

HELP CONTENTS

Applets:

[Previewing the Applet](#)

[Previewing the Applet using Netscape's Navigator](#)

Compiling:

[Compiler Error Messages](#)

[Compiling the Applet](#)

[.java Source File](#)

[Output Directory](#)

[Using Third Party Compilers](#)

Components:

[Adding a Component in the Coder](#)

[Adding an Interface to a Component](#)

[Changing Component Declarations](#)

[Initializing a Component](#)

[Making a Component for Reuse](#)

[Removing a Component](#)

Declarations:

[Component Referencing Model](#)

[Regular Java Declarations](#)

Functions:

[Adding a Function](#)

[Overriding a Function](#)

[Removing a Function](#)

[Updating the Code](#)

General:

[Dual Hierarchies](#)

[Importing a Package](#)

[Mojo Directory Structure](#)

[Mojo Package Structure](#)

Dual Hierarchies

Mojo uses two distinct hierarchical classifications -- a class hierarchy which is arranged into packages and a container hierarchy which organizes your project.

The Class Hierarchy (maroon colored), also referred to as the Component Hierarchy, is where all Java classes and available components reside. It can be accessed by pressing the Hierarchy Toggle button found on the left-hand side of the browser window. The Class hierarchy is arranged by packages. For instance, the Button class is found in:

```
java
  awt
    Button
```

The Project hierarchy (navy colored) is where all the instances of a class are stored. Programming in Mojo is accomplished by changing the attributes of instances. Associated classes are automatically generated for each instance. For example, you can change the behavior of a particular button, i.e. Button1, by overriding its action event or its paint method. Unlike raw Java, you do not need to create a new class for every component for which you wish to have a particular behavior. The programmer can concentrate purely on extending the functionality of what they have inserted into their project.

Adding a Component in the Coder

Slide open the Class hierarchy (maroon) by pressing the Hierarchy Toggle button. Browse and find the component in the Class Hierarchy which you wish to make an instance of. For example, if you wanted to add an instance of a Stack, you would browse through java -> util -> Stack and click on Stack. Now, click on the component in the Project hierarchy where you want to add the instances as a child of. Drag-and-drop the Stack component into the Project. A dialog will appear asking for the instance name. After you provide an instance name and click on **Ok**, an instance of Stack with the name you provided will be added to your project.

Removing a Component

To remove a component from either hierarchy, click on the Component in the name in the Available Components or Current Project panel. Then select Remove Component in the Component Menu. A short-cut is to use the right mouse button in the component or project panel for the pop-up menu. Removing a component will remove a component and all of its subcomponents as well as all subclasses of the component.

Overriding a Function

Synopsis: *Right-click on the Method and Event list to bring up an override list.*

To override a function in Mojo, select the component by clicking on it in the hierarchy. Then, press the right mouse button in the event and method list panel. A pop-up menu will appear. Select the Override option to bring up a function override list. If the override list is short, another pop-up menu will appear with all function headers which you can override (grouped by inherited components). If the override list is long, a separate dialog box will appear showing all inherited functions. Double-clicking on a header will override the function.

When a function is overridden, a skeleton of the function is loaded in the code editor.

Adding a Function

Synopsis: Add a function to a component using the Add Event or Add Method options under the Functions menu.

To add a function, select the component which you want to add the function to. Then, click on the Add Event or Add Method option under the Functions. Whether you are adding an event or method depends on which option is selected on the tab under the Method and Event List. When you add a method or event, you have the option to make the method public, private, or protected using radio buttons in the Add Function dialog box. A shortcut to add a function is to right-click in the Available Functions pane.

Removing a Function

Synopsis: Remove a function from a component using the Remove Event or Remove Method option under the Functions menu.

To remove a function, select the function header in the Method and Event List. Select the Remove Event or Remove Method from the Function menu to remove the function. A shortcut to add a function is to right-click in the Available Functions pane.

Updating the Code

Synopsis: *Use <CTRL>-U to update your code.*

Mojo uses a Smalltalk-like browser which has two important code editing commands – update and revert. When you type code into Mojo, it is temporarily stored in a buffer. In order for Mojo to incorporate what you have typed, you need to update your code by pressing <CTRL>-U in the Code Editor. Clicking away to another panel or issuing a command also updates the code. Until you update the code, you have the option to revert your code back to the last update by pressing <CTRL>-R in the Code Editor.

