valid values	Description
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.

Scrollbar Visibility property

Scrollbar display options for the TextArea component.

Runtime modification

This property cannot be changed after the TextArea is constructed.

Settings

Valid values	Description
SCROLLBARS_BOTH	(Default) Displays both scrollbars.
SCROLLBARS_HORIZONTAL_ONLY	Displays just the horizontal scrollbar.
SCROLLBARS_VERTICAL_ONLY	Displays just the vertical scrollbar.
SCROLLBARS_NONE	Never displays the scrollbars.

Searchable property

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.

Selected Index property

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).

Selection Background property

Defines the color to use for the background of the selected element in the component.

Runtime modification

At run time, use the `setBgHilite` method to modify this property.

Settings

Valid values	Description
black	Makes the selection background black.
blue	(Default) Makes the selection background blue.
cyan	Makes the selection background cyan.
darkGray	Makes the selection background dark gray.
gray	Makes the selection background gray.
green	Makes the selection background green.
lightGray	Makes the selection background light gray.
magenta	Makes the selection background magenta.
orange	Makes the selection background orange.
pink	Makes the selection background pink.
red	Makes the selection background red.
white	Makes the selection background white.
yellow	Makes the selection background yellow.

Selection Color property

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.

Selection Foreground property

Defines the color to use for the selected element’s text in the component.

Runtime modification

At run time, use the `setFgHilite` method to modify this property.

Settings

Valid values	Description
black	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	(Default) Makes the selection foreground white.
yellow	Makes the selection foreground yellow.

Selection End property

Sets the end position of the currently selected text. 1 = end before character #1, 2 = end before character #2, etc. The value is constrained from being less than the current selection start value.

Runtime modification

At run time, use the `setSelectionEnd` method to modify this property.

Settings

Valid values	Description
0	(Default) Specifies no selection when combined with a selection start of 0.
Integers >= 0	The end position of the currently selected text.

Selection Start property

Sets the start position of the currently selected text. 0 = start before character #0, 1 = start before character #1, etc. The value is constrained from being greater than than the current selection end value.

Runtime modification

At run time, use the `setSelectionStart` method to modify this property.

Settings

Valid values	Description
0	(Default) Specifies no selection when combined with a selection end of 0.
Integers >= 0	The start position of the currently selected text.

Separator property

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.

Shift Offset property

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.

Show Border property

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.

Show Focus property

Displays the component as having the focus when the mouse cursor enters the button display area.

Runtime modification

At run time, use the `setShowFocus` method to modify this property.

Settings

Valid values	Description
true	(Default) Displays the component as having the focus when the mouse is over it.
false	Displays the component as having the focus when the mouse is pressed in it.

Show Horizontal Scrollbar property

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.

Show Percentage property

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.

Show Link URL Status property

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.

Show Vertical Scrollbar property

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.

Space property

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

▶

Standard Image property

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.

State property

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
STATE_UNCHECKED	(Default) The checkbox is unchecked.
STATE_DEFAULT	Third state of three state check box. Box is grayed and checked.
STATE_CHECKED	The checkbox is checked.

Status Text property

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.

Status Text Color property

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.

String property

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.

Style property

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.

Style property (StateCheckBox)

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.

Synchronized Mode property

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.

Tab Labels property

Defines the text strings to use for each tab label in a TabPanel component.

Runtime modification

At run time, use the `setTabPanelLables` 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.

Tabs on Bottom property

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.

Tear Off property

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

Valid values	Description
true	The user can tear off the menu
false	(Default) The menu cannot be torn off.

Text property

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.

Text Alignment property

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	Center text within the label area.
ALIGN_LEFT	(Default) Justify the text to the left edge of the label area.
ALIGN_RIGHT	Justify the text to the right edge of the label area.

Text Color property

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.

Tick Frequency property

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.

Tick Style property

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.

Time property

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 milleseconds. The default is 1000.

Title property

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.

Unit Increment property

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.

URL List property

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 MoveingAnimation 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.

Use 3D Border property

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.

Use Offset property

Determines whether objects in 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.

Value property

Sets the position of the thumb in slider, spinner, progress bar and scrollbar 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.

Vertical Alignment Style property

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.

Visible Amount property

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.

Visible property

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.

Visible Rows property

Sets the number of rows to display in a list. If there are more items in the list than the number of visible rows, a scroll bar is automatically added to the component.

Settings

Any valid integer expression. The default value is 0 (zero).

Weight X property

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 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.

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.

Weight Y property

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 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.

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.

Wrappable property

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.

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 vcafepro.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 vcafepro.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

Event Overview

{button Event list,PI(`vcafe.hlp',`Events_summary')}

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 notifiy the listener of events.

Usage

In Visual Café, 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.

Event listing

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

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	

Window Events (java.awt.event.WindowEvent)

WINDOW_ACTIVATED	WINDOW_CLOSED
WINDOW_CLOSING	WINDOW_DEACTIVATED
WINDOW_DEICONIFIED	WINDOW_ICONIFIED
WINDOW_OPENED	

ACTION_PERFORMED ActionEvent

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.
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.

ADJUSTMENT_VALUE_CHANGED AdjustmentEvent

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:

Adjustment Type	User Action
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.

COMPONENT_ADDED ContainerEvent

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.

COMPONENT_MOVED ComponentEvent

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.

COMPONENT_REMOVED ContainerEvent

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.

COMPONENT_RESIZED ComponentEvent

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.

COMPONENT_SHOWN ComponentEvent

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.

COMPONENT_HIDDEN ComponentEvent

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.

FOCUS_GAINED FocusEvent

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.

FOCUS_LOST FocusEvent

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.

ITEM_STATE_CHANGED ItemEvent

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.

KEY_PRESSED KeyEvent

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.

KEY_RELEASED KeyEvent

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.

KEY_TYPED KeyEvent

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.

MOUSE_CLICKED MouseEvent

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.

MOUSE_DRAGGED MouseEvent

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 untill 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.

MOUSE_ENTERED MouseEvent

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.

MOUSE_EXITED MouseEvent

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.

MOUSE_MOVED MouseEvent

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.

MOUSE_PRESSED MouseEvent

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.

MOUSE_RELEASED MouseEvent

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 `mouseRelased` method. Stop listening for this event by using the event-generating object's `removeMouseListener` method.

TEXT_VALUE_CHANGED TextEvent

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.

WINDOW_ACTIVATED WindowEvent

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.

WINDOW_CLOSED WindowEvent

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.

WINDOW_CLOSING WindowEvent

This event is generated when a windows 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.

WINDOW_DEACTIVATED WindowEvent

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.

WINDOW_DEICONIFIED WindowEvent

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.

WINDOW_ICONIFIED WindowEvent

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.

WINDOW_OPENED WindowEvent

This event is generated when 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.

$ 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.

Project system, including Project Options dialog and project templates

updated 9/30/97: removed PDE features for Pro Free 1.1 release (some deleted, some just hidden)

Using Visual Cafe projects

{button How To,AL(`Creating_a_Project_howto;Opening_a_Project_howto;Saving_Renaming_Copying_a_Project_howto;Migration_howto;Closing_a_project_howto;Drag_and_Drop_Behavior_Project_overview;Opening_Editors_in_Project_howto;Object_Tab_F1;Package_Tab_F1;Files_Tab_F1;Project_Options_dialog_F1;Defining_Visual_Caf_s_Startup_Mode_howto;Automating_project_backups_howto;Defining_a_New_Default_Template_howto;Creating_a_Template_howto',0,`',`')}   {button See Also,AL(`Working_with_Templates_overview;Applets_step_overview;Applications_step_overview;Form_Designer_F1;The_Debugging_Workspace;Differences_between_Applets_and_Applications;Understanding_Workspaces;Using_Visual_Caf_as_Your_Development_Environment;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

The files needed to create one or more Java applets , applications , or related Web pages containing applets are stored in a Visual Cafe *project*. For example, you could store 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, the last project you viewed is opened or a new project is automatically created based on the default project template, Basic 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:

- Basic 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).
- Basic 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.

- Basic Java Bean — A JavaBean that is a round button which changes color when pressed, BeanInfo for the JavaBean, and an applet to view the JavaBean in. There are also sample icon graphics files. This template demonstrates how to create a JavaBeans component.
- Empty Project — No objects.

You can also create your own project templates. See Working with project templates.

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.

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 directories subordinate to the project directory. For example, if your project has many graphics files, you could store these files in a separate directory subordinate to 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**
.vpj	The Visual Cafe project
.vep	Visual Cafe project options and file list
.ve2	Secondary project information
.cdb	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**
.java	A Java source file, such as for an applet
.class	A compiled version of a Java source file
.obj	An intermediate file produced when you compile a Java source file to create a native Win32 application or DLL (Professional edition)

.lib	A library file used with native Win32 applications and DLLs
.html	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 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

You can open multiple projects simultaneously; they are displayed in separate Project windows. This helps you to easily navigate between projects.

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.

▶

Creating a new project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Closing_a_project_howto;Form_Designer_F1;Project_Options_dialog_F1;Adding_a_new_file_to_the_project_howto;Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Adding_Packages_to_a_Project_howto;Creating_a_Template_howto',0,`',`visual')}

The files needed to create one or more Java applets, applications, or related Web pages containing applets are stored in a Visual Cafe project. For example, in a project you could store 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.

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.

▸

Opening an existing project

{button Concepts,AL(`Projects_overview',0,`',`concept')}   {button See Also,Alink(Closing_a_project_howto;Creating_a_Project_howto;Converting_cafe_apps_to_VCafe_howto, , ,visual)}

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**.

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.

▸

Saving, renaming, copying, and deleting a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Closing_a_project_howto;Opening_a_Project_howto;Creating_a_Project_howto',0,`',`visual')}

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 directories subordinate to the project directory. For example, if your project has many graphics files, you could store these files in a separate directory subordinate to 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.

   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.

▸ 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).

Migrating a project from Java version 1.0 to 1.1

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Opening_a_Project_howto;Creating_a_Project_howto;Saving_Renaming_Copying_a_Project_howto',0,`',`visual')}

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 Café version 2.0, Visual Café 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.

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.

# Adding_a_Subproject_howto$ Adding a subprojectK projects, adding subprojects;subprojects;adding, subprojectsA Adding_a_Subproject_howto+ PRJ:0> visual

Closing a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Opening_a_Project_howto;Creating_a_Project_howto',0,`',`visual')}

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 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.

Dragging-and-dropping into the Project window

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Opening_a_Project_howto;Creating_a_Project_howto',0,`',`visual')}

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 form (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 Sharing files between projects.)

Opening editors from the Project window

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Opening_a_Project_howto;Creating_a_Project_howto',0,`',`visual')}

The Project window helps you to quickly edit the items in a project.

To access other editor windows from the Project window:

Double-click…	**To get this editor…**
component	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.

Viewing the components and HTML files in a project

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf',0,`',`concept')}    {button See Also,AL(`Opening_a_Project_howto;Creating_a_Project_howto;Opening_Editors_in_Project_howto;Copying_a_component_howto;Drag_and_Drop_Behavior_FormD_overview;Property_Inspector_using_F1;Designing_and_Building_GUIs_with_Visual_Caf',0,`',`visual')}

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 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

Viewing the packages in a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button How To,AL(`Adding_objects_to_project_howto;Opening_Editors_in_Project_howt;Renaming_an_Object_howto',0,`',`visual')}

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 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.

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

Files_Tab_F1 $▸

Viewing the files in a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button How To,AL(`Adding_objects_to_project_howto;Adding_Packages_to_Visual_Cafe_how_to;Opening_Editors_in_Project_howto;Renaming_an_Object_howto',0,`',`visual')}

Adding a component to a project

{button Concepts,AL(`Projects_overview;Insert_Menu_F1;GUI_development_with_Visual_Caf',0,`',`')}   {button See Also,AL(`Adding_a_Menu_to_a_Form_howto;Adding_Files_to_a_Project_howto;Adding_a_Dialog_howto;Forms_adding_objects ;Drag_and_Drop_Behavior_Project_overview;Designing_and_Building_GUIs_with_Visual_Caf',0,`',`')}

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

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.

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.

Copying a component in a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_objects_to_project_howto;Deleting_Objects_from_a_Project_howto',0,`',`visual')}

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.

Renaming a component in a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_objects_to_project_howto;Forms_adding_objects;Property_Inspector_using_F1',0,`',`visual')}

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, 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.

▸

Deleting a component from a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_objects_to_project_howto',0,`',`visual')}

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.

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 or interactions. Similarly, 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.

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 or interactions.

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 or interactions involving that component.

  If the component is a top-level container, the corresponding Java file is not deleted from the directory on hard disk. You must manually delete it.

▶

Adding a new file to a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto',0,`',`visual')}

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. Click Save.

   The file name appears in the title bar.

5. In the Source window, add appropriate code or text.

Adding an existing file to a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_a_new_file_to_the_project_howto;Deleting_Files_from_a_Project_howto',0,`',`visual')}

You can add HTML, Java, 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.

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.

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.

▸

Deleting a file from a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_a_new_file_to_the_project_howto;Adding_Files_to_a_Project_howto',0,`',`visual')}

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.

▶

Copying a file in a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_a_new_file_to_the_project_howto;Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto',0,`',`visual')}

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 Packages or Files 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.

Sharing files between projects

{button How To,AL(`Copying_a_Project_File_howto;Adding_Files_to_a_Project_howto;Adding_a_new_file_to_the_project_howto;Adding_Bean_to_Library_howto',0,`',`')}     {button See Also,AL(`Projects_overview',0,`',`')}

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.

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.

Adding packages to Visual Cafe

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Package_Tab_F1;Insert_class_dialog_F1;Adding_Bean_to_Library_howto',0,`',`visual')}

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

Setting project options

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Environment_Options_Dialog_F1;Automating_project_backups_howto;Defining_Visual_Caf_s_Startup_Mode_howto',0,`',`visual')}

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 arguments for application execution

Specifying whether to parse imports

Specifying whether to clear messages before builds

Setting compiler options for a project

Specifying file search paths and the output directory

Using version control

Setting debugger options for a project

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.

Defining general project and runtime options

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')}     {button See Also,AL(`F1_Messages_Window;Debugging Your Program;Running_a_Project_howto;Project_Options_dialog_F1',0,`',`visual')}

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 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 arguments for application execution

Specifying whether to parse imports

Specifying whether to clear messages before builds

Specifying whether compiles are debug or final

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')} {button See Also,AL(`Project_Options_dialog_F1;F1_Messages_Window;Debugging Your Program;Running_a_Project_howto',0,`',`visual')}

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.

Setting the project type to applet or application

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications;Java_HTML_and_Visual_Caf',0,`',`concept')}    {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto;;Adding_an_Applet_to_an_HTML_page_howto;Viewing_HTML_Files_howto',0,`',`visual')}

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	The project is an applet. When you run a project, the **Start with Web page** setting determines which HTML file is used. The HTML file determines which applets are run. See Specifying what applets to run and the HTML file. To specify that your applets should run in your default Web browser, select the **Execute applet in default Web browser** option. Deselect it if you want to run applets in the Applet Viewer associated with the Visual Cafe environment. For more information, see Making applets run in the Applet Viewer or a browser.
Application	The project is a standalone application. To run the application from Visual Cafe, the main class must also be specified. See Specifying the main class to run for an application.
	The project is a native, standalone executable.
	The project is a native Dynamic Link Library (DLL).

5. Click OK.

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

▶

Specifying what applets to run and the HTML file

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications;Java_HTML_and_Visual_Caf;HTML_and_Visual_Cafe_overview',0,`',`')}     {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto;Adding_an_Applet_to_an_HTML_page_howto;Viewing_HTML_Files_howto',0,`',`')}

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. 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.

▶

Making applets run in the Applet Viewer or a browser

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications;Java_HTML_and_Visual_Caf;HTML_and_Visual_Cafe_overview',0,`',`concept')}   {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto;Adding_an_Applet_to_an_HTML_page_howto;Viewing_HTML_Files_howto',0,`',`visual')}

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.

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.

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

▶ Associate; on Windows 95 and Windows NT 4.0 and higher, from a Windows file system window (such as the Explorer), choose View
▶ Options
▶ 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.

Specifying the main class to run for an application

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')} {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto',0,`',`visual')}

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 the Basic 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.

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.

Note If you rename a class that appears in this field, Visual Cafe updates the field for you.

# Specifying_the_native_name$ Specifying the name of a native application or DLLK applications, specifying the name of a native executable;native Win32 applications, name;DLL, specifying the library nameA Specifying_the_native_name+ PRJ:0> visual

# Setting_native_working_directory$ Setting the working directory for a native programK applications, specifying the working directory;native Win32 applications, working directory;DLL, specifying the working directoryA Setting_native_working_directory+ PRJ:0> visual

# DLL_debug_program$ Specifying a program for running and debugging a DLLK DLL, specifying program for debuggingA DLL_debug_program> visual

▶

Specifying arguments for running an application

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')}  {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto',0,`',`visual')}

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.

▶

Specifying whether to parse imports

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')}     {button See Also,AL(`Project_Options_dialog_F1;Running_a_Project_howto',0,`',`visual')}

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.

▶

Specifying whether to clear messages before builds

{button Concepts,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`concept')}     {button See Also,AL(`Project_Options_dialog_F1;F1_Messages_Window;Debugging Your Program;Running_a_Project_howto',0,`',`visual')}

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.

▶

Setting compiler options for a project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Project_Options_dialog_F1;F1_Messages_Window;Running_a_Project_howto',0,`',`visual')}

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. (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)

3. Click OK.

   The change takes effect the next time you compile your project.

# exports_native_howtoA exports_native_howto

# link_library_native_howtoA link_library_native_howto

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.
	Suppresses the console window. (Default: disabled)
	Attributes the code with profiling calls so the executable can generate profiling information. (Default: not selected)
	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.

Specifying file search paths and the output directory

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Environment_Options_Dialog_F1;Project_Options_dialog_F1;Package_Tab_F1;Files_Tab_F1',0,`',`visual')}

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.

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.

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.

   A list of directories that can contain source files displays. 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.

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 directory displays. If no directory is specified, Visual Cafe uses the default: a **class** file is placed in the same location as the corresponding **java** file. If you do specify an output directory, all output files are placed in this directory and any package file hierarchy structures are created as well.
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 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 Class files in the Show directories field.

   A list of directories displays. The directory order affects the search order: the topmost directory 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***\java\lib\vecclass.zip**

Setting environment variables in the sc.ini file

{button Concepts,AL(`Environment_customization_overview',0,`',`concept')}     {button See Also,AL(`Environment_Options_Dialog_F1;Project_Options_dialog_F1;Package_Tab_F1;Files_Tab_F1;Specifying_the_File_Search_Path_F1',0,`',`visual')}