Changing Component Declarations

Synopsis: *Use the components declarations to add instance variables and class variables.*

When you select a component by clicking on the component name in the Program or Component hierarchy, the components declarations appear. A components declarations store instance and class variables. You can type directly into the declarations in the Code Editor. After you update the code, the declarations will appear in the Program Declarations pane. *Note: Do not put the { or the } in the declarations, as they are automatically generated by the Mojo.*

Component Referencing Model

Synopsis: Use the Mojo reserved words **parent** and **applet** to reference other components.

Referencing components in Mojo is significantly easier than using pure Java. There are two primary keywords used for accessing parts of your applet:

```
applet  <- currently running applet
parent  <- parent of the object
```

For instance to find out where the applet was started from, you could call:

```
applet.getCodeBase();
```

For the given hierarchy below, you can reference another components methods and instances variables using the parent-reference model:

```
--+
|
+- Panel1
   |
   +- Button1
   |
   +- Button2
```

In the action event of Button2, you can call the action event of the Button1 with:

```
Button2:
    public boolean action(Event evt, Object what)
    {
        parent.Button1.action(evt, what);

        return(true);
    }
```

Using parent, you are able to access all parts of the Project hierarchy.

Initializing a Component

Synopsis: *Place initializations in the **void initialize()** method.*

In a visual object environment like Mojo, certain operations need to be done in the constructor. This is accomplished by the Mojo code generator. The code generator creates a special constructor which sets the default property values and handles the Component Referencing Model. However, there are instances when a programmer might wish to have code which is run as part of the initialization of a component. This is accomplished using the **void initialize()** method. Any code added to the initialize() method of a component will be run after all other initializations of the component. You can add the **void initialize()** method to a component by overriding it in the override list of a method or by using the Add function option under the Functions menu in the coder.

Importing a Package

Synopsis: Use the Attributes option under the Component menu to add or remove import statements for a particular component.

The common Java model for importing a package is done on a global level. In Mojo, packages can be imported on a component by component basis. Select the component to add an interface for. Select on the Attributes option under the Components menu. This will bring up the attributes dialog box. Click on the Imports tab. To add a package, click on the Add button. To remove a package, select the package to remove from the list and click on the Remove button. Click on the Attribute dialogs Ok button to accept all changes for the component.

Adding an Interface to a Component

Synopsis: Use the Attributes option under the Component menu to add or remove interface statements for a particular component.

Component can be made to implement a particular interface by adding an interface using the Attributes dialog (see Importing a Package). Interfaces are found under the Implements tab in the Attributes dialog. In the declaration **public MyApplet extends Applet implements Runnable**, the interface to add is **Runnable**. Java allows you to add more than one interface for each component.

Regular Java Declarations

Synopsis: Use the **Project** declarations to enter all straight Java declarations.

Most Java programs are arranged using flat files. Mojo makes programming simpler by arranging the program based on hierarchies. This divides a program into small functional units which are easier to manipulate. To allow programmers absolute flexibility, Mojo allows them to type in straight flat-file Java code.

Clicking the Project node in the Program hierarchy brings up the declarations for the entire project. The Code Editor now acts as a flat file editor for Java code. Class and Interface declarations as well as extra import lines can be typed into the Project declarations.

The Project declarations appear before any other Java code is generated.

Compiling the Applet

Synopsis: *Compile the Applet using Compile Applet option under the Project menu.*

Compiling an applet creates a .java file from the current Mojo project and compiles it using the compiler specified in the Project Preferences. Compile the Applet using the Compile Applet option under the Project menu.

Compiler Error Messages

Synopsis: If an error in the code is found during compilation and error message will be displayed in the Coder Status bar.

Compiler errors will attempt to jump to the code where the error occurred. Since Mojo is in part a visual environment, some error messages do not correspond to functions, methods or declarations. In these cases, an error dialog will appear showing all error messages. Double-clicking on an error moves the cursor to the corresponding error location in the .java source file.

Previewing the Applet

Synopsis: *Preview applets using Appletviewer.*

The compiled applet can be viewed in using the Preview Applet option under the Project menu. This will bring up Appletviewer setting the appropriate CLASSPATH. You can change the location of the appletviewer in the Project Preferences dialog (use the Project option under the Preferences menu).