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

```
CLASSPATH=.;%@P%\..\JAVA\LIB;%@P%\..\JAVA\LIB
   \SYMCLASS.ZIP;%@P%\..\JAVA\LIB\VECCLASS.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\VECCLASS.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:\visualcafe\resdir. Note that you must have the final backslash (\) for the **%@P%** macro to work.

▸

Using version control

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Environment_Options_Dialog_F1;Project_Options_dialog_F1',0,`',`')}

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.

Setting debugger options for a project

{button Concepts,AL(`Projects_overview;Working_with_Exceptions_in_Java_and_Visual_Caf',0,`',`concept')}     {button See Also,AL(`Project_Options_dialog_F1;Debugging Your Program;Debugging_Remotely;Ending_Remote_Applet_Application_Debugging',0,`',`visual')}

From the Debugger tab of the Project Options dialog box, you can do the following:

- From General, you can enable remote debugging, which allows you to debug a program from a remote site. For more information, see Starting remote applet/application debugging.
- 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.

▸

Defining how exceptions are handled by the debugger

{button Concepts,AL(`Debugging Your Program;Working_with_Exceptions_in_Java_and_Visual_Caf',0,`',`')}     {button See Also,AL(`Project_Options_dialog_F1;Projects_overview',0,`',`')}

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

Working with project templates

{button How To,AL(`Creating_a_Template_howto;Defining_a_New_Default_Template_howto;Deleting_Templates_from_the_Object_Library_howto;Adding_Bean_to_Library_howto',0,`',`visual')}     {button See Also,AL(`Creating_a_Project_howto;Projects_overview',0,`',`')}

When you create a new project, you specify a project template as the starting point. Visual Cafe provides these project templates:

Template	**Description**
Basic Applet	A project that contains a single Applet.
Basic 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).
	The same as basic application, except that the project options are set up to produce a native Win32 executable.
	A simple console application you can use as a starting point. The project options are set up for you.
	A simple DLL you can use as a starting point. The project options are set up for a DLL.
Basic Java Bean	A JavaBean that is a round button which changes color when pressed, BeanInfo for the JavaBean, and an applet to view the JavaBean in. There are also sample icon graphics files. This template demonstrates how to create a JavaBeans component.
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.

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.

Defining a new default template

{button Concepts,AL(`Working_with_Templates_overview;Projects_overview',0,`',`concept')} {button See Also,AL(`Creating_a_Project_howto;Creating_a_Template_howto;Deleting_Templates_from_the_Object_Library_howto',0,`',`visual')}

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.

▶

Creating a project template and adding it to the library

{button Concepts,AL(`Working_with_Templates_overview;Projects_overview',0,`',`concept')}    {button See Also,AL(`New_Project_dialog_F1;Defining_a_New_Default_Template_howto;Deleting_Templates_from_the_Object_Library_howto;Adding_Bean_to_Library_howto;Object_Library_F1',0,`',`')}

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.

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 subordinate 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 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.

▶

Deleting a project template

{button Concepts,AL(`Working_with_Templates_overview;Projects_overview',0,`',`concept')}     {button See Also,AL(`Creating_a_Template_howto;Adding_Bean_to_Library_howto',0,`',`visual')}

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 Window ▶ 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

Managing your projects in Visual Cafe

{button How To,AL(`creating_a_project_howto;adding_objects_to_project_howto;adding_files_to_a_project_howto;deleting_objects_from_a_project_howto;running_a_project_howto;deleting_packages_from_a_project_howto,')} {button See Also,AL(`Form_Designer_F1;The_Debugging_Workspace;Differences_between_Applets_and_Applications;Understanding_Workspaces;Using_Visual_Caf_as_Your_Development_Environment;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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 Basic 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.

Visual Cafe file extensions

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Insert_class_dialog_F1;Adding_Files_to_a_Project_howto;Opening_a_Project_howto',0,`',`visual')}

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

Extension	**Description**
.vpj	The Visual Cafe project
.vep	Visual Cafe project options and file list
.ve2	Secondary project information
.cdb	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**
.java	A Java source file, such as for an applet
.class	A compiled version of a Java source file
.html	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.

▶

Using the Project window

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Creating_a_Project_howto;Drag_and_Drop_Behavior_Project_overview;Adding_a_Dialog_howto',0,`',`visual')}

File ▶ Open File

▶ **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:

Double-click	Opens
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

Moving a file to another project

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Adding_a_new_file_to_the_project_howto;Adding_Files_to_a_Project_howto',0,`',`visual')}

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.

▶

Creating a new document

{button Concepts,AL(`Projects_overview',0,`',`')}     {button See Also,AL(`Adding_a_new_file_to_the_project_howto;Adding_Files_to_a_Project_howto',0,`',`')}

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.

Debugging in Netscape

{button Concepts,AL(`Debugging Your Program',0,`',`')}     {button See Also,AL(`Project_Options_dialog_F1;Projects_overview',0,`',`')}

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

Dragging	From	Into	Effect
A component or **java** 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 **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.
	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 Sharing files between projects.)

Designing forms, including Form Designer, Property List, and layout managers

GUI development with Visual Cafe

{button How To,AL(`Forms_adding_objects;Connecting_Form_components_howto;Creating_a_Form_howto;Designing_and_Building_GUIs_with_Visual_Caf;Property_Inspector_using_F1;arranging_components_on_a_form_howto;adding_a_menu_to_a_form_howto',0,`',`')}     {button See Also,AL(`Animation_and_Multithreading;Form_Designer_F1;Projects_overview;Creating_Interactions;Layout_Managers;Arranging_Objects_with_Layout_Managers;Maintaining_and_Porting_Your_Java_Applets_and_Applications;Object_Library_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Using_the_Menu_Editor_F1;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}

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.
- 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 the Basic Applet project template provided with Visual Cafe, the applet is already set up for you programmatically.

The top-level component for an application with a GUI is the Frame component. The **main** method, usually called from the command line, is the entry point of the application. If you use the Basic 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.

Designing a GUI with Visual Cafe

{button Concepts,AL(`Understanding_Containers;GUI_development_with_Visual_Caf;Arranging_Objects_with_Layout_Managers;Using_the_Menu_Editor_F1;Projects_overview;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}    {button See Also,AL(`Property_Inspector_using_F1;Deleting_Objects_from_a_Project_howto;arranging_components_on_a_form_howto;adding_a_menu_to_a_form_howto;Animation_and_Multithreading;Form_Designer_F1;Creating_Interactions;Layout_Managers;Maintaining_and_Porting_Your_Java_Applets_and_Applications;Object_Library_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components',0,`',`')}

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.

   Details on this step
2. Add components to your forms and arrange them.

   Details on this step
3. Modify component properties.

   Details on this step
4. Create component interactions.

   Details on this step

Understanding containers

{button How To,AL(`Drag_and_Drop_Behavior_FormD_overview;forms_adding_objects;Creating_a_Form_howto;Property_Inspector_using_F1;Arranging_Components_on_a_Form_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Layout Managers;Overview_of_Visual_Caf;Object_Library_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Working_Visually_with_Visual_Caf;Working_with_Windows_Menus_and_Dialogue_Boxes;Arranging_Objects_with_Layout_Managers',0,`',`')}

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.

How Visual Cafe keeps work synchronized

{button How To,AL(`Adding_Files_to_a_Project_howto;Adding_objects_to_project_howto;Creating_a_Project_howto;Forms_adding_objects;Form_Designer_F1;binding_code_to_a_form_component_howto;Arranging_Components_on_a_Form_howto;Designing_and_Building_GUIs_with_Visual_Caf',0,`',`')}     {button See Also,AL(`Property_Inspector_using_F1;Source_Window_F1;Class_Browser_using_F1;Overview_of_Visual_Caf;Understanding_Containers;GUI_development_with_Visual_Caf;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}

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

Editing forms in the Form Designer

{button Concepts,AL(`The_Development_Life_Cycle_a_Two_Step_Process;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Arranging_Objects_with_Layout_Managers',0,`',`')} {button See Also,AL(`Creating_a_Form_howto;Object_Library_F1;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1;Property_Inspector_using_F1;Arranging_Components_on_a_Form_howto',0,`',`')}

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*.

Adding a form to a project

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Forms_adding_objects;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Tabbing_Between_Fields_overview;Displaying_Invisible_Components_howto;Property_Inspector_using_F1;Connecting_Form_components_howto;Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1;Overlapping_components_overview',0,`',`')}

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.

Adding components to a form and arranging them

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')} {button See Also,AL(`Form_Designer_F1;Creating_a_Form_howto;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Tabbing_Between_Fields_overview;Displaying_Invisible_Components_howto;Property_Inspector_using_F1;Connecting_Form_components_howto;Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1;Overlapping_components_overview',0,`',`')}

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.

Details on this task

- Add components to the form.

Details on this task

  You can copy components to add them.

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.

  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*.

Dragging-and-dropping into the Form Designer

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`concept')} {button See Also,AL(`Form_Designer_F1;Creating_a_Form_howto;Forms_adding_objects;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Tabbing_Between_Fields_overview;Displaying_Invisible_Components_howto;Property_Inspector_using_F1;Connecting_Form_components_howto Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1;Overlapping_components_overview',0,`',`')}

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.

Overlapping components in applets

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')} {button See Also,AL(`Form_Designer_F1;Creating_a_Form_howto;Forms_adding_objects;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Tabbing_Between_Fields_overview;Displaying_Invisible_Components_howto;Property_Inspector_using_F1;Connecting_Form_components_howto;Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1',0,`',`')}

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.

Tabbing between fields

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')}     {button See Also,AL(`KeyPressManagerPanel;Form_Designer_F1;Creating_a_Form_howto;Forms_adding_objects;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Displaying_Invisible_Components_howto;Property_Inspector_using_F1;Connecting_Form_components_howto;Overlapping_components_overview;Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1',0,`',`')}

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.

Displaying invisible components

{button Concepts,AL(`Projects_overview;GUI_development_with_Visual_Caf;Understanding_Containers;How_is_my_Work_Kept_In_Sync_overview;Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Creating_a_Form_howto;Forms_adding_objects;Drag_and_Drop_Behavior_FormD_overview;Renaming_an_Object_howto;Deleting_Objects_from_a_Project_howto;Tabbing_Between_Fields_overview;Property_Inspector_using_F1;Connecting_Form_components_howto;Overlapping_components_overview;Deleting_Files_from_a_Project_howto;Designing_and_Building_GUIs_with_Visual_Caf;Using_the_Menu_Editor_F1',0,`',`')}

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.

Note   When you add an invisible component to a form, all invisible components display automatically.

Menu Designer

▸

Menu creation with the Menu Designer

{button How To,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')} {button See Also,AL(`Form_Designer_F1;Component_Properties_overview',0,`',`')}

You can add menu bars to frames and dialogs (they inherit from MenuContainer); applets do not support menu bars. To create a menu bar, you can do the following:

- Add a MenuBar component to a frame or dialog.
- 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 Basic 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.

Using the Menu Designer pop-up menu

{button Concepts,AL(`Using_the_Menu_Editor_F1',0,`',`')} {button See Also,AL(`Menus_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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.

Creating menu bars and menus

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')}     {button See Also,AL(`Popup_menu_MenuEditor_F1;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

You can add menu bars to frames and dialogs (they inherit from MenuContainer); applets 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 or dialog.

   If you used the Basic Application project template, you already have a menu bar in the frame it created for you.

   Details on this step
2. Add menus to your menu bar.

   Details on this step
3. Add menu items to your menus.

   Details on this step

   You can also add menu items that can have submenus.

   Details on this step
4. Edit the menu structure as needed.

   Details on this step
5. Associate command keys with menu items, as needed.

   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.

   Details on this step
7. Bind custom code to menu items, as needed.

   Details on this step

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.

▶

Adding a menu bar to a frame or dialog

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')} {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

You can add the MenuBar component to a frame or dialog 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 or dialog in the Project window and choose Insert Component.
- Copy and paste it within the same project or between projects.

▶ 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.

Adding menus to a menu bar

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')}     {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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.

▶ 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.

Adding menu items to menus

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')}     {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_Menu Bar_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to _a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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 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.

▶ Details on this step

2. In the Property List, set the menu item properties, including the menu item name in the Label property.

   Tip   To add more menu items, you can now select the empty bottom menu item in the list and enter its properties.

Adding submenus

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')} {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

You can add a MenuItem or CheckboxMenuItem component to a MenuItem component.

1. In the Menu Designer, right-click a menu item and choose Create Menu.

   A menu item that can have submenus appears.

2. While the menu item is selected, in the Property List set the submenu 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.

Associating command keys and menu items

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')} {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Editing_menu_structure_howto;Binding_Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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

Editing a menu structure

{button Concepts,AL(`Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')} {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_Menu Bar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Binding _Code_to_a_Menu_Command_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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.

  Note If you delete a component from a container, you must manually delete any custom code or interactions involving that component.

Binding code to a menu item

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Using_the_Menu_Editor_F1;Component_Properties_overview',0,`',`')} {button See Also,AL(`Popup_menu_MenuEditor_F1;Editing_Menus_howto;Adding_a_Menu_to_a_Form_howto;Adding_a_Menu_to_a_MenuBar_howto;Adding_a_MenuItem_to_a_Menu_howto;Adding_a_Submenu_howto;Adding_menu_command_keys_howto;Editing_menu_structure_howto;Property_Inspector_using_F1;Object_Library_F1',0,`',`')}

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

▶

Working with component properties

{button How To,AL(`Property_Inspector_using_F1',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Properties_summary;Object_Library_F1;Projects_overview;Arranging_Objects_with_Layout_Managers;Arranging_Components_on_a_Form_howto',0,`',`')}

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.

To display the Property List

- Choose Window ▶ Property List.

▸

Viewing and changing component properties

{button Concepts,AL(`Component_Properties_overview;Arranging_Objects_with_Layout_Managers',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Properties_summary;Object_Library_F1;Projects_overview;Arranging_Components_on_a_Form_howto',0,`',`')}

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 Window ▸ 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 dialog box.
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

Working with layout managers

{button How To,AL(`Property_Inspector_using_F1;arranging_components_on_a_form_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Creating_Interactions;Projects_overview;Overview_of_Visual_Caf;Object_Library_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Layout property',0,`',`')}

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.

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.

▸

Arranging components in the Form Designer

{button Concepts,AL(`Arranging_Objects_with_Layout_Managers;GUI_development_with_Visual_Caf',0,`',`')} {button See Also,AL(`Layout property;Forms_adding_objects;Layout_Menu_F1;Form_Designer_F1;Using_the_Grid_Options_Dialog_F1;Arranging_Components_on_a_Form_howto;Property_Inspector_using_F1;Form_Designer_F1',0,`',`')}

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 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 Position 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 position 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. 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 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

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. 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.

Manipulating the Form Designer grid

{button Concepts,AL(`Arranging_Objects_with_Layout_Managers;GUI_development_with_Visual_Caf',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Property_Inspector_using_F1;Arranging_Components_on_a_Form_howto;Layout_Menu_F1',0,`',`')
}

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

Changing component properties

{button Concepts,AL(`Component_Properties_overview;Property_Inspector_using_F1')}     {button See Also,AL(`Properties_summary;Property_Inspector_using_F1;Arranging_Components_on_a_Form_howto',0,`',`')}

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 Form Designer or from the 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.

Changing the component displayed in the Property List

{button Concepts,AL(`Component_Properties_overview',0,`',`')}  {button See Also,AL(`Property_Inspector_using_F1',0,`',`')}

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.

Viewing multiple property sets simultaneously

{button Concepts,AL(`Property_Inspector_using_F1',0,`',`')}     {button See Also,AL(`Property_Inspector_using_F1',0,`',`')}

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.

Modifying an object's properties

{button Concepts,AL(`Property_Inspector_using_F1',0,`',`')}     {button See Also,AL(`Connecting_Form_components_howto;Forms_adding_objects',0,`',`')}

Property values are changed in the Property List.

1. In the Form Designer, select an object.
2. Choose Windows ▶ 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.

Understanding the Component Library

{button How To,AL(`binding_code_to_a_form_component_howto;deleting_objects_from_the_object_library_howto;Adding_Bean_to_Library_howto;adding_objects_to_the_palette_howto;deleting_objects_from_the_palette;Palette_F1',0,`',`')} {button See Also,AL(`Form_Designer_F1;Projects_overview;Overview_of_Visual_Caf;Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}

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.

Creating and adding components to forms

{button How To,AL(`form_designer_f1;Adding_Code_to_a_Java_Source_File_howto;binding_code_to_a_form_component_howto;displaying_invisible_components_howto;creating_a_form_howto',0,`',`')}     {button See Also,AL(`Creating_Interactions;Layout_Managers;Projects_overview;Understanding_Visual_Components;Understanding_Workspaces;Working_Visually_with_Visual_Caf;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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.

Copying components

{button Concepts,AL(`Form_Designer_F1',0,`',`')}     {button See Also,AL(`Forms_adding_objects;Connecting_Form_components_howto',0,`',`')}

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.

Moving components between forms

{button Concepts,AL(`Form_Designer_F1;Projects_overview',0,`',`')} {button See Also,AL(`Forms_adding_objects;Property_Inspector_using_F1;Connecting_Form_components_howto;Copying_a_component_howto;Form_Designer_F1',0,`',`')}

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.

Deleting components from a form

{button Concepts,AL(`Form_Designer_F1',0,`',`')}     {button See Also,AL(`Forms_adding_objects;Deleting_a_connection_howto',0,`',`')}

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

Editing a form

{button Concepts,AL(`Form_Designer_F1',0,`',`')}     {button See Also,AL(`Deleting_Objects_from_a_Project_howto',0,`',`')}

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

▸

Copying a menu

{button Concepts,AL(`Form_Designer_F1',0,`',`')}     {button See Also,AL(`Adding_a_Menu_to_a_Form_howto;Copying_a_component_howto;Forms_adding_objects;Connecting_Form_components_howto;Property_Inspector_using_F1',0,`',`')}

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.

Importing foreign files into Visual Cafe (Adding an existing file to a project and Adding packages to Visual Cafe are in project.doc)

updated 9/29/97:   revised for Free 1.1 release

Importing Cafe projects into Visual Cafe

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Closing_a_project_howto;Opening_a_Project_howto;Creating_a_Project_howto',0,`',`visual')}

Your Cafe project is converted automatically when you open it with Visual Cafe.

Warning   Converted projects cannot be saved back into Cafe format. However, you can move Java source files between Cafe and Visual Cafe.

# Importing_VJ++_howto$ Importing Visual J++ projects into Visual CafeK Visual J++, importing projects;projects, importing, Visual J++;importing, Visual J++ projects;converting, Visual J++ projectsA Importing_VJ++_howto+ PRJ:0> visual

Considerations_import_J++_concept$ Considerations when importing Visual J++ projectsK Visual J++, importing projects;projects, importing, Visual J++;importing, Visual J++ projects;converting, Visual J++ projectsA Considerations_import_J++_concept+ PRJ:0> concept

	Visual J++ configuration that sets it…
	The
	The
	The
	The
	The

	Visual J++ configuration that sets it…
	The
	A
	The

Interactions

Understanding component interactions

{button How To,AL(`Form_Designer_F1;Class_Browser_using_F1;Source_Window_F1;Using_the_Interaction_Wizard_F1;Connecting_Form_components_howto;Deleting_a_connection_howto;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto;Changing_a_connection_howto',0,`',`')}
{button See Also,AL(`Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment;Projects_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}

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 reponds 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 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 must manually delete all of 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*.add*type*Listener or *object*.add*type*Adapter) 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);
    }
}
```

Using the Interaction Wizard

{button Concepts,AL(`Understanding_Component_Connections_overview',0,`',`')}     {button See Also,AL(`Deleting_a_connection_howto;Changing_a_connection_howto;Connecting_Form_components_howto;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto',0,`',`')}

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

Creating a component interaction

{button Concepts,AL(`Understanding_Component_Connections_overview;Projects_overview',0,`',`')} {button See Also,AL(`Changing_a_connection_howto;Deleting_a_connection_howto;Using_the_Interaction_Wizard_F1;Palette_F1;Form_Designer_F1;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto',0,`',`')}

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, select the trigger component by using one of these methods:
   - Right-click the component and select Add Interaction.
   - Click the Interaction tool icon ▶, then click the trigger component.
2. Drag a line from the trigger component to the action component within the same project.

   The action component is highlighted to help you identify which component you selected. If a component will not highlight, the interaction could be inappropriate.

   To connect a component to itself, double-click the component.

   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.
3. 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.
4. In the Start an interaction field, select the event that activates the interaction.
5. 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.
6. In the second window, specify the interaction condition.

   Tip A variable must be of the type listed for the second 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.
7. 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*.add*type*Listener or *object*.add*type*Adapter) 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.

Changing an existing interaction

{button Concepts,AL(`Understanding_Component_Connections_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}    {button See Also,AL(`Connecting_Form_components_howto;Deleting_a_connection_howto;Using_the_Interaction_Wizard_F1;Class_Browser_using_F1;Source_Window_F1;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto',0,`',`')}

Interactions are implemented as methods in the container class of a component. The interaction methods are called event handlers.

1. Make the project active by clicking the Project window.
2. Choose Window ▶ Class Browser

   You should see a listing of project classes 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 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.

   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_Action(java.awt.event.ActionEvent event)
   {
       // to do:  code goes here.

       //{{CONNECTION
       // Go to the SlideShow's next image
       VacationSlides.nextImage();
       //}}
   }
   ```
6. In the editing pane, make any source code changes you need.

▶ Details on this step

Deleting an interaction

{button Concepts,AL(`Understanding_Component_Connections_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}  {button See Also,AL(`Connecting_Form_components_howto;Changing_a_connection_howto;Using_the_Interaction_Wizard_F1;Class_Browser_using_F1;Source_Window_F1;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto',0,`',`')}

To delete an interaction, you delete the interaction in the event handler, as well as the applicable portions of the listener registration and 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_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);
    }
}
```

To delete the event handler in the Class Browser

1. Make the project active by moving focus to the project's Project window.

2. Choose Window ▶ 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. Delete the interaction in the event handler code. Or delete 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 delete the event handler in the Source window

1. Open the source file containing the event handler in the Source window.

▶ 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. Delete the interaction in the event handler code. Or delete the entire event handler if it only handles one interaction.

To 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*.add*type*Listener or *object*.add*type*Adapter call for that component, if the listener registration is not used by another event tied to that component.

Obsolete

Working with interactions

{button How To,AL(`connecting_form_components_howto;connecting_classes_howto;binding_code_to_a_menu_command_howto;binding_code_to_a_form_component_howto;adding_an_event_to_component_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Understanding_Workspaces;Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}

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 reponds 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.

Creating an interaction between a form and component

{button Concepts,AL(`Understanding_Component_Connections_overview;Palette_F1;Form_Designer_F1;Projects_overview',0,`',`')}
{button See Also,AL(`Connecting_Form_components_howto;Changing_a_connection_howto;Deleting_a_connection_howto;Using_the_Interaction_Wizard_F1',0,`',`')}

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

Running, compiling, executing

▶
Running a project

{button Concepts,AL(`Understanding_the_Visual_Caf_Debugger;The_Debugging_Workspace_and_Tools;Debugging_Applets_and_Applications',0,`',`')} {button See Also,AL(`Debugging Your Program;Controlling_Compiler_Information_F1;Setting_the_release_type;Project_Options_dialog_F1',0,`',`')}

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.