Previewing the Applet using Netscapes Navigator

Synopsis: Preview applets using Netscapes Navigator or any other browser by changing the Netscape Browser path in the Preferences dialog.

Use the Project option under the Preferences menu to bring up the Project Preferences dialog. Under the Browser tab, you can browse for your Netscape Navigator. After you have selected a browser, select Preview in Browser under the Project menu. This will check the Preview in Browser menu option. Now, when you go to Preview Applet, it will preview using the Netscape Navigator. This also applies for Running an Applet as well. To revert to using the Appletviewer, uncheck the Preview in Browser menu option by selecting it again.

Making a Component for Reuse

Synopsis: *Make components out of parts of your project for the ultimate in reuse.*

As you build a Mojo project, each part of the project can be made into a component for future reuse. If the component made is visual, it will appear in the User-Defined tab of your component palette of the Designer as a drag-and-dropable component. Otherwise, the component will be added to the user package in the Class hierarchy.

To make a component, either select it in the Designer or in the Program Hierarchy of the Coder. Then select the Make Component option from the Component menu. A dialog box will appear asking for the name and default package for the new component. *Note: The name of the new component must have a different name than the original component.*

If you make a component from a panel which contains other subcomponents, all the subcomponents will be made into components for future use as well; however, only the master component will appear on the palette. For example, an interface for a VCR control which has a reverse and forward button will create one VCR component which will include both the reverse and forward buttons which are accessible in the Class hierarchy.

Output Directory

Synopsis: *All .class, .java, .HTM, and resources are stored in the output directory of the current project.*

Mojo creates an output directory directory containing your resources which were gathered during development. This directory is located below your saved project file directory (<projectname>.output).

For example, a project named **MyApplet** stored in the directory **c:\applets\special**, creates the following directory structure:

c:\applets\special\MyApplet.prj	<- project file
c:\applets\special\MyApplet.def	<- project definition file
c:\applets\special\MyApplet.output\	<- project output directory

To put an applet onto a website, transfer all the files in the output directory, except .BMP files and the .java source, to the site. Depending on whether or not third party components are used, you may need to transfer additional .class or .zip files to your site as well. Please refer to special documentation on third party components to see which .class or .zip files need to be transferred to make a particular component work on the net.

.java Source File

Synopsis: A flat .java file is produced during compilation of your applet or application and is found in the output directory.

Mojo generates a complete .java file when it compiles or runs your applet or application. This source file can be found in the output directory.

Common uses for the .java is to use it in a third party editor, or for complex debugging or understanding how Mojo generates your Java program.

Mojo Directory Structure

Synopsis: *A listing of the Mojo directory structure.*

The following directories are assumed to branch off of the Mojo install directory:

<i>mojo_install_path</i>	Mojo executable
\images	.gif image library
\java	JDK
\lib	Mojo library files
\plugins	3rd party component definition files
\classes	3rd party component classes
\user	User-defined component (Make Component)
\examples	Example projects
\sounds	.au sound library
\docs	Off-line documentation
\shareware	Shareware components and documentation
\myproject	Temporary project until save

Mojo Package Structure

Synopsis: *Mojo components are arranged into a package view in the Class hierarchy.*

Mojo packages are arranged into abstract packages. Java classes are arranged into a package called *java* and organized in the same way as the package import specification of the JDK. Mojo components are in the *mojo* package. Third party components are placed in the *plugins* package. User-defined components (created with Make Component) are placed in the *user* package.

A package/component/subcomponent model is used to organize components. For example, a component made by a user called MyVCRComponent which is under the *mediaplayers* category will be stored in the following manner:

```
user
  MyVCRComponent
  subcomponents
    myvcrcomponent
      ForwardButton
      ReverseButton
      PauseButton
      PlayButton
      StopButton
```

Note: all subcomponents for the VCR control are stored under subcomponents in another package heading named by the same name.

Special Note: Package names should be all lowercase by convention, and Class names should start with a capital letter.

Using Third Party Compilers

Synopsis: Mojo allows third party compilers to be plugged in using the Project option under the Preferences menu.

Mojo allows the programmer to select any Java compiler of their choice. Currently, two compiler error models are support Suns javac.exe and Symmantics sj.exe. The compiler defaults to Suns javac. To change the compiler, select the Project option under the Preferences menu. The default Java compiler and CLASSPATH can be changed in the Compiler tab.