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.

The main class in bytecode applications

{button How To,AL(`Project_Options_dialog_F1;Running_a_Project_howto',0,`',`')}   {button See Also,AL(`Projects_overview;The_Differences_between_Applets_and_Applications',0,`',`')}

The *main class* is the name of the class with a **main** method. For a bytecode Java application, the main class is the starting point of execution.

- When you run a bytecode application from the command line, you type the name of the Java file that contains the **main** method as the first argument to java.exe. . See Specifying the main class to run for an application 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.

Compiling from the command line

{button Concepts,AL(`Projects_overview',0,`',`concept')}     {button See Also,AL(`Project_Options_dialog_F1;F1_Messages_Window;Running_a_Project_howto',0,`',`visual')}

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.

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 *file*.cdb	Generate compilation database *file*.cdb. 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 **cdb** 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 *path*	Use *path* instead of the setting of the environment variable CLASSPATH.
-d *outputdir*	Set the output directory. The default is to put **class** files with their respective **java** files.
-debug	When the compiler is Sun's compiler, javac, using this option causes the compiler to report diagnostic messages about its own execution. When the compiler is SJ, the switch is ignored.
-depend *file*.dep	Generate dependencies into *file*.dep. You can read it to see what classes your files need to deploy.
-g	Add debug information to output files so they can be debugged.
-help	Print help on compiler switches.
-j *codepage*	Override the system codepage default. Asian language character sets include double-byte characters, where certain prefix bytes mean that the following byte forms part of the character. Asian language characters can appear in "" strings, in '' character literals, and in comments. The codepage guides the compiler in converting from the ASCII character set used in the source

	java files into Unicode. The compiler normally uses the system default codepage as a guide for translating double-byte character sets. 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.

Description

Generate native x86 object files.

Link generated object files into outputname.[exe|dll]. This command supplies the same functionality as the obj switch, plus runs the linker.

Specify the main class for the native executable. mainclass is a fully qualified class name. (If no main switch is specified, it will default to outputname.)

Generate Windows-only executable (no console).

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.

Specify classes or packages to be exported from the DLL, so they are accessible from other executables. package|class can be either a fully qualified class name or a package specification (for example, java.awt.*).

Add debug information to output files so that the result can be debugged. Use file.tdb as the debug "type database" file. This is the format required to debug in the Symantec Visual Cafe environment.

Generate Pentium (p5) or Pentium Pro (p6) code.

Do performance profiling.

Pass /switch to the linker. To see the valid options, type link /?.

Link in the library file. Used with the link switch.

Link in the resource file. Used with the link switch.

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 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
```

Deployment

Deploying applets

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')} {button See Also,AL(`Deploying_Applications_how_to;Determining_What_Class_Files_an_Applet_or_Application_Needs_how_to;Configuring_UNIX-based_Web_Servers_how_to',0,`',`')}

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.

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. Instead, you want to deploy using just the classes needed by the components in your applets. These classes can be packaged in a JAR file or not.

To deploy your applets in a JAR

1. Use the Project ▸ JAR command to create a JAR file containing your applets and all supporting files, including Symantec **class** files.

▸ Details on this step

   Tip Your applets should use relative URLs for graphics files so you can easily move your applets to different computers.

2. Add the variable `ARCHIVE="`*`name`*`.jar"` to the applet tags in your HTML files. You can specify multiple JAR files by delimiting them with a comma (,).

   Tip 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.

To deploy your applets outside of a JAR

Here is one way:

1. Use the Project ▸ JAR command to create a JAR file containing your applets and all supporting files, including Symantec **class** files.

▸ Details on this step

   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.

▸ Details on this step

   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 (,).

4. Copy the HTML files for your Web pages into the new directory.

   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.

   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.)

▶ 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 (,).
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.

Deploying applications

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')}  {button See Also,AL(`Deploying_Applets_how_to;Determining_What_Class_Files_an_Applet_or_Application_Needs_how_to;Configuring_UNIX-based_Web_Servers_how_to',0,`',`')}

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 ▸ JAR command to create a JAR file containing your application and all supporting files, including Symantec **class** files.

▸ Details on this step

   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 JAR file 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.

   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.

To deploy your application outside of a JAR

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.

▸ Details on this step

   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.

Determining what class files an applet or application needs

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')} {button See Also,AL(`Deploying_Applets_how_to;Deploying_Applications_how_to;Configuring_UNIX-based_Web_Servers_how_to',0,`',`')}

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 by choosing Project ▸ JAR, 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 Creating a JAR file and 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.

Configuring UNIX-based Web servers

{button Concepts,AL(`Using_JAR_files_overview;Object_Library_F1;Projects_overview',0,`',`')} {button See Also,AL(`Deploying_Applications_how_to;Deploying_Applets_how_to;Determining_What_Class_Files_an_Applet_or_Application_Needs_how_to',0,`',`')}

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.

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.

Details on this step

3. Create a symbolic link from /home/symantecclasses/symantec to the user's home HTML directory. For example:

   ```
   ln -s /home/symantecclasses/symantec /home/joeuser/public_html
   ```

   joeuser can now run applets from Web pages without the need to copy the full Symantec class structure to his public_html directory.

HTML and VC

Using HTML files in the Visual Cafe environment

{button How To,AL(`viewing_html_files_howto;Applets_step_overview;Adding_an_Applet_to_an_HTML_page_howto;Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto;Specifying_what_applets_to_run;Making_applets_run_in_viewer_or_browser;Adding_a_new_file_to_the_project_howto',0,`',`')}     {button See Also,AL(`Projects_overview;Overview_of_Visual_Caf;Differences_between_Applets_and_Applications;Using_Visual_Caf_as_Your_Development_Environment;Working_Visually_with_Visual_Caf;The_Development_Life_Cycle_A_Two_Step_Process;Java_HTML_and_Visual_Caf',0,`',`')}

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.

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.

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.

The relationship between Java and HTML

{button How To,AL(`viewing_html_files_howto;Applets_step_overview;Adding_an_Applet_to_an_HTML_page_howto;Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto;Specifying_what_applets_to_run;Making_applets_run_in_viewer_or_browser;Adding_a_new_file_to_the_project_howto',0,`',`')}    {button See Also,AL(`HTML_and_Visual_Cafe_overview;Projects_overview;Overview_of_Visual_Caf;Differences_between_Applets_and_Applications;Using_Visual_Caf_as_Your_Development_Environment;Working_Visually_with_Visual_Caf;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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.

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.

Viewing and editing HTML files

{button Concepts,AL(`HTML_and_Visual_Cafe_overview;Java_HTML_and_Visual_Caf;Projects_overview',0,`',`')}    {button See Also,AL(`Applets_step_overview;Adding_an_Applet_to_an_HTML_page_howto;Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto;Specifying_what_applets_to_run;Making_applets_run_in_viewer_or_browser;Adding_a_new_file_to_the_project_howto',0,`',`')}

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 view), double-click the HTML file name.
   - In the Project window (Object 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.

2. View the file and make your edits in the Source window.

▸ Details on using the Source window

3. Save the file by choosing File ▸ Save.

Adding an applet to an HTML page

{button Concepts,AL(`HTML_and_Visual_Cafe_overview;Java_HTML_and_Visual_Caf;Projects_overview',0,`',`')}     {button See Also,AL(`Applets_step_overview;Adding_Files_to_a_Project_howto;Deleting_Files_from_a_Project_howto;Viewing_HTML_Files_howto;Specifying_what_applets_to_run;Making_applets_run_in_viewer_or_browser;Adding_a_new_file_to_the_project_howto',0,`',`')}

You can add an applet to an HTML page by adding an applet tag to the HTML code.

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.

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

Tools for working with source code

{button How To,AL(`Adding_Code_to_a_Java_Source_File_howto;binding_code_to_a_form_component_howto;class_browser_using_f1;hierarchy_editor_f1;Source_Window_F1;Editing_SearchReplace_overview;Migrating_10_to_11_event_model_howto',0,`',`')}
{button See Also,AL(`visual_Cafe_tools_overview;Form_Designer_F1;Object_Library_F1;Understanding_Workspaces;Working_with_Basic_User_Interface_Components;Understanding_Component_Connections_overview;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}

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.

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.

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

▸

Using the Source window

{button How To,AL(`Customizing_your_enviro_appearance_Display_tab_F1_howto;Defining_auto_filetype_formatting_F1_howto;Adding_Code_to_a_Java_Source_File_howto',0,`',`')} {button See Also,AL(`Debugging Your Program;Form_Designer_F1;Projects_overview;Property_Inspector_using_F1;Insert_class_dialog_F1;Visual_Caf_Tools_for_Working_with_Classes_and_Components;Hierarchy_Editor_F1;Class_Browser_using_F1;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}

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.
- Double-click in the Form Designer.
- 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.

   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.

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

Using the Format Options dialog box

{button Concepts,AL(`Debugging Your Program;Source_Window_F1',0,`',`')}     {button See Also,AL(`GoTo_Buffer_Dialog_F1',0,`',`')}

Sourc++e ▸ Format Options
Source

▸ **Go to Buffer**
▸ **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 Defining Automatic File Type Formatting 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.

Class Browser

▶

Using the Class Browser

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Adding_Code_to_a_Java_Source_File_howto',0,`',`')}     {button See Also,AL(`Popup_menu_Class_Pane_F1;Popup_menu_Member_Pane_F1;Source_Window_F1;Hierarchy_Editor_F1;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}

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.

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 Window ▶ Class Browser.

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.

Using the Member Attributes dialog box

{button Concepts,AL(`Class_Browser_using_F1',0,`',`')}     {button See Also,AL(`Insert_class_dialog_F1',0,`',`')}

Classes ▸ 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

▸

Using the Hierarchy Editor

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Adding_Code_to_a_Java_Source_File_howto',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;How_is_my_Work_Kept_In_Sync_overview',0,`',`')}

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.

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 Window ▸ 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.

▸ 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.

Using ZIP files in Visual Cafe

{button How To,AL(`Adding_Files_to_a_Project_howto',0,`',`concept')}     {button See Also,AL(`Deleting_Files_from_a_Project_howto;Projects_overview',0,`',`')}

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

Adding code to a Java source file

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto',0,`',`')}

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.

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 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, only use code syntax that matches what Visual Cafe can generate, or 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.

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.

Caution   Custom code and interactions are not deleted from the source file when you delete a component. You must manually maintain any file that contains custom code or interactions.

Binding code to a component

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Editing_Event_Methods_howto;Connecting_Form_components_howto;Adding_Code_to_a_Java_Source_File_howto', 0,`',`')}

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*.add*type*Listener or *object*.add*type*Adapter) after the comment tag REGISTER_LISTENERS. If the adapter/listener class has already been instantiated, it is used.
- Fourth, an event handler is generated. If the event handler already exists, it is used.

If instead you use the Interaction Wizard to create an interaction, Visual Cafe makes the four modifications mentioned previously, 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.
   - Double-click in the Form Designer while the form with the component is displayed.
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.

Binding KeyPress events

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Button_component;Editing_Menus_howto;Adding_Code_to_a_Java_Source_File_howto',0,`',`')}

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("A Basic 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

▸

Searching and replacing in the Source window

{button How To,AL(`Find_dialog_F1;Replace_dialog_F1',0,`',`')} {button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto;Find_in_Files_dialog_F1',0,`',`')}

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 string you want to search for 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.

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.

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
▸ Environment Options
▸ 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.

▸ Details on this step

To set a bookmark

- Choose Search ▸ Bookmarks and complete the dialog box.

▸ 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.

▸ Details on this step

3. Set advanced search options as needed.

▸ 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

Using the Find dialog box

{button See Also,AL(`Find_in_Files_dialog_F1',0,`',`')}

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, **in** matches **in**, but not **include**.
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.

Using the Replace dialog box

{button See Also,AL(`Find_in_Files_dialog_F1',0,`',`')}

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.

▸

Using the Go to Line dialog box

{button Concepts,AL(`Debugging Your Program;Source_Window_F1',0,`',`')}     {button See Also,AL(`Goto_Function_F1;Bookmark_window_F1;Find_dialog_F1;Search_Menu_F1',0,`',`')}

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.

Using the Go to Function dialog box

{button Concepts,AL(`Debugging Your Program;Source_Window_F1',0,`',`')}     {button See Also,AL(`Goto_Line_Window_F1;Bookmark_window_F1;Find_dialog_F1;Search_Menu_F1',0,`',`')}

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.

▸

Using the Go To Buffer dialog box

{button Concepts,AL(`Debugging Your Program;Source_Window_F1',0,`',`')}     {button See Also,AL(`Bookmark_window_F1',0,`',`')}

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

▸

Using the Bookmark dialog box

{button Concepts,AL(`Debugging Your Program;Source_Window_F1',0,`',`')}     {button See Also,AL(`Find_dialog_F1;Goto_Function_F1;Goto_Line_Window_F1',0,`',`')}

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.

▸

Using the Find in Files dialog box

{button See Also,AL(`Find_dialog_F1',0,`',`')}

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)

Specifying the search files and location

{button See Also,AL(`Find_in_Files_dialog_F1;Setting_Advanced_Search_Criteria_howto',0,`',`')}

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.

To	Do
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.

To	Do
Expand a search into subfolders	Select the Search subfolders option.

Setting advanced search criteria

{button See Also,AL(`Find_in_Files_dialog_F1',0,`',`')}

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.

Using the Compare Files dialog box

{button See Also,AL(`Find_dialog_F1;Find_in_Files_dialog_F1;Replace_dialog_F1',0,`',`')}

Search ▸ 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

Selecting text

{button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto',0,`',`')}

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.

Toggling overtype and insert typing modes

{button How To,AL(`Defining_auto_filetype_formatting_F1_howto',0,`',`')} {button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1',0,`',`')}

Source windows support two typing modes: overtype, which replaces characters as you type; and insert, which adds new characters to the file.

To toggle the typing mode

- Press the INSERT key to toggle between overtype and insert modes.

  The typing mode is displayed in the status bar.

Note A change in typing mode applies to all open Source windows, not just to the active window.

▸

Indenting code automatically

{button How To,AL(`Defining_auto_filetype_formatting_F1_howto',0,`',`')}    {button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Format_Options_Dialog_F1',0,`',`')}

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.

▶

Wrapping lines of code

{button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto;Format_Options_Dialog_F1',0,`',`')}

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

▸

Finding a class or class definition

{button Concepts,AL(`Projects_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Source_Window_F1;Class_Browser_using_F1;Hierarchy_Editor_F1;Files_Tab_F1;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto',0,`',`')}

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 ▸ Find to look in the current file.
- Choose Search ▸ Find in Files to look in multiple files.

▸ 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 **-**).
  - 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.

  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.

▸ 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.

▸ 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.

Adding a new class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Finding_a_class_howto',0,`',`')}

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.

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.

Editing a class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Adding_a_class_howto;Finding_a_class_howto',0,`',`')}

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.

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.

Using the Class Wizard

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}    {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto',0,`',`')}

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 class or Interface if you are creating an interface.
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 package 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 inner classes only.
Final or Abstract	Select Final class or Abstract class.

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. 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

▶

Copying or moving a class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto',0,`',`')}

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.)

Renaming a class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto',0,`',`')}

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.

▸

Deleting a class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto',0,`',`')}

You can delete a class from the Source window.

- Select the code, then press DELETE or choose Edit ▸ Delete.

▸

Changing a class inheritance

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Insert_class_dialog_F1;Adding_Code_to_a_Java_Source_File_howto;Viewing_Imports_Hierarchy_Editor_howto;Class_Display_in_Class_Browser_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto;Hierarchy_Editor_F1',0,`',`')}

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.

▸

Controlling class display in the Class Browser

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Insert_class_dialog_F1;Adding_Code_to_a_Java_Source_File_howto;;Viewing_Imports_Hierarchy_Editor_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto;Hierarchy_Editor_F1',0,`',`')}

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 where you want to show imported classes.

Viewing imports in the Hierarchy Editor

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Insert_class_dialog_F1;Adding_Code_to_a_Java_Source_File_howto;Class_Display_in_Class_Browser_howto;Deleting_a_Class_Inheritance_howto;Deleting_a_class_howto;Renaming_a_class_howto;Copying_a_class_howto;Insert_class_dialog_F1;Editing_a_class_howto;Adding_a_class_howto;Finding_a_class_howto;Hierarchy_Editor_F1',0,`',`')}

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.

Controlling which classes and members display

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`WinStyle_Controlling_Inherit_Method_Display_F1;WinStyle_Grouping_and_Sorting_Class_and_Members_F1;Adding_Code_to_a_Java_Source_File_howto;Class_Browser_using_F1',0,`',`')}

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.

Grouping and sorting classes and members

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`WinStyle_Controlling_Class_and_Member_Display_F1;WinStyle_Controlling_Inherit_Method_Display_F1;Adding_Code_to_a_Java_Source_File_howto;Class_Browser_using_F1',0,`',`')}

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.

Members

▸

Finding a method or data variable

{button Concepts,AL(`Projects_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Source_Window_F1;Class_Browser_using_F1;Hierarchy_Editor_F1;Files_Tab_F1;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Deleting_a_member_howto;Renaming_a_member_howto;Copying_a_member_howto;Editing_a_member_howto;Adding_methods_ClassBrowser_howto',0,`',`')}

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.

   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.

▸ 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 ▸ Find to look in the current file.
- Choose Search ▸ Find in Files to look in multiple files.

▸ Details on this task

To locate a method or data variable in the Class Browser

1. Select a class in the Classes pane.

▸ 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).

   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.

Adding a method or data variable

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Source_Window_F1;Class_Browser_using_F1;Hierarchy_Editor_F1;Adding_methods_ClassBrowser_howto;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Deleting_a_member_howto;Renaming_a_member_howto;Copying_a_member_howto;Editing_a_member_howto;Finding_a_member_howto',0,`',`')}

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.

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.

   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*.add*type*Listener or *object*.add*type*Adapter) after the comment tag REGISTER_LISTENERS. If the adapter/listener class has already been instantiated, it is used.
- Fourth, an event handler is generated. If the event handler already exists, it is used.

If instead you use the Interaction Wizard to create an interaction, Visual Cafe makes the four modifications mentioned previously, 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.

▸

Editing a method or data variable

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Deleting_a_member_howto;Renaming_a_member_howto;Copying_a_member_howto;Adding_methods_ClassBrowser_howto;Finding_a_member_howto',0,`',`')}

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.

  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.

▸ 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.

▸ Details on this step

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 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.

▸ Details on finding a member

▸

Copying or moving a method or data variable

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Deleting_a_member_howto;Renaming_a_member_howto;Editing_a_member_howto;Adding_methods_ClassBrowser_howto;Finding_a_member_howto',0,`',`')}

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.)

Renaming a method or data variable

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Deleting_a_member_howto;Copying_a_member_howto;Editing_a_member_howto;Adding_methods_ClassBrowser_howto;Finding_a_member_howto',0,`',`')}

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.

Details on finding a class

Details on finding a member

2. When an edit box appears, type the new name or edit the existing name.

   The constructor is renamed for you.

To rename a member in the Source window

- Find the member and rename it, then search for its name and rename all occurrences.

Details on finding a member

▸

Deleting a method or data variable

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview;Adding_Code_to_a_Java_Source_File_howto;Member_Display_in_Class_Browser_howto;Renaming_a_member_howto;Copying_a_member_howto;Editing_a_member_howto;Adding_methods_ClassBrowser_howto;Finding_a_member_howto',0,`',`')}

You can delete a member from the Class Browser or Source window.

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 the contains the member.

▸ Details on this step

2. Select a member or shift-click multiple members 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 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.

   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.

Controlling member display in the Class Browser

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`Insert_class_dialog_F1;Adding_Code_to_a_Java_Source_File_howto;Class_Browser_using_F1;Deleting_a_member_howto;Renaming_a_member_howto;Copying_a_member_howto;Editing_a_member_howto;Adding_methods_ClassBrowser_howto;Finding_a_member_howto',0,`',`')}

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.

▸
Controlling the display of inherited methods

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}   {button See Also,AL(`WinStyle_Controlling_Class_and_Member_Display_F1;WinStyle_Grouping_and_Sorting_Class_and_Members_F1;Adding_Code_to_a_Java_Source_File_howto;Class_Browser_using_F1',0,`',`')}

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

Adding an event handler to a component

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto;Adding_Code_to_a_Java_Source_File_howto;Editing_Event_Methods_howto;Deleting_an_Event_Handler_howto;Listener_Accessing_Outside_howto;Migrating_10_to_11_event_model_howto',0,`',`')}

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*.add*type*Listener or *object*.add*type*Adapter) after the comment tag REGISTER_LISTENERS. If the adapter/listener class has already been instantiated, it is used.
- Fourth, an event handler is generated. If the event handler already exists, it is used.

If instead you use the Interaction Wizard to create an interaction, Visual Cafe makes the four modifications mentioned previously, 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.

Editing an event handler

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Visual_Caf_Tools_for_Working_with_Classes_and_Components;Adding_Code_to_a_Java_Source_File_howto;Adding_an_Event_to_Component_howto;Deleting_an_Event_Handler_howto;Listener_Accessing_Outside_howto;Migrating_10_to_11_event_model_howto',0,`',`')}

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.

Details on finding a class
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.

Deleting an event handler

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto;Adding_Code_to_a_Java_Source_File_howto;Adding_an_Event_to_Component_howto;Editing_Event_Methods_howto;Listener_Accessing_Outside_howto;Migrating_10_to_11_event_model_howto',0,`',`')}

Deleting an event-handler is the same as deleting an interaction. See Using the Interaction Wizard for more information.

Migrating a Java file from the 1.0 to 1.1 event model

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto;Migration_howto;Adding_Code_to_a_Java_Source_File_howto;Adding_an_Event_to_Component_howto;Editing_Event_Methods_howto;Deleting_an_Event_Handler_howto;Listener_Accessing_Outside_howto',0,`',`')}

Visual Cafe has a command that automatically converts a Java source file from the 1.0 event model to the 1.1 event model.

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.

Accessing a listener from outside an init method or application constructor

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto;Migration_howto;Adding_Code_to_a_Java_Source_File_howto;Migrating_10_to_11_event_model_howto',0,`',`')}

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 lSymAction 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

Understanding basic object-oriented programming concepts

{button How To,AL(`adding_methods_to_a_class_howto;adding_methods_classbrower_howto;Adding_methods_ClassBrowser_howto;Insert_class_dialog_F1;adding_a_subclass_from_the_class_browser_howto;adding_a_subclass_from_the_hierarchy_editor_howto;inserting_a_custom_class_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Object_Library_F1;Differences_between_Applets_and_Applications;Using_Visual_Caf_as_Your_Development_Environment;Working_Visually_with_Visual_Caf;Working_with_Basic_User_Interface_Components',0,`',`')}

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

▶

Using the Classes Pane

{button How To,AL(`WinStyle_Grouping_and_Sorting_Class_and_Members_F1',0,`',`')}     {button See Also,AL(`Source_Window_F1;Class_Browser_using_F1;Popup_menu_Class_Pane_F1',0,`',`')}

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.

Using the Members Pane

{button How To,AL(`WinStyle_Grouping_and_Sorting_Class_and_Members_F1',0,`',`')}     {button See Also,AL(`Source_Window_F1;Class_Browser_using_F1;Popup_menu_Member_Pane_F1',0,`',`')}

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.

Visually working with project classes

{button How To,AL(`Class_Browser_using_F1;Hierarchy_Editor_F1',0,`',`')}     {button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}

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.

Adding a class

{button Concepts,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Visual_Caf_Tools_for_Working_with_Classes_and_Components;Working_with_the_Source_Editor',0,`',`')} {button See Also,AL(`Class_Browser_using_F1;Source_Window_F1;Hierarchy_Editor_F1;Projects_overview',0,`',`')}

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 class or Interface if you are creating an interface.
Name	Type the name of the class or interface.
Source	Type the complete path to the class file. <<I don't get the Edit Class File dialog box.>>
Package	Choose a package 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. <<Is this working right?>>
Final or Abstract	Select Final class or Abstract class.

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

Adding a subclass from the Class Browser

{button See Also,AL(`Adding_a_subclass_from_the_Hierarchy_Editor_howto;Adding_a_Subclass_from_the_Project_window_howto',0,`',`')}

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.

Adding a subclass from the Hierarchy Editor

{button See Also,AL(`Adding_a_subclass_from_the_Class_Browser_howto;Adding_a_Subclass_from_the_Project_window_howto',0,`',`')}

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.

Details on this step

▸

Adding a subclass from the Project Window

{button See Also,AL(`Adding_a_subclass_from_the_Class_Browser_howto;Adding_a_Subclass_from_the_Hierarchy_Editor_howto',0,`',`')}

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

Working with the Source Editor

{button How To,AL(`Customizing_Editing_windows_Editing_tab_F1',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Projects_overview;Overview_of_Visual_Caf;Object_Library_F1',0,`',`')}

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.

Viewing a component's Java source

{button Concepts,AL(`Adding_Code_to_a_Java_Source_File_howto;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Source_Editor_F1;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto',0,`',`')}

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.

▸

Moving around in a file

{button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto;Format_Options_Dialog_F1;Goto_Line_Window_F1;Goto_Function_F1',0,`',`')}

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.

▶

Searching through and comparing multiple files

{button How To,AL(`Find_in_Files_dialog_F1;Compare_Files_Dialog_F1',0,`',`')}     {button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto',0,`',`')}

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 Search ▶ Compare Files.

Using the Class Attributes dialog box

{button Concepts,AL(`Class_Browser_using_F1',0,`',`')}     {button See Also,AL(`Insert_class_dialog_F1',0,`',`')}

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???>>

Adding a method from the Source window

{button Concepts,AL(`Source_Window_F1;Adding_Code_to_a_Java_Source_File_howto',0,`',`')} {button See Also,AL(`Adding_methods_ClassBrowser_howto;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto',0,`',`')}

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.

Adding a method to a class

{button Concepts,AL(`Adding_Code_to_a_Java_Source_File_howto',0,`',`')}     {button See Also,AL(`Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto',0,`',`')}

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 windowAdding_methods_SourceEditor_howto

Editing source code

{button How To,AL(`Adding_Code_to_a_Java_Source_File_howto;Adding_Files_to_a_Project_howto;Adding_methods_to_a_class_howto',0, `',`')}    {button See Also,AL(`Debugging Your Program;Source_Menu_F1;Source_Window_F1',0,`',`')}

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

Enhancing Java code for a component

{button See Also,AL(`Insert_class_dialog_F1;Adding_Code_to_a_Java_Source_File_howto;Adding_methods_to_a_class_howto;Binding_Code_to_a_Form_Component_howto;Binding_Code_to_a_Menu_Command_howto',0,`',`')}

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

Dragging-and-dropping in the Source window

{button How To,AL(`Projects_overview;Source_Window_F1',0,`',`')}

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?>>

Dragging	**Into**	**Effect**
Code in the Source window	Windows operating system	Creates a **src** 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

9/29/97: revised for Free 1.1 release

Welcome to Visual Cafe

{button How To,AL(`Applets_step_overview;Applications_step_overview',0,`',`')}     {button Concepts,AL(`Overview_of_Visual_Caf;Working_Visually;Visual_Caf_as_a_Development_Environment;The_Differences_between_Applets_and_Applications;GUI_development_with_Visual_Caf',0,`',`')}   {button See Also,AL(`Visual_Cafe_Tools_overview;How_is_my_Work_Kept_In_Sync_overview;How_WYSIWYG_is_Visual_Cafe_overview;Workspace_customization_overview',0,`',`')}

Symantec Visual Cafe is the first 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.

When developing an applet or application in Visual Cafe, you work mainly with projects.

With Visual Cafe, you can

- assemble an applet
- assemble an application

Overview of Visual Cafe

{button How To,AL(`applets_step_overview;applications_step_overview;section_overview',0,`',`')} {button See Also,AL(`Java_HTML_and_Visual_Caf;Projects_overview;The_Differences_between_Applets_and_Applications;Understanding_Visual_Components;Working_Visually;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

Symantec Visual Cafe for Windows is the first 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 and stand-alone Java applications.

Featuring a sophisticated set of high level, two-way tools, Visual Cafe is the leading Java development environment:

- Form-centric development environment with a powerful integrated form design tool.
- Configurable widget palette contains your favorite form components.
- Component Library contains a repository of components for forms, windows, and databases.
- Supports add-in third party components via the extensible Component Library.
- Interaction Wizard allows you to specify actions based on events.
- Form and property sheet modifications generate real-time source code.
- Variables window incorporates a tree view to collapse and expand the various scopes.

Symantec's Java compiler and the Visual Cafe environment

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 Iavac.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.

Visual Cafe tools

{button Concepts,AL(`Debugging;Understanding_the_Visual_Caf_Debugger;The_Debugging_Workspace_and_Tools;Debugging_Applets_and_Applications',0,`',`')} {button See Also,AL(`Visual_Caf_Tools_for_Working_with_Classes_and_Components;Visual_Cafe_window_F1;Projects_overview;Form_Designer_F1;Palette_F1;Using_the_Menu_Editor_F1;Using_the_Interaction_Wizard_F1;Property_Inspector_using_F1;Source_Window_F1;Object_Library_F1;Class_Browser_using_F1;Hierarchy_Editor_F1',0,`',`')}

Visual Cafe provides several tools to help you create forms. With Visual Cafe, you add visual components to your forms to 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.

The main Visual Cafe tools are:

- Visual Cafe window
- Project window
- Form Designer
- Source window
- Class Browser
- Debugger
- Component Palette
- Menu Editor
- Interaction Wizard
- Property List
- Component Library
- Hierarchy Editor

Working visually with Visual Cafe

{button How To,AL(`customizing_your_enviro_appearance_display_tab_f1_howto;customizing_editing_windows_editing_tab_f1;environment_customization_overview;',0,`',`')} {button See Also,AL(`Projects_overview;Overview_of_Visual_Caf;Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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.

Using Visual Cafe as your development environment

{button How To,AL(`customizing_your_enviro_appearance_display_tab_f1_howto;customizing_editing_windows_editing_tab_f1;environment_customization_overview',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Projects_overview;Overview_of_Visual_Caf;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment;Working_Visually_with_Visual_Caf;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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.

The development life cycle: a two-step process

{button How To,AL(`applets_step_overview;applications_step_overview;',0,`',`')}     {button See Also,AL(`projects_overview;Projects_overview;Overview_of_Visual_Caf;Differences_between_Applets_and_Applications;Using_Visual_Caf_as_Your_Development_Environment;Working_Visually_with_Visual_Caf',0,`',`')}

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.

The differences between applets and applications

{button How To,AL(`converting_cafe_apps_to_vcafe_howto;converting_an_applet_to_an_application_howto;Applications_step_overview;applets_step_overview;testing_an_applet_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Java_HTML_and_Visual_Caf;Projects_overview;Overview_of_Visual_Caf;Understanding_Visual_Components;Visual_Caf_as_a_Development_Environment;Working_Visually;The_Development_Life_Cycle_A_Two_Step_Process',0,`',`')}

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.

Understanding workspaces

{button How To,AL(`workspaces_aud_howto',0,`',`')}     {button See Also,AL(`Form_Designer_F1;Projects_overview;Overview_of_Visual_Caf;Differences_between_Applets_and_Applications;Understanding_Visual_Components;Using_Visual_Caf_as_Your_Development_Environment;Working_with_Basic_User_Interface_Components',0,`',`')}

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!

▸

Assembling applets

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;The_Differences_between_Applets_and_Applications',0,`',`')}     {button See Also,AL(`Object_Library_F1',0,`',`')}

An applet can add dynamic, interactive functionality to your Web page. Visual Cafe makes it extremely 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 the Basic Applet project template.

▸ Details on this step

2. Design the user interface.

▸ Details on this step

3. Enhance Java source code.

▸ Details on this step

4. Set project options.

   If you started with the Basic Applet project template, you probably do not need to set project options.

▸ Details on this step

5. Run the applet in Visual Cafe.

▸ Details on this step

6. Debug the applet, if needed.

▸ Details on this step

7. Add the applet to your HTML page.

▸ Details on this step

8. Run the applet in the HTML page.

▸ Details on this step

9. Deploy the applet.

▸ Details on this step

▸

Assembling applications

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;The_Differences_between_Applets_and_Applications',0,`',`')}  {button See Also,AL(`Object_Library_F1',0,`',`')}

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 the Basic Application project template.

▸ Details on this step

2. Design the user interface.

▸ Details on this step

3. Enhance Java source code.

▸ Details on this step

4. Set project options.

   If you started with the Basic Application project template, you probably do not need to set project options.

▸ Details on this step

5. Test run the application in Visual Cafe.

▸ Details on this step

6. Debug the application, if needed.

▸ Details on this step

7. Deploy the application.

▸ Details on this step

9/29/97 - revised for Visual Café Pro Free 1.1

Environment Options

Development environment customization

{button See Also,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}

You can customize the Visual Cafe development environment by:

Setting environment options

Setting project options

Modifying workspaces

▸

Visual Cafe workspaces

{button How To,AL(`Workspaces_AUD_howto',0,`',`')}     {button Concepts,AL(`Understanding_Workspaces;The_Debugging_Workspace_and_Tools',0,`',`')}

A large number of views are available in the Visual Cafe environment. Visual Cafe lets you save a configuration of windows as an 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.

▶

Modifying workspaces

{button Concepts,AL(`Workspace_customization_overview;Understanding_Workspaces',0,`',`')}     {button See Also,AL(`Toolbars_workspace_F1',0,`',`')}

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
▶ *Workspacename*.
- 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
▶ 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
▶ 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.

Note   You cannot delete the last remaining workspace.

- Choose Window ▶ Workspaces
▶ Delete.

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

Setting environment options

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Project_Options_dialog_F1',0,`',`')}

Tools ▸ Environment Options

Visual Cafe allows you to set options that are global to all of your projects.

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

Setting the database environment options

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

Setting environment options in the General tab

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button How To,AL(`Defining_the_Help_File_Set_howto;Finding_Java_source_howtoDefining_Visual_Caf_s_Startup_Mode_howto;Specifying_the_File_Search_Path_F1;Setting_Environment_Variables_in_scini_howto',0,`',`')}

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

To open the Environment Options General view

- Choose Tools ▸ Environment Options

▸ General tab.

▶

Defining the Visual Cafe startup mode

{button Concepts,AL(`Projects_overview',0,`',`')}     {button See Also,AL(`Creating_a_Project_howto;Environment_Options_Dialog_F1',0,`',`visual')}

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.

▶

Finding Java source files

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Setting_General_Environment_Options_General_tab_F1;Specifying_the_File_Search_Path_F1;Setting_Environment_Variables_in_scini_howto;Environment_Options_Dialog_F1',0,`',`')}

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.

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.

   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.

▶

Defining the Help file set

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}     {button See Also,AL(`Setting_General_Environment_Options_General_tab_F1;Environment_Options_Dialog_F1',0,`',`')}

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
▶ General tab.
2. In the Help Files field, enter the **hlp** file names.

   Note   These directories are automatically searched by Visual Cafe: the installation directory, and the release \bin and \help directories.

▸

Setting debugging options for the environment

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Incremental_debugging_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Defining_the_Help_File_Set_howto;Defining_Visual_Caf_s_Startup_Mode_howto;Environment_Options_Dialog_F1',0,`',`')}

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 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.

Setting the database environment options

{button See Also,AL(`dbNAVIGATOR_pop_up_menu;dbNAVIGATOR_window',0,`',`')}

When you install Visual Café Pro, a Database tab is added to the Environment Options dialog box. You use this screen to set the components to be associated with particular data types. The components in the Database tab are used in the Choose Database Columns window of the dbAWARE Project Wizard and by dbNAVIGATOR.

To use the Database Environment Options dialog box setting:

1. Choose Tools ▸ Environment Options.

   The Environment Options dialog box displays.
2. Click on the Database tab of the Environment Options.

   The Database Environment Options display.
3. Click on the Component field of any data type to see the drop-down list of component data type options.
4. Select the appropriate data type for your component.

▶

Specifying text formatting for Visual Cafe windows

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Format_Options_Dialog_F1;Custom_keywords_dialog_F1;Customizing_your_enviro_appearance_Display_tab_F1_howto;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1',0,`',`')}

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 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.
Tab width	Specifies the number of columns between tab

	stops (1-16).The default is four character widths. This value may be overridden locally in each buffer.
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.

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.

- Click Edit Custom.

  The Custom Keywords dialog box opens so that you can enter the new keywords.

▸

Using the Enter New File Extension dialog box

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}     {button See Also,AL(`Defining_auto_filetype_formatting_F1_howto;Environment_Options_Dialog_F1',0,`',`')}

Environment Options ▸ Format tab

▸ **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.

▸

Using the Custom Keywords dialog box

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}     {button See Also,AL(`Defining_auto_filetype_formatting_F1_howto;Environment_Options_Dialog_F1',0,`',`')}

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.

▸

Mapping Visual Cafe commands to key sequences

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}    {button See Also,AL(`Format_Options_Dialog_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto;Mapping_commmands_to_keyboard_F1_howto;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1',0,`',`')}

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 macro*filename.* ScriptMaker macro files, stored in \Bin\Macs, 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.

Note   Help on the commands is available in the Macro help. This help is available when you are editing macros. For example, choose Tools ▸ Macro
▸ ScriptMaker, then click Edit; in the window that displays, press F1 to get the Macro help contents.

In the Keyboard view of the Environment Options, 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.

   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.

   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.

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

▶

Specifying key editing options for the Source Editor

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}  {button See Also,AL(`Format_Options_Dialog_F1;Customizing_Editing_windows_Editing_tab_F1;Defining_auto_filetype_formatting_F1_howto;Mapping_commmands_to_keyboard_F1_howto;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1',0,`',`')}

These settings apply to all Source windows and panes.

1. Choose Tools ▶ Environment Options

▶ 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.

▶

Using the Save Key Bindings dialog box

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')}  {button See Also,AL(`Mapping_commmands_to_keyboard_F1_howto;Environment_Options_Dialog_F1',0,`',`')}

Environment Options ▶ Keyboard tab

▶ Save

Use the Save Key Binding dialog box to save a **key** file.

Customizing the font and color in Visual Cafe windows

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Customizing_Editing_windows_Editing_tab_F1;Customizing_Source_Window_editing_howto;Setting_the_cursor_style_howto;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1;Hierarchy_Editor_F1',0,`',`')}

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

1. 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.

Specifying backup and save options

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')} {button See Also,AL(`Automating_project_backups_howto;Setting_the_Scope_of_the_Undo_Command_howto;Environment_Options_Dialog_F1',0,`',`')}

Visual Cafe provides several backup and save options:

Saving files for recovery purposes

Automating source file backups

Setting the scope of the Undo Command

▸

Saving files for recovery purposes

{button Concepts,AL(`Projects_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Defining_Save_and_Backup_Options_Saving_Tab_F1;Environment_Options_Dialog_F1',0,`',`visual')}

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
▸ 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.

▸

Automating source file backups

{button Concepts,AL(`Projects_overview;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Defining_Save_and_Backup_Options_Saving_Tab_F1;Environment_Options_Dialog_F1',0,`',`visual')}

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.

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
▸ 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 *file*.bak.
Copy to directory	Copy the source files to the specified directory. You can type the directory with the full path into the text box, or click **…** to select a directory from a dialog box. There is no default.
Invoke OnBackup script	Run your own macro, created with the Visual Cafe macro utility and containing an **OnBackupFile** statement. All macros are placed in the \VisualCafe\Bin\Macs directory.

▸

Setting the scope of the Undo command

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Defining_Save_and_Backup_Options_Saving_Tab_F1;Environment_Options_Dialog_F1',0,`',`')}

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
▸ 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.

Specifying code editing options

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}  {button See Also,AL(`Customizing_Editing_windows_Editing_tab_F1;Customizing_Source_Window_editing_howto;Setting_the_cursor_style_howto;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1;Hierarchy_Editor_F1',0,`',`')}

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 show horizontal scroll bars

- Select this option to display a horizontal scroll bar at the bottom of Source windows and panes. (It is selected by default.)

▶

Setting the cursor style

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment;Visual_Caf_Tools_for_Working_with_Classes_and_Components',0,`',`')}     {button See Also,AL(`Customizing_Editing_windows_Editing_tab_F1;Environment_Options_Dialog_F1;Source_Window_F1;Class_Browser_using_F1',0,`',`')}

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.

Controlling toolbar position and visibility

{button Concepts,AL(`Palette_F1;Toolbars_standard_F1',0,`',`')}     {button See Also,AL(`Customizing_the_Palette_from_the_Palette_Tab_F1;Environment_Options_Dialog_F1',0,`',`')}

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.

▸

Enabling ValueTips

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')} {button See Also,AL(`Setting_General_Environment_Options_General_tab_F1',0,`',`')}

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.

▶

Customizing Class Browser and Class Hierarchy window editing

{button Concepts,AL(`Class_Browser_using_F1;Hierarchy_Editor_F1',0,`',`')}     {button See Also,AL(`Customizing_Editing_windows_Editing_tab_F1;Visual_Caf_Tools_for_Working_with_Classes_and_Components;Source_Window_F1',0,`',`')}

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.

▶

Changing editor properties

{button Concepts,AL(`Using_Visual_Caf_as_Your_Development_Environment',0,`',`')} {button See Also,AL(`Format_Options_Dialog_F1;Defining_auto_filetype_formatting_F1_howto;Customizing_your_enviro_appearance_Display_tab_F1_howto;Environment_Options_Dialog_F1',0,`',`')}

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 ▶ Environment Options. Local changes can be made to an editing session by using the Format Options dialog box, Source

▶ 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.